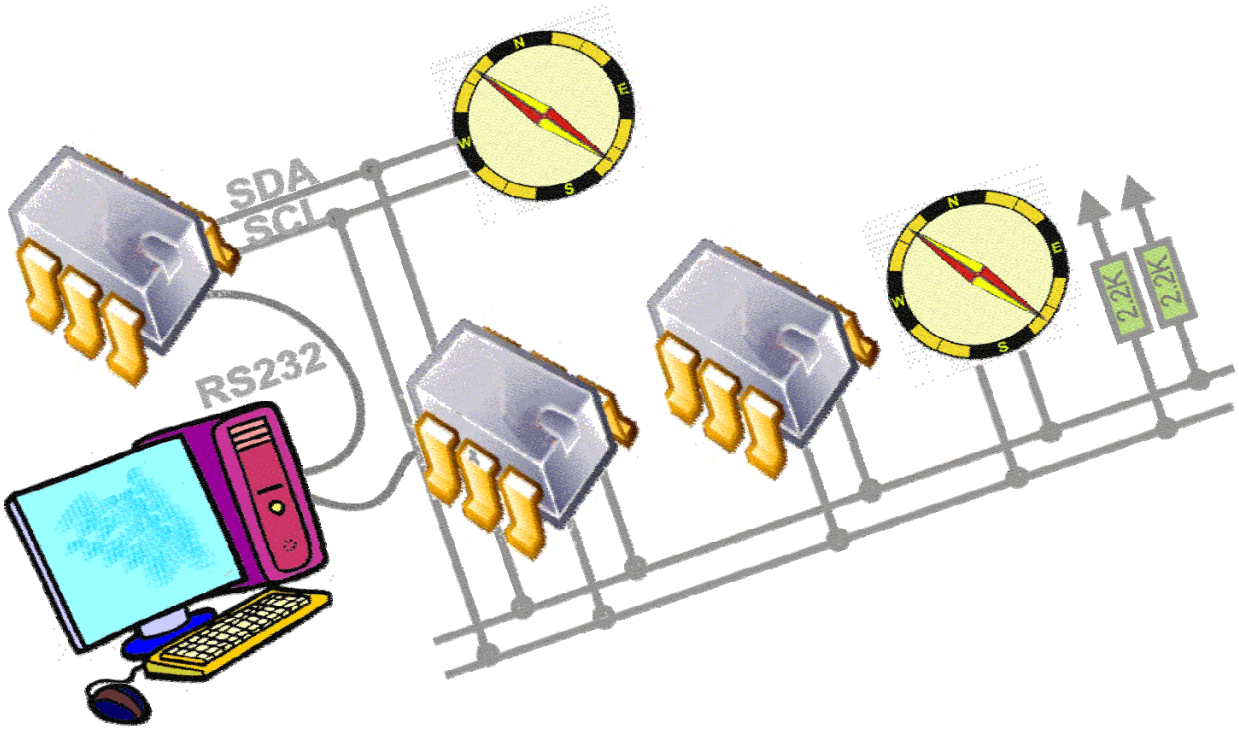
Introducció als fonaments del computador amb EduP12.

Joan Oliver



Editat pel Centre de Visió per Computador

Introducció als fonaments del computador amb EduP12.

Joan Oliver

Editat pel Centre de Visió per Computador

Introducció als fonaments del computador amb EduP12.

Joan Oliver

Departament de Microelectrònica i Sistemes Electrònics

Escola d'Enginyeria

Universitat Autònoma de Barcelona

08193 Bellaterra (Barcelona)

Edició 1a.

Editat per: Centre de Visió per Compurador.

Edifici O (Campus UAB)

08193 Bellaterra (Barcelona)

2012

ISBN: 978-84-940231-2-5

ÍNDEX

PRÒLEG

Introducció als fonaments dels computadors amb EduP12 és una introducció pràctica als fonaments dels computadors basada en el processador EduP12, un processador de 12 bits que presenta les següents característiques:

- És un processador ideal per comprendre el funcionament dels processador RISC basats en arquitectura von Neumann i Harvard. El processador disposa d'un bon repertori d'instruccions i admet un conjunt divers de modes d'adreçament que el fa adequat per a introduir el lector en els fonaments del computador.
- Es basa en una arquitectura de 12 bits, el que permet que tingui un conjunt d'instruccions simple però potent, amb codificació amena i permet adreçar memòria de programes i memòria de dades adequades per aplicacions que van més enllà de les docents.
- Introdueix prestacions avançades com ara interrupcions i la possibilitat d'introduir un conjunt important de perifèrics.
- Inclou eines d'aprenentatge basades en EduP12 com un assemblador i un simulador del nucli del processador.
- El processador està implementat en llenguatge VHDL per a ser programat en FPGA.
- La codificació de les instruccions i la metodologia de treball és similar a la que empren microcontroladors comercials actuals.

El llibre complementa l'estudi del processador amb una primera part que és una introducció als fonaments del computador, basada en una introducció als sistemes de numeració emprats en els sistemes digitals, una introducció als sistemes digitals i al funcionament del processador. Aquesta primera part és de lectura obligada per a tot lector que no tingui un coneixement bàsic del computador si després vol entendre el funcionament de qualsevol processador i, en concret, d'EduP12.

La segona part del llibre descriu en profunditat el processador EduP12. S'introdueix la CPU i la seva arquitectura, el codi màquina i modes d'adreçament i l'ús de perifèrics en el sistema. En capítols posteriors s'aprofundeix en l'arquitectura del processador amb la introducció d'interrupcions i la personalització del processador introduint aquells perifèrics que es requereixi en cada aplicació.

EduP12 és un processador de codi obert descrit en VHDL preparat per a ser usat sobre FPGA. El capítol 10 presenta la jerarquia de blocs establerta en la construcció del processador en VHDL i

incorpora informació sobre com modificar el *wrapper* per expandir el processador amb la introducció de nous perifèrics.

EduP12 està provat en la placa Spartan3 de Digilent. S'ha fet servir en aplicacions d'adquisició de senyal en sistemes de sensors. Molts dels exemples posats estan basats en aquestes aplicacions.

Capítol 1

INTRODUCCIÓ

El capítol introdueix el concepte d'informàtica des del punt de vista històric fent un breu repàs als antecedents històrics que, a partir dels avenços científics i els desenvolupaments tecnològics, han fet evolucionar les màquines calculadores convertint-les en els computadors que avui en dia coneixem i utilitzem.

Després el capítol introdueix de forma breu l'arquitectura genèrica del computador i el concepte d'algorisme, introduint els nivells conceptuals de descripció del computador.

El capítol acaba parlant sobre llenguatges de programació i fa una breu introducció a la metodologia de treball que s'ha d'emprar en el desenvolupament de programes.

1.1 Informàtica i computador.

Avui en dia quan es parla d'*informàtica* tothom entén que és la branca del coneixement que tracta sobre els computadors. És un terme comú emprat, entre altres llengües, en català, castellà i francès, que deriva de les paraules informació i automàtica, accions que eren dutes a terme pels primers aparells capaços d'actuar sobre unes dades mitjançant un automatisme amb l'objectiu ed calcular una nova dada.

En la informàtica hi convergeix la computació, l'arquitectura de computadors, la programació, metodologies de desenvolupament de software, la intel·ligència artificial i les xarxes de computadors com a disciplines més importants.

La paraula informàtica, comuna en llengües romàniques, no té el seu paral·lelisme en llengües anglosaxones. El terme informàtica sol ser traduït pel de *computing*, i els ensenyaments que aquí en diem informàtica es solen traduir pels termes *computer science* o *computer engineering* fent referència a la vessant més de programació (traduït per *software*) o de maquinari (vegi's *hardware*), respectivament.

En qualsevol cas avui en dia les antigues màquines calculadores s'han convertit en potents aparells de còmput i d'aquí els noms actuals de computadors o ordinadors, terme aquest darrer derivat més aviat del francès.

Segons el Diccionari de la Llengua Catalana *computador és una màquina electrònica digital que permet el processament automàtic, l'obtenció, l'emmagatzematge, la transformació i la comunicació de la informació mitjançant programes preestablerts*.

En termes d'operativitat podem dir que *el computador és una màquina capaç d'acceptar dades d'entrada, efectuar operacions lògiques i/o aritmètiques, i proporcionar la informació resultant a través d'un mitjà de sortida, sense intervenció d'un operador humà i amb el control d'un programa d'instruccions prèviament emmagatzemat en el propi computador*.

Queda clar així que el computador treballa amb dades, terme que al llarg de la història ha pogut fer referència a diferents formes de representar la informació capaç d'entendre la màquina calculadora o el computador. Avui en dia *dada es pot entendre com el conjunt de símbols emprats per expressar o representar un valor numèric, un objecte, un fet, una idea, en la forma adequada per a ser processada*. I aquesta definició és tan àmplia com l'enorme quantitat de tipus dades que un computador actual port acceptar com a entrada, anant des del concepte de bit fins a imatges capturades pels perifèrics.

El **bit** és la dada elemental de l'ordinador essent la unitat d'informació. Pot prendre dos valors possibles, 0 ò 1, configurant la base del codi binari.

Tot i la varietat de dades que existeixen i que un computador pot interpretar, s'ha de recordar que el computador en el seu interior només entén i treballa amb dades binàries. Qualsevol acció que es fa des del món extern cap al computador passa inevitablement per la traducció del valor extern a codi binari; i la traducció va de codi binari cap a dada externa si l'acció va del computador cap al món extern.

Donat que el bit és un terme massa petit quan es parla d'informació s'empren múltiples del mateix. Val la pena, per tant, recordar els seus valors:

- Bit. Unitat: 1.
- Byte. 8 bits o 2^3 bits.
- K (quilo) equivalent a 2^{10}. Per tant, 1 Kb són 1024 bits, i 1 KB són 1024bytes. Compte a no confondre el terme K com a 1000, que és la seva traducció en el sistema decimal![1]
- M (mega). Equival a $2^{20} = 1024^2 = 1.048.576$. Així 1MB són 1024 Bytes
- G (giga) val $2^{30} = 1024^3 = ...$
- T (tera) és $2^{40} = 1024^4$
- P (peta) és $2^{50} = 1024^5$
- E (exa) equival a $2^{60} = 1024^6$
- Z (zetta) és $2^{70} = 1024^7$
- Y (iota) és $2^{80} = 1024^8$

1.2 Antecedents històrics

La humanitat, des dels seus inicis, ha fet servir les seves habilitats per cercar la manera de trobar els enginys necessaris per imposar-se, no sols al medi, sinó també als seus semblants. L'evolució fins a l'era actual ha estat lenta. Però es pot dir que ha estat la conjunció del coneixement i de la tècnica que ha fet explotar el desenvolupament tecnològic aquests dos darrers segles. I l'evolució de l'era

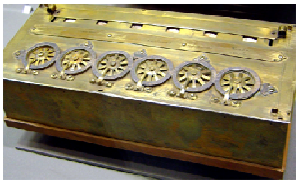
[1] Per a evitar aquesta confusió s'estan introduint les unitats binaries, en contraposició a les decimals. Així, mentre que 10^3 Megabytes equivalen realment a 1GB (1000·1000·1000 bytes), l'equivalent binari de 1024·1024·1024 bytes s'ha de dir que és un GiB (*Giga Binary Byte*).

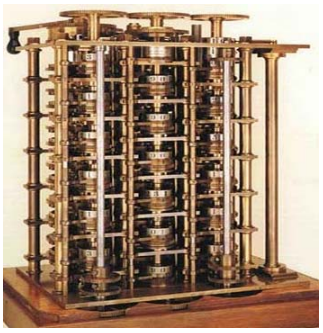
industrial a l'era del coneixement i de les xarxes socials actuals ha estat impulsada, sens dubte, per l'evolució de la informàtica.

1.2.1 Les màquines calculadores

Les primeres màquines inventades per l'home es van usar fonamentalment per ajudar-lo en tasques repetitives. En aquest sentit, *l'àbac* es pot considerar com el primer enginy pel càlcul manual d'operacions aritmètiques emprat ja en temps dels sumeris (2000 ac.).

La *clepsidra* és un altra automatisme que ens ha quedat constància de l'edat antiga. Conegut pels mesopotamis i millorat per les civilitzacions fins als grecs la clepsidra era un mecanisme basat en aigua que servei com a rellotge per a la mesura del temps.





a) **b)**

Figura 1.1. a) Pascalina. b) Màquina analítica

Tot i aquests invents, però, no va ser fins a l'era actual que es van començar a fer servir mecanismes mecànics que van treballar com a calculadores. En el s. XVII Blaise Pascal va crear la *Pascalina*, una màquina que funcionava amb engranatges, i que és coneguda com a primera calculadora mecànica. Anys més tard Gottfried Leibniz la va millorar.

Un salt important en la cerca d'automatismes el va produir a inicis del s. XVIII Charles Babbage. Babbage havia constatat que moltes operacions consistien de la repetició constant d'un conjunt predeterminat d'operacions. Aquest fet el va empènyer a buscar solucions cap a calculadores mecàniques que li resolguessin el càlcul d'aquestes operacions. Pel càlcul de logaritmes va crear la *màquina de diferències*. Continuant en aquest sentit va dissenyar la *màquina analítica*, una màquina mecànica basada en engranatges que havia de ser programada per l'usuari. La màquina analítica incloïa la majoria de les parts lògiques d'un ordinador actual, com ara unitat de memòria, procés de tasques repetitives, un programa de control i entrada/sortida. Un dels punts clau de la màquina analítica era que treballava amb el concepte de tarja perforada, invent emprat per Joseph Marie Jacquard per fer funcionar els seus talers mecànics. Problemes tècnics i econòmics van fer que mai s'arribés a construir.

Però les idees van quedar. El concepte de tarja perforada va ser emprat per Hollerith a finals del s. XVIII per a realitzar el cens nord-americà. Va construir la *Màquina Tabuladora* capaç de classificar 300 targetes perforades per hora. Hollerith va fundar el 1924 l'empresa *International Business Machines* (IBM).

A inicis del s. XX es produeixen dos fets importants. A nivell de lògica, Boole postula la seva àlgebra, avui en dia coneguda com a *àlgebra de Boole*, que constituirà la base de funcionament de tots els ordinadors. A nivell tecnològic s'inventa la vàlvula de buit, dispositiu electrònic que configurarà, més endavant, el nucli dels primers ordinadors electromecànics.

Emprant el corrent i el voltatge, i utilitzant dispositius com engranatges i relés, els inicis del s. XX veuen una escalada important en l'ús de calculadores mecàniques per a realitzar tasques diverses. La

culminació arriba a inicis de la Segona Guerra Mundial quan Aiken i altres presenten el que són considerats els primers ordinadors digitals *Mark I* i *Mark II* preparats per al càlcul i simulació de trajectòries balístiques, entre altres.

1.2.2 La 1ª generació, basada en els tubs de buit.

El 1943 J. P. Eckert i John Mauchly de la Universitat de Pennsylvania comencen a construir la que es considera la primera computadora electrònica, l' *Electronic Numerical Integrator and Computer* (ENIAC). Estava destinada al càlcul de trajectòries balístiques durant la Segona Guerra Mundial, però no va arribar a servir mai per acabar-se la seva construcció el 1946. Era una màquina enorme. Feia 25 metres de llarg, 2.7 metres d'alt i 60 cm. d'amplada, i contenia 18.000 vàlvules (figura 1.2). La seva capacitat de càlcul era de 1900 sumes/segon.

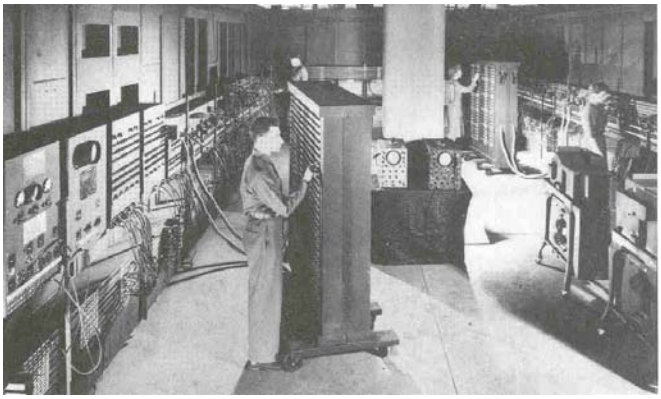


Figura 1.2 Fotografia de l'ENIAC. Es programava a base de clavilles.

El 1945 J. V. Neumann, J. P. Eckert i J. Mauchly passen a construir la computadora *EDVAC*. A diferència d'ENIAC, la computadora ja treballa amb codi binari. La seva principal aportació va ser un disseny ja basat en l'estructura de John von Neumann. Això és, el computador ha de tenir entrada/sortida de dades, memòria (per a emmagatzemar instruccions i dades que han de s'han de poder escriure i llegir) i recursos de càlcul per a operar amb les dades.

A finals de la dècada dels 40 es creen les memòries de ferrites dissenyades per a aplicacions de processament en temps real de senyals de radar. Pels computadors suposen un avanç important en l'emmagatzemament de la informació.

Ja en els anys 50 es construeixen diferents models de la computadora *UNIVAC*. Són les primeres computadores de propòsit general, inicialment emprades en càlculs de matrius, problemes logístics, etc. Tot i ser una computadora costosa, la UNIVAC I (1951) valia 1 milió de dòlars, se'n van fabricar 48 unitats ...tot un èxit !

Com a característica general d'aquestes màquines val a dir que eren màquines pesades, i grans, amb gran consum d'energia que requerien de sistemes de refrigeració especials. Tenien contínues aturades del sistema per fallades de components. L'entrada/sortida era per mitjà de targetes perforades i es programaven en assemblador. La informació es guarda en tambors magnètics.

1.2.3 La 2ª generació de computadores, basada en transistors.

El transistor, inventat el 1947, revoluciona el mercat de les computadores. El transistor és un dispositiu actiu (amplificador) més petit, més barat, i que dissipa menys calor que les vàlvules. Les millores que introdueix en els computadors es veuen en la taula 1.1.

	IBM701 (1952)	IBM7094II (1964)
Tecnologia:	Tubs de buit	Transistors
Memòria:	Amb tubs de buit	Amb ferrites
Temps de cicle:	30 μseg	1.4 μseg
Memòria:	2 K	32 K
Codis d'operació:	24	185
Aritmètica:	De punt fix	De punt flotant amb doble precisió

Taula 1.1. Comparativa entre dues comparatives IBM de la 1ª i la 2ª generació.

La segona generació de computadors es basa, així, en el transistor, molt més petit que la vàlvula, fet que fa disminuir la mida del computador i el consum d'energia. És molt més ràpid que la vàlvula i molt més fiable. Ja s'empren memòries de ferrita per guardar la informació i s'introdueixen les cintes i els discs per guardar la informació. La programació avança i es comencen a introduir els llenguatges d'alt nivell.

1.2.4 La 3ª generació de computadores, basada en transistors.

Els anys 60 es caracteritzen per l'evolució dels circuits integrats que permeten que múltiples transistors puguin ser posats sobre un mateix substrat emprant tècniques de creixement monolític, permetent la fabricació en massa de transistors, i abaratint-ne els preus. A més, el circuit integrat és un dispositiu actiu (amplificador) encara més petit, més barat, i que dissipa menys calor.

A nivell dels computadors el circuit integrat permet reduir-ne la seva mida al mateix temps que s'augmenten les seves prestacions. És molt més fiable. S'introdueix la compatibilitat per compartir software entre diferents equips.

Un computador considerat clàssic de la 3ª generació és el *PDP-11* de *Digital Equipment Corporation*, considerat el primer miniordenador comercial i que, només, costava 20.000$!.

1.2.5 Les generacions a partir dels anys 70.

Els anys 70 es caracteritzen, a nivell tecnològic, per la consolidació de les tecnologies de fabricació de circuits integrats. Apareixen els circuits LSI (*Large Scale Integration*) que contenen més de 1000 transistors i la memòria semiconductora. Al mateix temps s'han millorat les prestacions dels ordenadors i s'abarateixen els seus preus.

L'any 1971 apareix en el mercat el processador *Intel 4004*, el primer processador integrat en un mateix xip. Contenia 2300 transistors i era un processador basat en 4 bits.

A partir dels 70, el mercat dels computadors evoluciona en dos sentits: el dels supercomputadors, emprats en aplicacions amb grans bases de dades i el dels microcomputadors, més personals i per a petites empreses.

El microprocessador miniaturitza l'estructura del computador. Es minimitzen els circuits i augmenta la capacitat d'emmagatzemament de la informació. Es redueixen els temps de resposta. A mitjans dels anys 70, a més, s'introdueix la memòria semiconductora, introduint un salt molt important en la capacitat de guardar informació i la velocitat de lectura/escriptura de dades.

A inicis dels anys 80 IBM treu al mercat els ordinadors personals. El *Commodore VIC-20* representa el primer computador personal que surt l'any 1980 a un preu inferior a 300$. El 1984 IBM treu al mercat el *Personal Computer,* conegut com a *PC*, un ordinador personal potent per a aplicacions de petites indústries i de baix preu que es comença a distribuir per les llars. Els ordinadors personals i portàtils actuals són l'evolució lògica del PC.

A partir dels anys 90 es generalitzen les aplicacions que fan ús del computador. El computador passa a formar part dels aparells d'ús quotidià. La seva aplicabilitat s'estén a tots els camps de l'activitat humana: indústria, medicina, educació, llar, comerç, administració, agricultura, etc.

Ja en el s. XXI, i amb noves tecnologies d'integració basades en FPGAs, el microprocessador, nucli del computador, passa a ser un component software en els sistemes embeguts que, amb la inclusió de perifèrics en la mateixa, particularitza l'ús de la computació en qualsevol àrea de coneixement.

1.2.6 Evolució del programari o *software*.

La base d'entrada del computador són les dades. La principal missió del computador és el tractament de les dades per transformar-les en informació. El software ha aparegut amb la generalització del computador. Indica la capacitat d'un hardware per a ser programat per fer diverses tasques canviant, simplement, el conjunt d'instruccions que s'executen.

El software és un concepte nou. Va ser en el s. XVII quan Ada Augusta, que mantenia correspondència amb Charles Babbage, va descriure de forma metòdica un conjunt de passos per calcular la seqüència dels nombres de Bernouilli amb la maquina analítica, el que es considera com el primer programa escrit. El llenguatge conegut avui en dia com *ADA* té el seu nom en honor seu. De fet, Babbage agafa la idea de les targetes perforades de Jacquard per descriure els programes que havien de córrer en la seva màquina analítica.

En el 1943 la computadora ENIAC mostra la seva capacitat de realitzar diferents càlculs per via de la programació. Els programats eren entrats a través de connexions amb clavilles, a mode de la típica centraleta telefònica, i emprava programació digital binària.

Aquesta manera de fer rudimentària comportava molts errors. Es deu a un conjunt de dones matemàtiques que treballaven amb els enginyers amb J. P. Eckert i John Mauchly el pas de programa descrit en codi màquina a assemblador. La idea va ser simple: donar un nom lògic a cada codi màquina (*mnemotècnic*) i crear un traductor que passés a codi màquina cada instrucció.

Tot i això, en aquesta època cada ordinador tenia el seu codi màquina. Cada nova versió d'ordinador necessita del seu llenguatge. Amb UNIVAC II es ven el fet que el software que anava amb la maquina ja és compatible amb el maquinari anterior.

Amb la segona generació de computadors comencen a aparèixer els llenguatges d'alt nivell que permeten realitzar més fàcilment programes complexes. És l'era dels llenguatges FORTRAN, COBOL, BASIC, ... Comencen a aparèixer les aplicacions comercials, especialment per a l'elaboració de nòmines, facturació i comptabilitat.

Però no serà fins a la tercera generació que es generalitzarà l'ús dels llenguatge d'alt nivell i es compatibilitzarà el software per a ser emprat en diferents màquines. S'introdueix el teleprocés, terminals remots per a tractar i realitzar operacions amb bancs de dades. Apareix la multiprogramació i el temps compartit que permet a diferents usuaris compartir el computador ja que és capaç discernir entre diferents processos. L'ús de l'ordinador s'estén a aplicacions industrials, educació, llar, administració, joc, .etc.

A partir de la quarta generació, amb la introducció del microprocessador, es projecte l'ús del microprocessador més enllà del computador i s'introdueix en tot tipus d'aparells. Ja en els anys 80, en plena era PC, apareixen els programes específics d'usuari, molts d'ells coneguts avui en dia com a ofimàtica: processadors de text, CAD's, etc. Apareix la intel·ligència artificial amb què es vol equipar a les computadores amb la capacitat de raonament dels humans. Apareixen els sistemes experts com a sistemes amb múltiple informació i estratègies que ajuden a l'usuari en els problemes a resoldre.

1.3 Nivells conceptuals de descripció del computador

El computador és una màquina complexa que des del punt de vista estructural i de l'ús que se'n fa pot ser vista des de diferents nivells conceptuals (figura 1.3).

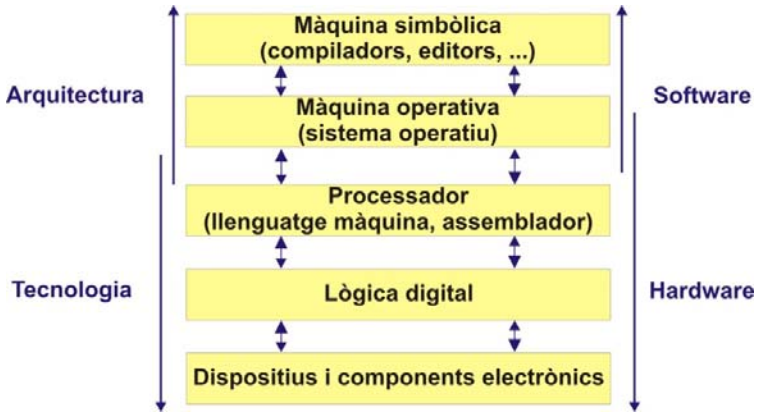


Figura 1.3. Nivells conceptuals del computador

Des del punt de vista arquitectònic el computador es pot dividir en el nivell tecnològic i el nivell arquitectural.

El nivell tecnològic està format per aquell conjunt de dispositius i components que configuren el nucli electrònica de la màquina. És la part de més baix nivell. Les branques de la informàtica que estudien aquesta part són la dels sistemes digitals i, a més baix nivell, ens posaríem ja en l'electrònica digital. Aquesta part física té la seva frontera entre o sobre el nivell especificat per l'assemblador.

A partir de la frontera entre assemblador i sistema operatiu s'entra ja en la branca de l'arquitectura de computadors, que tracta sobre l'estructura fonamental que defineix el computador i que ve focalitzada, fonamentalment, per la forma de treballar la CPU. El conjunt d'instruccions (*ISA* o *Instruction Set Architecture*) i les formes d'adreçament a memòria emprades per la CPU defineixen l'estructura principal del computador.

El computador també es divideix tradicionalment en *maquinari* (*hardware*) i *programari* (*software*). Queda molt clar per tothom que el *hardware* el forma el conjunt de components que configuren la màquina que tots veiem i que és on corren totes les aplicacions i programes que empren. Per altra part, el software el composen el conjunt de procediments de càlcul, documents i dades que s'empren en les operacions que es realitzen en un computador. Forma la part lògica del computador.

La frontera entre el que s'anomena hardware i software queda difusa entre l'assemblador i el sistema operatiu. L'assemblador està lligat totalment al codi màquina de la CPU. Per altra part, el sistema operatiu conté ja un conjunt de programes que, encara que siguin de nivell baix, controlen el funcionament de la màquina i deslliuren el programador del coneixement precís de l'arquitectura del computador.

1.3.1 El software.

A nivell de software es sol parlar de trens grans grups: el software de sistema, el de programació i el d'aplicació.

El software de sistema s'encarrega del control del computador i dels seus perifèrics aïllant el programador d'aquestes tasques rutinàries. D'aquesta manera l'usuari no cal que tingui coneixement de l'arquitectura ni composició del computador: memòria, discs, ports, comunicacions i perifèrics en

general. El software de sistema inclou fonamentalment a sistemes operatius, controladors de dispositius, servidors i utilitats similars.

El software de programació el formen les eines que permeten al programador desenvolupar els programes informàtics que empren els diferents recursos del computador i que corren sobre el computador. A tal efecte el programador disposa de diferents recursos o llenguatges de programació. S'hi inclouen en aquest grup els editors de text, compiladors, intèrprets, enllaçadors, depuradors i entorns integrats de desenvolupament, anomenats genèricament com a IDEs.

Finalment queda el grup del software d'aplicació, format per tots aquells programes i aplicacions de control que permeten a l'usuari dur a terme tasques específiques en la seva àrea de treball. Entren en aquest grup les aplicacions ofimàtiques, softwares diversos (del món empresarial, educatius, de càlcul numèric, etc.), bases de dades, entorns CAD, videojocs, etc.

1.4 Llenguatges de programació i algorismes

1.4.1 Llenguatge de programació.

El computador és una màquina destinada a executar les accions que li són programades. Només desenvoluparà les tasques que se li hagin especificat en termes d'operacions bàsiques que la màquina és capaç d'entendre i executar.

Tot programa que ha de ser executat en el computador s'ha de descriure en una terminologia adequada i amb una seqüència de passos elementals i seqüencials que duguin al computador a executar aquelles accions precises que condueixin a un resultat esperat. És el que s'anomena *algorisme*, i que es pot definir com la seqüència ordenada de passos elementals (primitives), exempta d'ambigüitat, que condueix a la resolució d'una tasca determinada amb uns recursos predeterminats.

S'entén així per *programa* la forma d'expressió de l'algorisme. I el *llenguatge de programació* no és res més que el format d'expressió del programa. El llenguatge es caracteritza per tenir una sèrie de regles semàntiques i sintàctiques estrictes que s'han de seguir quan s'escriu un programa.

Si es posen els llenguatges de programació en relació amb els nivells conceptuals de descripció que s'han vist de l'ordinador s'observa que els programes que es creen poden treballar amb el computador a molt baix nivell (propers al codi màquina) fins a molt alt nivell, permetent a l'usuari la realització de programes sense conèixer res de la màquina. Una classificació dels llenguatges de programació atenent al nivell amb el que interactuen amb el computador és la següent:

- Llenguatge màquina. És el llenguatge de més baix nivell, escrit en llenguatge directament intel·ligible per la màquina Les instruccions, anomenades codi màquina, són cadenes binàries (normalment 0's i 1's). Depèn totalment del processador. Per tant, cada processador té el seu propi llenguatge màquina.

  Són llenguatges eficients, però poc fiables, difícils de descodificar i poc transportables.

- Assemblador. És un llenguatges de baix nivell, similar al codi màquina. La diferència entre els dos està en què l'assemblador empra mnemotècnics per codificar les instruccions codi màquina. Un traductor realitza el pas d'assemblador a codi màquina.

  Com el codi màquina també és un llenguatge dependent del processador.

- Llenguatges d'alt nivell, com C, Pascal, Fortran, Lisp... Són llenguatges més familiars a la parla habitual, transportables i independents de la màquina. Els programes escrits amb aquests llenguatges es poden executar en gairebé qualsevol màquina sense cap o amb molt poques modificacions en el programa.

Els programes són més fàcils de depurar, costen menys de ser apresos i es redueix el seu cost de creació

En contrapartida, els recursos de la màquina no s'aprofiten tant. Un programa escrit en un llenguatge d'alt nivell empra més recursos de memòria i necessita més temps d'execució que el seu equivalent escrit en assemblador.

Al codi d'un programa escrit en llenguatge d'alt nivell se l'anomena *codi font*. La descripció del codi font es du a terme treballant amb editors de text genèrics o específics al llenguatge de programació. Tot codi font s'ha de traduir o *compilar* per a ser executat en la màquina.

Hi ha alguns llenguatges d'alt nivell que no empren el procés de compilació en l'execució dels seus programes. Són llenguatges que tradueixen directament les seves instruccions executant el programa línia a línia. El tradicional Basic, el QuickBasic, l'actual Python en són alguns exemples.

A diferència d'aquests, els llenguatges d'alt nivell que empren la compilació primer tradueixen el programa a un codi intermedi, anomenat *codi objecte*, per després fer la traducció a codi màquina. La traducció es fa per arxius complerts. Com a exemples de llenguatges compilats hi ha el Fortran, el Pascal, el C, etc.

Els traductors i compiladors sovint fan ús d'un conjunt de programes predissenyats que interactuen amb l'entorn i faciliten la tasca de creació de programes. Aquests programes es troben precompilats en llibreries i agilitzen enormement la tasca del programador.

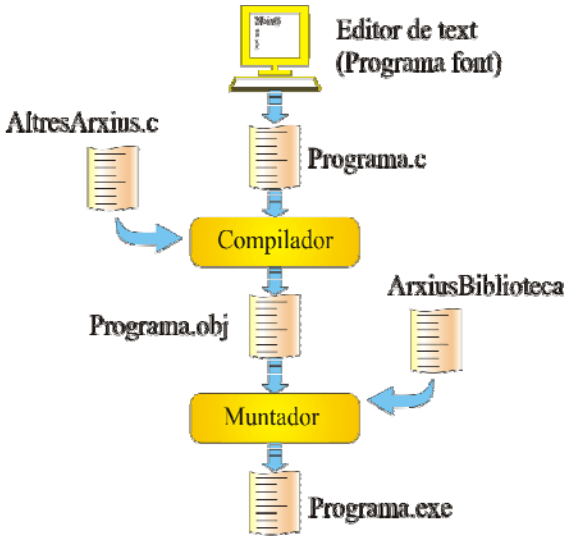


Figura 1.4. Procés de compilació i traducció d'un llenguatge de programació.

1.4.2 Processament de la informació en el computador.

El computador ha evolucionat enormement en les darreres dècades passant de ser simples calculadores mecàniques a complexes màquines actuals amb capacitats de còmput enormes. Avui en dia els computadors són veritables cervells de processament de la informació. El computador treballa amb dades i en retorna un resultat (figura 1.5).

Figura 1.5. Computador i processament

En el processament de la informació hi intervenen, doncs, tres components:

- Les dades, o representacions de les entitats amb què es treballa en un format adequat per a ser processada
- El procés, que ve donat com a un conjunt d'operacions que manipulen les dades
- I la informació, que es el conjunt de dades ordenades corresponents a la sortida del procés

En tot aquest procés s'ha de veure al computador com una màquina capaç d'entendre els algoritmes per tal d'obtenir la informació processada a partir de les dades subministrades. S'ha de ser doncs molt conscient en què el computador és ideal per:

- A la realització de tasques que es poden repetir en el temps.
- Com a medi d'emmagatzemar i tractar informació
- A la realització de tasques que requereixen d'altes velocitats de càlcul

I s'ha de tenir sempre present que el executarà exactament aquelles accions que se li hagin programat, fet que porta a la necessitat de saber escriure de forma precisa i sense ambigüitat el programa. En aquest sentit, és fonamental entendre el paper que juga l'algorisme en el desenvolupament dels programes.

1.4.3 Creació d'algorismes

S'entén com a *algorisme* el mètode de resolució d'un problema que segueix una seqüència d'operacions precises per tal d'aconseguir el resultat esperat. Algorisme és una paraula d'etimologia àrab que ens ha arribat de la persona *Muhàmmad ibn Mussa al Khwarazmí*, matemàtic, geògraf i astrònom del s. IX que va descriure i posar en ordre per primer cop en una obra la representació posicional dels nombres en el sistema decimal. Conjuntament amb Euclides (s. IX ac.), que va escriure un mètode per trobar el màxim comú divisor de dos nombres enters, se'ls considera els iniciadors dels algorismes.

A nivell de programació la resolució de programes passa a ser una tasca metòdica que es desglossa en una sèrie de passos que convé seguir (figura 1.6):

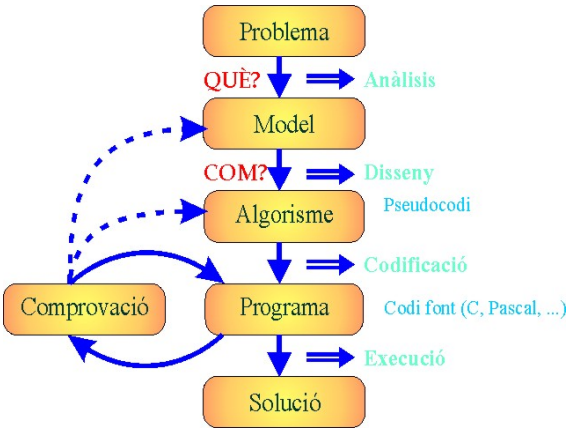


Figura 1.6. Etapes en el desenvolupament d'un programa.

- El primer pas en la realització d'un programa és entendre bé el problema. En aquesta etapa cal fer una anàlisi acurada de l'enunciat preguntant què s'ha de fer. S'han d'entendre molt bé quines són les entrades i sortides que es demanen en el programa, i s'ha d'establir el procés a realitzar.

- De l'anàlisi n'ha de sortir el model que ha de permetre concebre i dissenyar el problema que es planteja. Del model en surt el cóm s'ha d'aconseguir resoldre el problema.
- La resposta al cóm la dóna l'algorisme. És el desglossament del problema en un conjunt de passos elementals que dóna solució al problema. L'algorisme ha de ser precís, finit i determinista. S'entén per precís que ha de donar una solució no ambigua. La finitud implica que l'algorisme ha de tenir una entrada i una sortida, no es pot quedar per sempre executant processos sense arribar a un final. Finalment per determinista s'entén el fet que tota execució de l'algorisme amb unes determinades dades ha de donar, sempre, el mateix resultat.
- El programa estableix el conjunt d'instruccions que executen l'algorisme emprant unes variables d'entorn predeterminades o, més concretament, emprant un llenguatge específic. L'algorisme es genèric per a tots. El programa particularitza l'algorisme.
- Una etapa molt important en el procés de creació de programes és la de verificació. No es pot donar un problema per resolt si no s'han provat totes les seves possibilitats. En programes grans la creació de patrons de test passa a ser una etapa difícil en tot el procés de creació del programa.

En definitiva, tot el procés de creació de programes ha de ser coherent. És a dir, el programa ha de resoldre de forma precisa el problema plantejat. I per fer-ho cal indicar a la computadora els passos adequats que duen a la seva resolució.

Durant la creació d'un programa, i a nivell de programació, es sol passar per tres etapes:

- Creació del *diagrama de flux*. El diagrama de flux és una representació visual de l'algorisme basada en figures geomètriques predeterminades que indiquen accions concretes a fer pel programa. Representa el pas inicial del procés que, visualment, permet comprovar de forma immediata si l'algorisme que es planteja resol el problema enunciat.
- A un nivell similar o lleugerament inferior s'hi posa l'algorisme en format pseudocodi. El *pseudocodi* és explicitar en llenguatge escrit l'algorisme representat en el diagrama de flux. El pseudocodi no és en sí mateix un llenguatge de programació, sinó l'expressió de l'algorisme en llenguatge col·loquial. L'experiència en la creació de programes fa que diagrama de flux i pseudocodi presentin prestacions similars a ulls de programadors experts.
- El programa. Finalment hi ha la descripció de l'algorisme en la sintaxi i semàntica del llenguatge de programació. És quan el diagrama de flux i pseudocodi queden personalitzats ens unes instruccions concretes establertes pel llenguatge de programació, creant el programa.

La creació de programes és una tasca gens senzilla en la majoria dels casos per les múltiples implicacions i processos que s'han de tenir presents. Molts passos, que semblen insignificants, són essencials per a una execució correcta del programa. Per exemple, ens hem plantejat mai què implica engegar i arrencar el cotxe? Soms capaços de posar en un algorisme tots els passos que cal executar per a què una persona que no hagi vist mai un cotxe sigui capaç d'aconseguir-ho?

Evidentment no és un exercici per a ser programat en un ordinador, però sí que és un exercici interessant de discussió en classe.

El mateix esquema que se segueix en la reflexió de l'algorisme, serveix en el plantejament d'exercicis més enfocats al processament de dades en un ordinador.

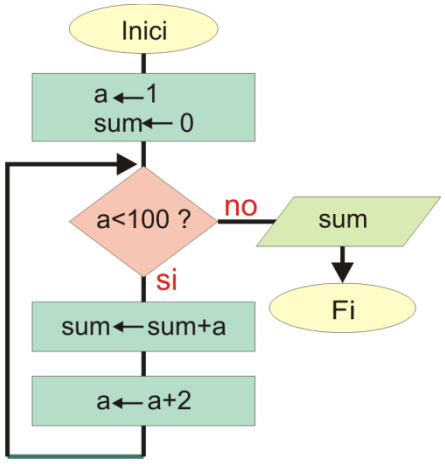
Exemple.

L'exercici següent planteja trobar el programa que realitza la suma dels nombres senars entre 1 i 100.

Els passos a seguir són els següents:

1. Definir clarament quines són les entrades i les sortides.
   En aquest cas vénen donades per:
   Entrades: els nombres positius que es troben entre 1 i 100
   Sortida: La suma d'aquests nombres
2. Es planteja l'algorisme. Es pot representar en diagrama de flux o en pseudocodi. L'algorisme es planteja tenint present:
   a. L'estructura o estructures que defineixen la seqüència de passos. En aquest cas es realitza un bucle que ha d'agafar els valors senars que hi ha entre 1 i 100.
   b. Per a realitzar tota la seqüència calen variables per guardar valors intermedis. En aquest cal una variable *a* per dur el compte del número amb el que estem treballant, i la variable *sum* que portarà la suma fins al final.

   La figura 1.7 mostra el diagrama de flux i un algorisme escrit en pseudocodi que implementa l'enunciat plantejat.




```
Vars: a, sum;
Inici:
        A ← 1, sum ← 0;
        Si a<100 aleshores
                sum ← sum + a
                a ← a + 2
        Fi Si
        Imprimir (sum)
Fi
```

Figura 1.7. Diagrama de flux i pseudocodi

3. Un Cop queda clar l'algorisme que s'ha d'implementar es pot personalitzar en el llenguatge d'alt nivell amb què es treballa.
   La figura 1.8 mostra l'algorisme escrit en C i en Python.

```
#include "stdlib"
void main() {
        a = 1;
        som = 0;
        while (a<100) {
                som = som+a
                a = a+2
        }
        printf ("%s", som);
}
```

```
a=1
som=0
while a<100:
        som = som+a
        a = a+2
print som
```

Figura 1.8. Programa escrit en C i en Python

1.5 Desenvolupament de software amb el computador

El desenvolupament de software és una tasca complexa a la que les empreses hi dediquen molts de recursos. No es tracta només del simple fet de desenvolupar un programa, si no que a més s'ha de garantir el seu perfecte funcionament i la seva continuïtat en el temps. El desenvolupament de software és una tasca dura en la que es contempla que hi ha tres etapes fonamentals:

- Desenvolupament i disseny. És l'etapa de creació del programari. Passa per les etapes següents.
    - En l'etapa d'anàlisis es responen les preguntes de què ha de fer el programa, quins resultats ha de donar i quines són les dades (entrada/sortida) amb què s'ha de treballar. També cal valorar el llenguatge que s'empra en l'escriptura.
    - L'etapa de disseny és l'encarregada de dissenyar i plantejar els diferents passos que s'han de preveure en la resolució del problema plantejat. La resolució del problema implica passar per un procés d'etapes successives de refinament.
    - Etapa de codificació. És l'etapa d'implementació. L'escriptura del programa es fa en llenguatge d'alt nivell, on convé seguir les normes de la programació estructurada i modular. La programació orientada a objecte facilita en gran mesura la creació de programes de manera ràpida i eficient.
    - Etapa de verificació. Es realitza a partir d'un conjunt de dades de prova que verifiquen el comportament del programa.

Figura 1.9. Etapes en la vida del software

- Documentació. És la generació de documents descriptius del programa. En la vida d'un software convé mantenir una documentació coherent i fidedigne de tot el que afecta al software. Per tant, convé tenir una descripció exacta del què fa, tenir una descripció del desenvolupament seguit i documentar el software. S'han de documentar també els canvis que sofreixen els algorismes, llistar els programes, guardar els resultats de les proves i realitzar uns bons manuals d'usuari.
- Manteniment. Es fa durant la vida normal d'ús del programa. Sol requerir un esforç molt gran ja que quan convé depurar falles ò *bugs* no detectats durant l'etapa de disseny, implica tornar-se a posar dintre l'algorisme amb què es va dissenyar. És important aquí haver documentat correctament el software realitzat

L'enginyeria del software és la branca de la informàtica que s'encarrega del desenvolupament del software. Tots aquests aspectes es troben detallats i explicats en un conjunt de normatives que estandarditzen el procés de creació de programes.

1.6 Resum del capítol

El capítol dóna una visió genèrica introductòria dels diferents vessants que es troben en la informàtica:

- Es presenta a nivell històric l'evolució de la informàtica.
- Introdueix el computador com a màquina simbòlica a través de la seva composició, des de component electrònic fins a màquina de procés.
- Es dóna una classificació del software des del punt de vista funcional.
- S'introdueix el concepte d'algorisme i diagrama de flux com a eina genèrica per a la creació d'algorismes.
- I finalment es fa un èmfasis en la rasca de l'enginyeria del software en les etapes de creació i manteniment del software.

Capítol 2

LA INFORMACIÓ EN EL COMPUTADOR

En el capítol 1 s'ha introduït que la tasca fonamental de l'ordenador és el processament de la informació. L'ordinador ha de rebre dades d'entrada per guardar-les, processar-les i/o enviar-les cap a perifèrics de sortida.

Si ens centrem en el tipus de dades, el problema principal que sorgeix és el de proporcionar les dades a l'ordinador en el format adequat per a ser processades. Text, nombres, imatge, so i vídeo són dades amb les que actualment treballa l'ordinador. Cada tipus de dada té el seu format i la seva codificació. Aquest capítol tracta dels diferents formats de representació de dades en l'ordinador.

2.1 La informació en l'ordinador

Les dades que es tracten en el món real són dades analògiques, no digitals. Quan l'ordinador es comunica amb l'exterior la primera tasca que ha de fer és transformar la informació que li arriba en el format adequat. La unitat d'informació en l'ordinador és el bit. Però això no significa que tota la informació es guardi en el mateix format. Quina és la codificació dels nombres, el text, so, imatge en bits?

- El text normalment s'entra en l'ordinador en el format reconegut pels humans. Les lletres són entrades com a tals. Internament cada lletra tindrà la seva codificació.
- Els nombres s'entren de la mateixa manera que els altres caràcters. Si es guarden com a caràcters no necessiten de transformació, però quan el significat posicional de les xifres s'ha de tenir en compta, fa que el nombre s'hagi de tractar com a tal i no com a caràcter.
- El so és una dada més complexa. El so és una ona sonora que es propaga. En l'ordinador el so és capturat per un micròfon que en fa una transducció a senyal analògic. Posteriorment aquest senyal és tractat i digitalitzat per a ser tractat per l'ordinador com les altres dades, en format binari.
- Les imatges representen dades compostes per molts valors que requereixen d'un ús intensiu del computador. Les imatges poden ser estàtiques o dinàmiques (vídeo). Es solen emprar tècniques de compressió de la imatge, especialment en aquest darrer cas.

- Finalment cal tenir presents les instruccions. Tant a alt nivell com a baix nivell, es solen entrar en format text. Amb un procés de codificació adequat a 0's i 1's es transformen, després en codi màquina.

2.2 Representació de text

El text es representa per mitjà de caràcters. El computador empra conjunt de caràcters més ampli que l'usat normalment en l'escriptura quotidiana. Es poden classificar en:

- Alfabètics. Són els caràcters emprats normalment en escriptura: A, B, ..., a, b, ...
- Numèrics. Corresponen als números emprats en base decimal: 0, 1, ... 9
- Especials. Són els caràcters de puntuació: ! ” . , / SP, ...
- Geomètrics i gràfics. Corresponen a altres caràcters especials, alguns corresponents a lletres amb signes de puntuació i altres a vegades emprats per fer gràfics simples: |, ¶, ◘, ¬, ñ, ¢, Ã, ┌, ■, ß, ...
- Caràcters de control. Són caràcters introduïts d'ençà amb la informàtica i que impliquen realitzar accions especials en el computador o en perifèrics. Exemples en són LF (salt de línia), CR (retorn de carro), timbre (BEL), etc. Alguns d'aquests caràcters realitzen accions especials i no són imprimibles.

La representació de caràcters en un computador implica realitzar una codificació binària de cada caràcter. És una codificació bijectiva del tipus β = {a, b, ...A, B, ...0, 1, ...”, $, ...} → $\{0,1\}^n$. Això és, a cada caràcter se li assigna un vector binari. Malauradament la codificació de caràcters no és única, existint diferents jocs de caràcters. Alguns d'aquests jocs de caràcters són:

- El *Standard Binary Coded Decimal* o BCD. És un codi de 6 bits introduït els anys 60 per millorar l'entrada binària de les targetes perforades en els ordinadors. Permet representar fins a 64 caràcters. Inclou els números, les lletres majúscules i caràcters de control. En l'apartat 2.5.14 es veu de forma més detallada.
- L' *American Standard Code for Information Interchange* o ASCII és un joc de caràcters de 7 bits introduït el 1967 que permet representar els números, les lletres majúscules i minúscules, caràcters de puntuació i caràcters de control, alguns no imprimibles. Tot i que inicialment el vuitè bit estava destinat a un bit de paritat, la limitació de caràcters representables feu que l'ASCII s'estengués a 8 bits emprant el darrer bit per codificar altres caràcters no inclosos en la llengua anglesa, com ara accents o caràcters com la ç. Aquestes representacions agafen els valors superiors al 127 de la taula. Tot i que varien d'un estàndard a l'altre el més utilitzat és el ISO Llatí-1 o ISO-8859-1, corresponent a Amèrica i Europa Occidental.
- El *Extended Standard Binary Coded Decimal Interchange Code* o EBCDIC, també de 8 bits de representació.
- Tots aquests codis són de 8 bits, pel que tenen limitada la representació de caràcters. UNICODE (ISO/IEC 10646) és un nou joc de caràcters proposat per empreses i associacions que han de processar texts de molt diverses procedències que pretén incloure tots els signes gràfics emprats en totes les llengües. Per això empra 16 bits de representació, vol ser universal, amb unicitat de caràcters, i uniforme, no codifica caràcters de control, inclou caràcters combinats (com la *ñ*) i intenta evitar duplicitats de caràcters molt semblants. Empra un esquema de zones

La taula 2.1 mostra la codificació ISO-Llatí15 definit per l'Organització Internacional de l'Estandardització. És la versió 15 de l'ISO 8859 que conté el conjunt de símbols emprats en els

llenguatges dels països de l'Europa Occidental. Estan remarcats els símbols introduïts darrerament amb l'entrada de l'euro. Les files ressaltades en colors rosa i verd corresponen als caràcters de control.

ISO-8859-15																
	x0	**x1**	**x2**	**x3**	**x4**	**x5**	**x6**	**x7**	**x8**	**x9**	**xA**	**xB**	**xC**	**xD**	**xE**	**xF**
0x	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1x	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2x	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8x	PAD	HOP	BPH	NBH	IND	NEL	SSA	ESA	HTS	HTJ	VTS	PLD	PLU	RI	SS2	SS3
9x	DCS	PU1	PU2	STS	CCH	MW	SPA	EPA	SOS	SGCI	SCI	CSI	ST	OSC	PM	APC
Ax	NBSP	¡	¢	£	€	¥	Š	§	š	©	ª	«	¬	SHY	®	¯
Bx	°	±	²	³	Ž	µ	¶	·	ž	¹	º	»	Œ	œ	Ÿ	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Taula 2.1. Taula ISO-Llatí15

2.3 Representació de so

Les aplicacions multimèdia processen text, imatge i so. El so és, per tant, un altre tipus de dada que el computador ha de tractar. És una ona que es transmet en un medi i que el computador captura a través del micròfon. El micròfon dóna un senyal analògic continu en el temps com el que es mostra en la figura 2.1, corresponent a la captura de la paraula *hola*.

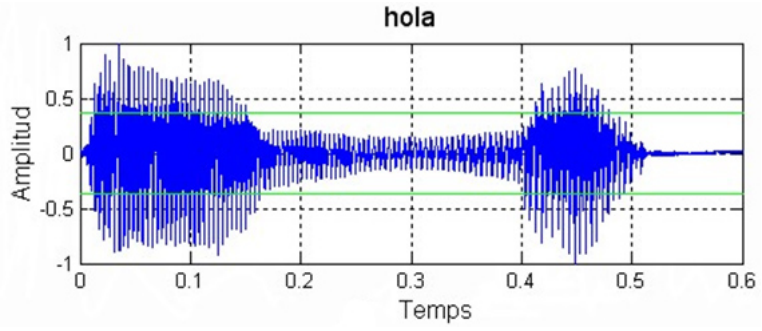


Figura 2.1. Gravació de la paraula *hola*.

Per a poder emmagatzemat,aquest senyal s'han de realitzar dos passos:

- Primer es mostreja el so a una freqüència fs determinada, de forma que s'agafa una mostra del senyal cada Ts=1/fs. Per exemple, si els 0.5 segons que dura el so en la figura 2.1 es mostregen a 22.225 Hz, vol dir que s'agafen 11.112 punts d'aquesta ona a intervals iguals.
- Cadascun dels punts, segons la figura, agafa un valor entre 1 i -1. Aquests són valors encara analògics. Per a passar cada punt a valor binari es pot emprar un conversor analògic a digital. Considerant que s'empra una codificació binària d'un byte per punt, el so de la figura necessita una capacitat de memòria d' 11.112 bytes.

Aquest mètode de guardar la informació d'àudio es basa en la digitalització de senyals analògiques, mètode introduït el 1937 per Alec Reeves i s'anomena *Pulse Code Modulation* (PCM). La qualitat del senyal que es guarda depèn de dues variables:

- La *freqüència de mostreig* que en determina la *qualitat*. Per tenir una qualitat mínima la freqüència de mostreig ha de ser almenys dos cops superior a la màxima freqüència del so adquirit.
- La *profunditat de mostra en bits* que és nombre de bits a què es transforma cada mostra, i en dóna la *precisió.* El nombre de bits emprat per la codificació determina la precisió del so adquirit.

La qualitat i precisió amb què es guarda un so depèn en gran mesura de l'aplicació. El so que es transmet per un fil telefònic no té per què requerir la mateixa qualitat que un so d'enregistrament de música digital. Evidentment, la quantitat de memòria que es requerirà per guardar una mateixa quantitat de temps el so també serà diferent.

La taula 2.2 mostra les característiques de diferents senyals de so digitalitzats.

	Bits/mostra	**Fs (Ks)**	**Ts (μseg)**
Telèfon (PCM)	8	8	125
Telèfon de qualitat	8	11,025	90,7
Ràdio	8	22,05	45,4
CD (per canal)	16	44,1	22,7
DAT (Àudio professional)	24	96	10.4

Taula 2.2. Gravació de la paraula *hola*.

La quantitat de memòria que es necessita per guardar un so es pot calcular a partir de les característiques proporcionades en la taula. Per exemple, la quantitat de memòria que es necessita per guardar una cançó gravada en so estereofònic de qualitat que dura 2 minuts i mig és:

$$150\,\text{seg}\cdot\ 44100\frac{mostres}{seg}\cdot 16\frac{bits}{mostra}\cdot 2canals = 211.68MBits = 26.46MB$$

De la figura es desprèn que la quantitat de memòria que es necessita per guardar so de qualitat és gran. Per altra part, la pròpia estructura del so mostra que una forma de gravació directa sense compressió, com el càlcul que s'ha fet, guarda molta informació redundant.

Per intentar reduir el volum d'informació en la gravació de sons es solen aplicar un conjunt de transformacions per *codificar* el so llur missió és reduir la informació a guardar. Per a reproduir el so es realitza la transformació inversa amb algorismes anomenats *descodificadors*, recuperant el so inicial. Els algorismes que realitzen ambdues transformacions s'anomenen CODECs.

Formats d'àudio i CODECs

El so es pot guardar en diferents formats, entre els que destaquen els següents:

- PCM (*Pulse Code Modification*). És el format bàsic d'adquisició de senyal, basat en un tren de polsos per a cada mostra, que es un transmet en sèrie cap al perifèric de guarda. La mida del fitxer on es guarda el so depèn de la freqüència de mostreig i de la profunditat de mostra.

  La figura 2.2 mostra el concepte bàsic de codificació PCM.

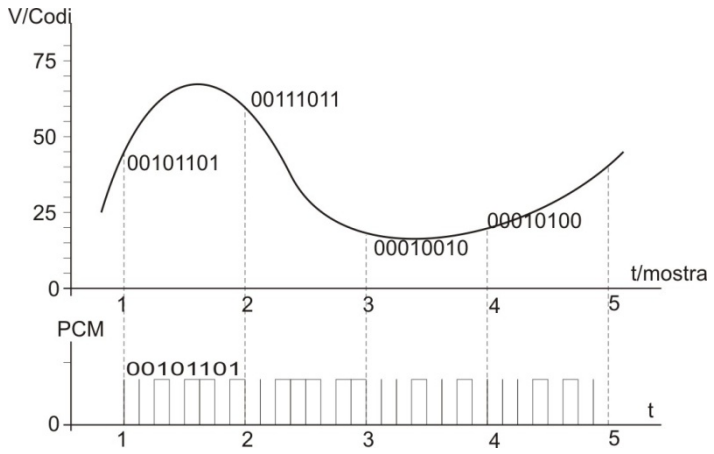


Figura 2.2. Codificació PCM

- El WAV és un format de guarda d'àudio propietari (IBM i Microsoft) que permet guardar àudio en el PC sense compressió amb característiques mono i estèreo, a diverses resolucions, i diferents velocitats de mostreig. D'acord amb la taula 2.2, per tenir bona qualitat (qualitat CD) convé emprar 44100 mostres/s. Com que normalment no empra compressió sol necessitar força memòria.
- DPCM (*Differential PCM*). En aquest cas enlloc de guardar la mostra es guarda la diferència entre una mostra i la següent. L'algorisme suposa que, donat que el so és continu, la diferència que hi haurà entre una mostra i la següent és petita, fet que permet emprar menys bits per guardar la informació. Amb el DPCM s'aconsegueixen compressions de l'ordre de 4.
- ADPCM *(Adaptive Differential Pulse Code Modification)*. És una modificació del codificador DPCM. Donat que el so segueix una ona quasi-predictible en aquest cas es guarda la diferència entre la mostra i la predita, amb el que s'aconsegueixen compressions majors.
- MPEG. MPEG (*Moving Picture Experts Group*) és un grup creat per a normalitzar la compressió en so i vídeo. MPEG-1 correspon a l'estàndard inicial de compressió d'àudio i vídeo, on s'introdueix el forma de compressió MP3. MPEG-2, MPEG-3 i posteriors comporten normes per l'edició de vídeo i so en alta definició.

  MP3 (o *MPEG-1 Audio Layer III*) és un format d'àudio digital basat en la compressió amb pèrdues dissenyat per a l'emmagatzematge i distribució d'àudio en temps real, podent arribar a la transmissió de 192 kbps. En aquest cas s'empra un nombre variable de bits en la representació (en dóna la precisió) segons la freqüència del senyal d'àudio, amb correspondència a la sensibilitat de l'oïda humana.
- Existeixen altres CODECs propietaris com *Windows Media* (Microsoft), *Real Networks* (Apple), ...

2.4 Representació d'imatges

La imatge representa una altra dada complexa de guardar en la computadora per la quantitat d'espai que requereix emmagatzemar tota la informació que conté i la velocitat de transmissió de dades que requereix. El mètode més simple d'obtenir una imatge és per captura mitjançant perifèrics com la màquina fotogràfica, el vídeo o l'escàner. També poden ser creades emprant programes gràfics.

Com tota dada, la imatge s'emmagatzema emprant un patró de bits determinat. En el cas de les imatges, però, es parla de dos formats: el format vectorial i el mapa de bits.

Format vectorial.

El format vectorial (figura 2.3) és adequat per emmagatzemar imatges (dibuixos) geomètrics que es descriuen fàcilment emprant figures geomètriques com línies, cercles, ... Els formats més típics emprats en aquests casos són el DXF (AutoCad, CorelDraw, ...), el EPS (Adobe), etc. L'avantatge del format vectorial és que una imatge ocupa poc espai en comparació am imatges basades en mapa de bits. Per exemple, per guardar una imatge que és una línia només cal guardar les coordenades del punt inicial i el punt final.

Figura 2.3. Format vectorial

Format mapa de bits.

Els mapes de bits s'empren per emmagatzemar imatges en les que cal guardar tot el detall, com és el cas de les fotos que realitzem amb una càmera digital. La imatge es divideix en multitud de punts cadascun dels quals conté les característiques pròpies d'aquella part de la imatge. L'agrupació d'aquests punts es guarda en un fitxer sota diferents formats. Els formats vectorials més coneguts són BMP (Microsoft), JPEG (grup JPEG), GIF (CompuServe), PNG (Consorci WWW), ...

La imatge té, per tant, infinits punts cadascun amb informació particular per guardar. La informació a guardar depèn del procés que es vulgui fer amb la imatge. Pot ser informació sobre l'escala de gris, el color, el to, ...

Com que no es pot guardar la informació d'infinits punts, la imatge es divideix en unitats d'imatge. La *resolució* de la imatge és el número de línies per número de columnes en què es divideix. La figura 2.4 mostra una imatge en dos formats diferents 10 x 6 i 15 x 25. És a dir, en 60 punts o en 375 punts. Evidentment, quan més punts més precisament es guarda la imatge.

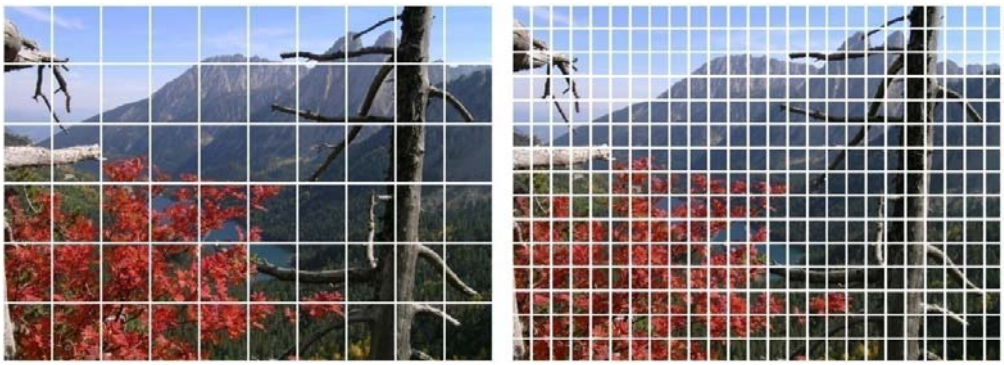


Figura 2.4. Format mapa de bits.

A cada unitat d'imatge se l'anomena *píxel* i se li associa la informació o atribut gràfic: blanc o negre, nivell de gris o color (en format RGB o CMYK, per exemple). Una imatge que es guarda en format blanc o negre necessita, evidentment, només 1 bit per codificar l'atribut del píxel, ja que només pot prendre dos valors. Quan la imatge es guarda en nivells de gris, aleshores normalment es sol reservar un byte d'informació per a cada píxel. Això vol dir que es poden tenir fins a 256 nivells possibles, anant del blanc fins al negre passant per tots els grisos.

La representació simple del color és sol fer per descomposició en els seus colors primaris vermell, verd i blau o *RGB*. És un model basat en la síntesis additiva dels components. La proporció en què es barreja cada color es sol representar en un byte. Per tant, el color d'un píxel es representa amb un vector de tres coordenades corresponents a la intensitat de cadascun dels tres colors primaris.

Amb aquesta informació és fàcil calcular, per tant, l'espai de memòria que es necessita per guardar una imatge. Per exemple, una imatge en format estàndard de 1024 x 768 píxels guardada en format RGB necessita més de 2,3MB de memòria. Si volguéssiu guardar en aquest format un vídeo d'1 hora de duració i que voleu passar a velocitat de 24 imatges/segon, podeu calcular quin espai de memòria necessiteu? Es comprova fàcilment que guardar imatge implica tenir molta memòria. A mode d'exemple es poden tenir les següents referències:

- Quan es parla de VGA en pantalles d'ordinadors (avui en dia sona quasi a prehistòria!) s'està parlant de pantalles de 640 x 480 píxels.
- Les targetes gràfiques SVGA tenen una resolució de 800 x 600 píxels.
- La norma actual WXGA treballa amb un format de 1366x768 o 1.049.088 píxels.
- En televisió d'alta definició s'està parlant d'imatges de 1920 x 1080 píxels/imatge a una velocitat de 30 imatges/segon.

La manera com la informació de la imatge es guarda en l'ordinador influeix també en la quantitat de memòria que es necessita per la imatge. Així, quan es guarda una imatge en el que s'anomena format *raw* s'està guardant tota la informació de cada píxel. No sempre és,però, necessari guardar tanta informació. A l'igual que passava amb el so , existeixen formats de compressió d'imatges que permeten estalviar força memòria. Entre els formats per guardar imatges més coneguts hi ha:

- *Document Exchange Format (DxF)*. És un estàndard de fitxers de text per a guardar dades en format vectorial per fitxers CAD. És el format original del conegut AutoCAD.
- *Encapsulated Postcript (EPS)*. Format vectorial. És una ampliació del llenguatge d'impressió Postcript per a imatges que permet la inserció d'imatges en diferent format com ara TIFF.
- Bitmap (BMP). És un arxiu de mapa de bits, originari de Microsoft, amb píxels emmagatzemats en forma de taula emprant colors reals. Per tant, és un format senzill, de gran qualitat però que ocupa molt d'espai, fet que no el fa adequat en planes web.

- *Tagged Image File Formats* (TIFF). Original d'Aldus i després Adobe el TIFF és pràcticament un estàndard per imatges d'alta qualitat (amb profunditat de color de 24 bits). Tot i que admet compressió també ocupa molt.
- *Joint Photographic Experts Group (JPEG).*Estàndard de codificació i compressió d'arxius gràfics amb pèrdua promogut pel grup JPEG. Té una qualitat d'imatge bona amb compressió ajustable per l'usuari. S'ha d'anar en compta en fer successives compressions ja que la pèrdua de qualitat és additiva.
- *Graphics Interchange Format (GIF).* És un format gràfic àmpliament emprat en la web per imatges i per animacions. Va ser creat el 1987 per CompuServe emprant, aleshores, un algorisme de compressió més eficient que els existents, el que permetia la descàrrega d'imatges grans en temps raonables. És un format que no té pèrdua de color per imatges de fins a 256 colors.
- *Portable Network Graphics (PNG).* Introduït per Unysis és una versió millorada de GIF. És un format gràfic sense pèrdua per bitmaps. Per animació APNG és un format que el suporta, podent quedar la imatge fixa en cas que el descodificador no el suporti.

2.5 Representació numèrica

La representació de números en el computador és un tema complex per la pròpia manera com pot ser tractat un número. Per entendre'ns, un número es pot representar en format tipus text, en format numèric sense signe, com a número amb signe, sota diferents codificacions, etc. I només coneixent d'antuvi la forma de representació emprada s'és capaç de llegir correctament el nombre escrit.

Evidentment que els nombres en el computador es poden representar en format tipus text, emprant la codificació ASCII. Però no és la seva forma natural ni és pràctic quan s'ha d'operar. De la mateixa manera que nosaltres fem servir de forma natural el sistema decimal per operar, en informàtica els nombres tenen diferents representacions que ens faciliten l'operativitat.

En aquest apartat es dóna una mica de llum respecte a aquest tema. Tot i així es veuran només els sistemes de representació indispensables per entendre com opera el computador en capítols posteriors. Es dóna per suposat que l'usuari coneix de forma mínima el tema de canvi de base en representació numèrica o, com a mínim, el canvi de base de decimal a binari i de binari a decimal.

2.5.1 Sistemes de numeració en informàtica

Un sistema de numeració és un conjunt de símbols i regles que permeten construir tots els nombres vàlids. Així, en base 10 (el sistema decimal) s'usa 10 símbols que corresponen a S = {0,1, 2, 3, 4, 5, 6, 7, 8, 9}. El computador treballa amb el sistema binari. Vol dir que només empra dos símbols S = {0,1}.

Apart del sistema binari, per comoditat d'operativitat en informàtica també s'empren els sistemes octal on S = {0,1, 2, 3, 4, 5, 6, 7} i l'hexadecimal S = {0,1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

Les mateixes regles que serveixen per crear els números en el sistema decimal serveixen en els sistemes binari, octal i hexadecimal. Les mateixes regles de pes posicional del nombre que s'apliquen en base 10, també s'apliquen en les altres bases. Per tant, en un número de n xifres cadascuna d'elles contribueix en el valor del nombre depenent de

- El valor de la xifra en sí mateix
- La posició que ocupa

Així, si en base 10 el nombre 527,03 correspon al nombre

$$5 \cdot 10^2 + 2 \cdot 10^1 + 7 \cdot 10^0 + 0 \cdot 10^{-1} + 3 \cdot 10^{-2} = 527{,}03$$

essent la posició -2 la posició corresponent a la de les centèsimes (10^{-2}), la -1 la posició de les dècimes (10^{-1}), posició 0 la corresponent a les unitats (10^{0}), la posició 1 la corresponent a les desenes (10^{1}), la 2 a la de les centenes (10^{3}), etc.,

la mateixa regla s'estableix en binari, octal i hexadecimal.

Exemple 1

En aquests dos casos següents cal observar que la representació d'un nombre en qualsevol base segueix les mateixes regles que el sistema de numeració en base 10. Només cal tenir en compta la base amb què es treballa.

$1AD.9_{(16} = 1\cdot16^{2} + A\cdot16^{1} + D\cdot16^{0} + C\cdot16^{-1}{}_{(10} = 1\cdot16^{2} + 10\cdot16^{1} + 13\cdot16^{0} + 12\cdot16^{-1}{}_{(10} = 429{,}75_{(10}$

$1001.01_{(2} = 1\cdot2^{3} + 1\cdot2^{0} + 1\cdot2^{-2}{}_{(10} = 8 + 1 + 0.25{}_{(10} = 9{,}25_{(10}$

Cal observar, també, que l'operativitat seguida en aquest exemple dóna el canvi de base d'un número expressat en qualsevol base (en el cas de l'exemple primer en hexadecimal i després en binari) a base decimal.

2.5.2 Representació numèrica en format tipus text

Senzillament s'empra el codi ASCII per a representar un número.

Exemple 2

El número 3567 es representaria amb la seqüència de bits corresponents als números 33 35 36 37.

En quant a espai de memòria es necessita un byte per a cada número. Evidentment, no es pràctic per operar.

Exemple 3.

Considerar el cas que, per teclat, s'entra el número 12345. Sabríeu dir quantes bytes ocupa?

Per a resoldre l'exercici primer cal saber si el número es guarda com a nombre o com a cadena.

Si el número es guarda com a enter es té que en base dos ve donat per

$12345_{(10} = 11\ 0000\ 0011\ 1001_{(2}$

Això vol dir que es necessiten 14 bits. La representació necessita dos bytes.

Si el número es guarda com a cadena aleshores s'ha de fer el canvi de cada número a codi ASCII, el que dóna:

12345 = 49 50 51 52 53

Es necessiten, per tant, 5 bytes.

2.5.3 Classificació de les representacions numèriques en informàtica.

Les representacions numèriques més emprades en informàtica són les següents:

- Per a nombres enters es troben les representacions:
  - Enters sense signe
  - Enters amb signe. Aleshores es parla de les representacions:
    - Amb signe i magnitud
    - Complement a 1

  - Complementa a 2.
  - Esbiaixada o en excés
- Representació de dígits decimals o BCD
- Representació de nombres reals. Hi ha dues representacions fonamentals:
  - Representació en punt fix
  - Representació en punt flotant

2.5.4 Representació de nombres enters sense signe

És la representació de nombres naturals. Els bits del número representen el número expressat en binari natural.

Exemple 4.

El número $10110001_{(2}$ correspon al número $177_{(10}$.

La taula 2.4 mostra els nombres naturals que es poden tenir en una representació binària emprant tres dígits.

Número binari	Número decimal
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Taula 2.4. Enters sense signe

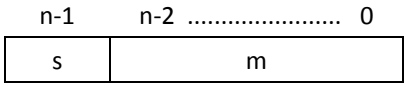
Es compleix que els números més petit i més gran vénen donats per:

- El número més petit és el 0: $N_{mínim} = 0$.
- El número més gran correspon a la base menys 1: $N_{màxim} = 7 = 2^N-1$

2.5.5 Representació de nombres enters amb signe i magnitud

Entre les representacions d'enters amb signe es parla usualment de les representacions en signe i magnitud, complement a 1 i complementa 1. En aquest punt cal anar en compta per què la diferència entre les diferents representacions està en la representació dels nombres negatius. Per tant, tot i que es parlarà de diferents representacions, quan els nombres són positius la representació és la mateixa. Però la de nombres negatius és totalment diferent. Aquest fet porta sovint a confusió a gent no avesada a treballar-hi.

També es compleix que tots els números amb signe dediquen el bit més significatiu al bit de signe. Per tant, el format dels nombres amb signe és:



on *s* indica el bit de signe i *m* indica a la resta de dígits que correspon a la magnitud.

Representació de nombres amb signe i magnitud

En la representació de nombres amb signe i magnitud es compleix que:

- El bit de signe (bit n-1) és 0 quan el nombre és positiu. És 1 quan el nombre és negatiu.
- La resta de dígits (bits de n-2 a 0) representen el valor absolut del nombre en binari.
- La representació d'un nombre en SM ve donada per [(signe) r_{n-2} ... r_1 r_0]

La taula 2.5 mostra els nombres naturals que es poden tenir en una representació binària emprant tres dígits.

Número binari	Número decimal
000	0
001	1
010	2
011	3
100	-0
101	-1
110	-2
111	-3

Taula 2.4. Enters sense signe

Es compleix que els números més petit i més gran vénen donats per:

- El número més petit és: $N_{mínim} = -3 = -(2^{n-1}-1)$.
- El número més gran correspon a la base menys 1: $N_{màxim} = 3 = 2^{n-1}-1$

Sobre l'ús de la representació en signe i magnitud cal tenir present que:

- Hi ha doble representació del 0.
- L'aritmètica és poc operativa. Per exemple, per fer una resta de dos nombres, els passos a executar són: primer s'ha de mirar quin és el nombre més gran per posar-lo com a subtrahend; després s'ha de fer la resta; i, finalment, el signe és el del subtrahend.

2.5.6 Nombres enters en complement a la base disminuïda o complement a 1

El complement a la base disminuïda es troba restant el número de la base menys 1 (el que s'anomena base disminuïda). En base 2 agafa el nom de complement a 1.

En base 2 queda explicitat com a:

- $C1(N) = N_{(2}$ quan $N \geq 0$
- $C1(N) = (2^n - 1) - |N|_{(2}$ quan $N < 0$

Per a calcular el C1 es pot emprar un dels dos mètodes següents (que en sí són equivalents):

- Utilitzar estrictament la definició anterior.
- Invertir els bits. És a dir, canviar els 0s per 1s i els 1s per 0s.

Exemple 5.

Trobar el complement a 1 dels nombres 11 i -11, suposant que es treballa amb nombres de 8 bits.

- La representació del nombre positiu és directa. Només cal fer el canvi de base 10 a base 2 del nombre i posar-lo en format de 8 bits. Així: C1(11) = 00001011.
- Per trobar el C1(-11) es pot agafar el número en positiu (que es té de l'apartat anterior) i restar-lo de la base menys 1.

  Base menys 1 → 11111111
  C1(11) → - 00001011
  11110100

  Es pot observar que el C1(-11) es pot trobar invertint directament tots els bits.

Propietats del C1.

Algunes característiques del C1 són:

- El C1(C1(n)) = n. Aquesta propietat és molt simple i alhora porta molts conflictes de mala interpretació en el sentit següent: quan es parla de representació en C1 ens estem referint al sistema de treball, el que vol dir que és la representació que s'utilitza el que diu el sistema amb el que es treballa, no que el número sigui positiu o negatiu!
- Es compleix que el signe és 0 si el nombre és positiu, i 1 si el nombre és negatiu.
- El rang de representació per nombres de n bits és:

  $N_{mínim} = -(2^{n-1}-1)$

  $N_{màxim} = 2^{n-1}-1$

- Com la representació en signe i magnitud també té doble representació del 0, que es dóna quan tots els bits són 0 o tots són 1.

La taula 2.5 mostra els nombres que es poden representar en C1 emprant tres dígits binaris.

Número en C1	Número decimal
000	0
001	1
010	2
011	3
100	-3
101	-2
110	-1
111	-0

Taula 2.5. Enters sense signe

2.5.7 Nombres enters en complement a la base o complement a 2

El complement a la base es troba restant el número de la base. En base 2 agafa el nom de complement a 1.

En base 2 queda explicitat com a:

- $C2(N) = N_{(2}$ quan $N \geq 0$
- $C2(N) = 2^n - |N|_{(2}$ quan $N<0$

Per a calcular el C2 es pot emprar un dels dos mètodes següents:

- Utilitzar estrictament la definició anterior.
- Trobar el C2 del número i aleshores sumar-li 1.

- Hi ha una regla mnemotècnica que diu que, començant en el dígit menys significatiu, es mantenen tots fins que es troba el primer 1. A partir del dígit següent s'inverteixen tots.

Exemple 6.

Trobar el complement a 2 dels nombres 36 i -36, suposant que es treballa amb nombres de 8 bits.

- La representació del nombre positiu és directa. Només cal fer el canvi de base 10 a base 2 del nombre i posar-lo en format de 8 bits. Així: C2(36) = 00100100.
- Per trobar el C2(-36) es pot agafar el número en positiu (que es té de l'apartat anterior) i restar-lo de la base.

Base	→	100000000
C2(36)	→	- 00100100
		11011100

Es pot observar que es compleix la regla mnemotècnica.

Propietats del C2.

Algunes característiques del C2 són:

- El C2(C2(n)) = n
- Es compleix que el signe és 0 si el nombre és positiu, i 1 si el nombre és negatiu.
- El rang de representació per nombres de n bits és:

$N_{mínim} = -2^{n-1}$

$N_{màxim} = 2^{n-1}-1$

- El 0 té representació única.

La taula 2.6 mostra els nombres que es poden representar en C1 emprant tres dígits binaris.

Número en C2	Número decimal
000	0
001	1
010	2
011	3
100	-4
101	-3
110	-2
111	-1

Taula 2.6. Enters sense signe

2.5.8 Representació de nombres enters amb *biaix* o *excés-b*.

És un sistema de numeració que suma un biaix *b* al número, normalment 2^n-1. També s'anomena d'excés-b. El resultat és que un valor queda representat per un número sense signe que és b cops més gran que el que és.

El número queda explicitat com a:

- $Cb(N) = N_{(2} + b = N_{(2} + 2^n-1$

Com a conseqüència el 0 queda representat pel número b, i el número –b és el tot zeros.

Aquesta representació s'empra fonamentalment per calcular l'exponent de la representació de números reals en punt flotant.

Exemple 7.

Trobar la representació esbiaixada del nombre 36 i -36, suposant que es treballa amb nombres de 8 bits.

El biaix en aquest cas és b = 01111111.

Aleshores Cb(36) és:

Biaix	→	01111111
- 36	→	+ 00100100
		10100011

I el Cb(-36) és:

Biaix	→	01111111
- 36	→	+ (- 00100100)
		01011011

2.5.9 Taula comparativa de la representació de nombres enters.

La taula 2.7 mostra la representació dels nombres enters de 4 bits emprant les 4 representacions anteriors.

Número	SM	C1	C2	Cb (b=7)
8	----	----	----	1111
7	0111	0111	0111	1110
6	0110	0110	0110	1101
5	0101	0101	0101	1100
4	0100	0100	0100	1011
3	0011	0011	0011	1010
2	0010	0010	0010	1001
1	0001	0001	0001	1000
0	0000	0000	0000	0111
-0	1000	1111	0000	1111
-1	1001	1110	1111	0110
-2	1010	1101	1110	0101
-3	1011	1100	1101	0100
-4	1100	1011	1100	0011
-5	1101	1010	1011	0010
-6	1110	1001	1011	0001
-7	1111	1000	1001	0000
-8	----	----	1000	

Taula 2.7. Representació de nombres enters

2.5.10 Representació de nombres reals en punt fix.

La representació de nombres reals en el computador es pot realitzar emprant dues representacions:

- Representació en punt fix. Estableix un nombre determinat de bits per representar el número dels que un nombre fix es reserven per la part fraccionària.

- Representació en punt flotant. Empra una representació fixa en nombre de bits en la que el punt fraccionari queda flotant dintre la representació.

  La representació en punt flotant es tractarà en el proper apartat.

La representació en punt fix empra un nombre predeterminat de bits fixant un nombre de bits m destinats a la part fraccionària del número. La part fraccionària queda implícita:

$$x_{n-1}\, x_{n-2} \ldots x_0 . x_{-1}\, x_{-2} \ldots x_{-m}$$

El rang de representació en punt fix ve donat pels bits de la part entera. El rang de precisió ve donat pels bits de la part fraccionària.

Exemple 8.

En aquest exemple es representa el número -105.321 suposant que es treballa amb 16 bits/paraula. Es destinen 8 bits a la part entera i 8 a la fraccionària.

Els passos que s'han de seguir per representar el número són els següents:

- Primer es representa en binari la part entera i la part fraccionària per separat:

  Part entera: $105_{(10} = 1101001_{(2}$

  Part fraccionària: $.321_{(10} = .01010010_{(2}$

- Aleshores es representa el número en la representació que s'està utilitzant:

  En signe i magnitud el número queda com: SM(-105.321) = $11101001.01010010_{(2}$. Es pot observar que el valor absolut del número correspon a la concatenació de la part entera i la fraccionària.

  En complement a 1 i complement a 2 es fa una operació similar a l'anterior. El número es considera com a un tot en valor absolut. Aleshores es realitza l'operació que requereix d'acord amb el signe del número. Així,

  En complement a 1 el número queda com: C1(-105.321) = $10010110.10101101_{(2}$

  I en complement 2: C2(-105.321) = $10010110.10101110_{(2}$

  El punt decimal només es mostra a efectes de visualització del número. Cal tenir present que en realitat el número és un tot i no hi és.

2.5.11 Extensió del bit de signe.

En qualsevol de les representacions introduïdes fins ara s'ha suposat de partida que el número es representava d'acord a un número determinat de bits prèviament definit.

A vegades, però, s'han de realitzar transformacions del número que implica passar a un nombre major de bits.

Quan el número és positiu l'extensió no té cap efecte per què es tracta, senzillament, d'afegir 0's a l'esquerra.

Quan el nombre és negatiu, però, l'extensió es fa posant 1's. És el que s'anomena extensió del bit de signe.

Exemple 9.

El número de 4 bits $1011_{(2}$, expressat en complement a 2, correspon al número enter -5.

Si es demana passar aquest número a 8 bits, aleshores no es poden posar 0's a l'esquerra, sinó que s'ha d'estendre el bit de signe. El número queda com $11111011_{(2}$.

2.5.12 Representació de nombres reals en punt flotant.

La representació de nombres reals en punt flotant utilitza com a base la notació científica del nombre expressat en binari.

En notació científica (punt flotant) un número N ve representat per $N = m \cdot B^e$, i a on m és la mantissa, e és l'exponent i B correspon a l'arrel.

L'avantatge de la notació científica és que permet representar nombres grans i petits tot mantenint una precisió acceptable.

La representació, tant en decimal com en base 2, segueix la mateixa estructura.

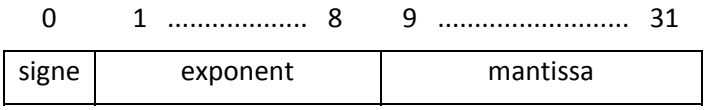
Exemple 10.

En decimal la notació posicional permet saber que els següents representacions corresponen al mateix número: $1234{,}56 = 0{,}123456 \cdot 10^4 = 123456 \cdot 10^{-2}$

La mateixa argumentació serveix en el sistema binari. Els següents número són el mateix: $1001{,}01 = 0{,}100101 \cdot 2^4 = 100101 \cdot 2^{-2}$

En base 2 s'empra l'estàndard IEEE 754 per a representar dades reals en punt flotant. S'especifica la notació per nombres de 32 bits i nombres de 64 bits. Aquí s'explica la representació emprada en nombres de 32 bits.

El nombre normalitzat (estàndard) consta de 32 bits distribuïts en el format signe, exponent i mantissa següent:



La notació pressuposa la base 2. La representació en punt flotant exigeix que els nombres es representin en notació normalitzada. En notació normalitzada es compleixen les següents propietats:

- El bit de signe val 0 si el nombre és positiu, i 1 quan és negatiu.
- L'exponent es representa en notació entera esbiaixada (*biased*) a $(2^{ne-1}-1)$, on ne és el nombre de bits de l'exponent. Com que l'exponent és un número de 8 bits, l'excés o biaix és de 127. Per tant, la representació de l'exponent ve donat per

  exponent = biaix + E = 2^{ne-1} -1 + E, on E és el valor de l'exponent
- La mantissa (magnitud) també s'ha de normalitzar. Es representa en valor absolut, ajustada de forma que l'1 més significatiu es trobi en la posició de les unitats. Es compleix que la mantissa es troba entre $2 > m \geq 1$.

Exemple 11.

S'ha de representar el número $21 \cdot 2^{-24}$ en punt flotant normalitzat. De fet, es tracta d'un número ja preparat en tant que la base ja està expressada en binari. El cas en què la base és decimal necessita d'un pas previ de transformació de base 10 a base 2.

El primer pas és escriure el número en binari i normalitzar-lo: $21 \cdot 2^{-24} = 10101 \cdot 2^{-24} = 1{,}0101 \cdot 2^{-20}$

En el segon pas es calcula l'exponent esbiaixat: exponent = 127 – 20 = 107 = 01101011

Després es troba la mantissa: mantissa = 0101. Cal tenir present que l' 1 de les unitats se suposa i no es representa.

Finalment la representació del número és:

0	01101011	0101000...0

La representació en punt flotant té un conjunt de situacions que s'han de preveure.

- Quan l'exponent és 0, la mantissa es posa denormalitzada. És a dir, el 1 se suposa no implícit.

0	00000000	m ≠ 0

- El número zero es representa amb exponent=0 i mantissa = 0

0	00000000	000...0

- Quan els bits de l'exponent són tots 1, es poden tenir diferents situacions:
  - Si la mantissa és 0, es té un número $\pm\infty$

0/1	11111111	000...0

  - Quan la mantissa no és 0, es tracta d'un codi *NaN*.

0	11111111	m ≠ 0

En representació en punt flotant els punts límits vénen donats per les següents representacions:

- El nombre més gran és

0	11111110	111...1

- El nombre més petit normalitzat és el

0	00000001	000...0

- I el número més petit desnormalitzat és

0	00000000	000...01

2.5.13. Punt fix enfront punt flotant

En una representació de 32 bits en punt fix es poden representar la mateixa quantitat de nombres que en una representació en punt flotant de 32 bits. La diferència entre les dues representacions està en els nombres que es representen en cada cas.

Representació en punt fix.

En la representació en punt fix els màxims valors representables s'obtenen quan no s'empra cap nombre fraccionari. El rang de representació va de $2^{(n-1)}$-1 pels nombres positius fins a $-2^{-(n-1)}$ en nombres negatius (aquest mínim es dóna en representació en complement a 2). La figura 2.5 mostra aquests límits en el cas que n sigui 32.

En el cas de representar nombres en punt fix els majors nombres fraccionaris representables en una representació de n bits són $1-2^{-(n-1)}$ pels positius i -1 pels negatius (aquest cas es dóna només en complement a 2). La resolució és $2^{-(n-1)}$.

Representació en punt flotant de 32 bits

La part inferior de la figura 2.5 mostra el rang de representació en punt flotant. El nombre de valors que es poden representar és el mateix que en punt fix, només que a diferent resolució. En aquest cas, però, la resolució és variable segons la zona en la que es troben els valors. De fet es poden representar tants números de valor absolut superior a 1 com inferiors (zones A i B, respectivament, en la figura).

Els valors de la figura 2.5 en punt flotant fan referència al format de flotant normalitzat (IEEE 754). S'observa que el rang de valors representable és $\pm (1 - 2^{-23}) \cdot 2^{\pm 128}$. El punt flotant permet representar nombres molt petits amb gran precisió i nombres molt grans amb poca precisió.

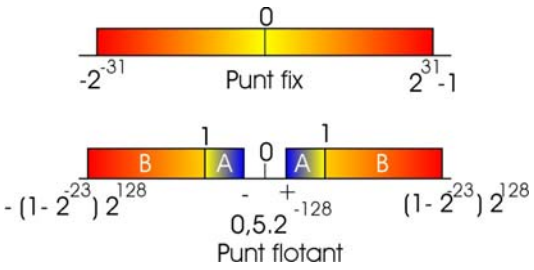


Figura 2.5. Comparació de la representació en punt fix enfront punt flotant.

2.5.14 Codis especials

El codi BCD (*Standard Binary Coded Decimal*)

El codi BCD és un codi alfanumèric de 6 bits més paritat introduït en els anys 60 que permet representar més valors que els pròpiament numèrics. Admet, per tant, 64 possibles valors. La figura 2.6 mostra el format del codi. Es pot veure que dels 6 bits emprats per a representar el caràcter, els dos bits més significatius corresponen a una clau que indica el tipus de caràcter representat. El bit més significatiu és el bit de paritat. S'empra paritat parella, és a dir, la suma de tots els uns ha de ser un nombre parell.

Paritat(1 bit)	Clau (2 bits)	Codi (4 bits)
P	C1 C0	r3 r2 r1 r0

Figura 2.6. Format del codi BCD

Tot i que el codi BCD ha quedat obsolet en el que respecte a representació alfanumèrica, encara és usat en la representació dels nombres decimals en binari (taula 2.9). La clau en el cas dels caràcters numèrics és 00.

Número	Codificació
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Taula 2.9. Valors numèrics del codi BCD

El codi Gray

El codi de Gray és un sistema de numeració que es caracteritza per mantenir una diferència d'1 dígit entre dos codis successius. És el que tècnicament es diu que la distància de Hamming és 1.

La taula 2.10 mostra el codi de Gray de 4 dígits.

Número	Codificació
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Taula 2.10. Codi de Gray

El codi de Gray s'empra en situacions sorolloses en les que sigui fàcil introduir atzars en els canvis de senyal, com pot ser en el disseny de circuits digitals seqüencials. El fet de tenir una distància mínima entre codi i codi ajuda a evitar que es produeixen errors de transmissió. Per exemple, la figura 2.7a mostra un codificador rotatori Gray, emprat per convertir posicions angulars a codi binari, que pot ser emprat en el càlcul de velocitats en eixos rotatoris.

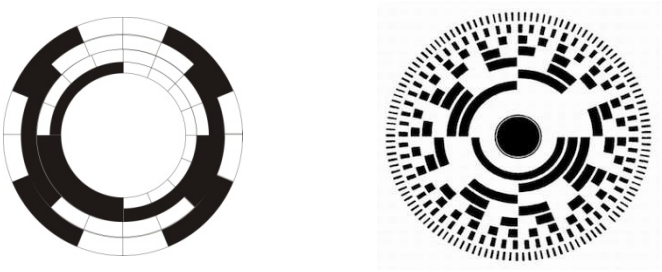


Figura 2.7. a) Codificador rotatori Gray. b) Codificador rotatori binari

Codis redundants

En certes aplicacions informàtiques, en teoria de la informació i en comunicacions és normal utilitzar codis detectors i correctors d'errors per assegurar la transmissió correcta dels senyals enviats. Un codi detector d'error és aquell que permet detectar la transmissió d'un error. Un codi corrector va més enllà: corregeix (fins a un cert límit) l'error detectat.

Fins ara els codis que s'han vist han estat codis sense detecció ni correcció d'errors, a excepció del codi BCD que és capaç de detectar la transmissió d'un bit erroni mitjançant el bit de paritat. La manera com ho fa és el mètode que sempre es segueix: introduint bits addicionals portadors, no d'informació, sinó de la variació que es produeix en la transmissió del senyal.

Amb un bit addicional el codi BCD permet detectar que s'ha variat almenys un bit en la transmissió. Variacions de més bits no són detectables quan entre elles es compensen.

La taula 2.11 mostra l'ús d'un codi amb bit de paritat. Es suposa que és un codi de 5 bits més un bit de paritat. El bit de paritat es posa a 1 per fer que la suma del nombre sempre tingui un nombre d'uns parell o senar,segons es tracti de paritat parell o senar respectivament

Codi inicial	Codi amb paritat parella	Codi amb paritat senar
00000	000000	100000
01100	001100	101100
11010	111010	011010

Taula 2.11

Amb més d'un bit addicional al mínim nombre de bits que permeten codificar tots els símbols es pot començar a implementar codis correctors d'errors.

Codis correctors de Hamming.

Un codi de Hamming és un codi detector i corrector d'error que funciona afegint al costat del missatge més d'un bit detector/corrector d'error. Per a què la codificació permeti corregir l'error cal que l'ordenació dels bits produeixi diferents resultats segons l'error que es produeixi, de manera que el bit erroni pugui ser identificat.

És comú parlar del codi de Hamming (7, 4) que indica que és un codi de 7 bits dels quals, 4 pertanyen al missatge i 3 als bits detectors/correctors d'error. Pot corregir qualsevol error d'un sol bit, però falla quan hi ha errors en més d'un bit ja que la paraula transmesa es confon amb una altra amb error en un sol bit, sent corregida, però de forma incorrecta

El codi de Hamming estès afegeix un bit més permetent detector errors de dos bits però no corregir-los.

2.6 Resum del capítol.

El capítol ha introduït els formats bàsics de representació de la informació en el computador, especificant com es guarda la informació segons sigui text, imatge o números.

Respecte a la representació de números s'han vist els formats més emprats de representació. Tot i que per aplicacions específiques es poden emprar altres representacions, en computadors genèrics el format més emprat és el de complement a 2.

En l'apartat d'exercicis resolts es mostra com es fa la suma i la resta en signe i magnitud i complement a 2.

2.7 Exercicis resolts.

Durant el capítol s'han introduït els formats més usuals de representació numèrica. Amb aquest exercici es mostra com es fa la suma en signe i magnitud i en complement a 2, mostrant l'avantatge que ofereix aquest respecte a signe i magnitud.

Exercici 1.

Siguin els dos números A = +41 i B = -28. Es demana realitzar les operacions A+B, A-B, -A+B i –A-B en representació de signe i magnitud.

Solució.

2.7 Exercicis resolts.

La representació en signe i magnitud de dos enters (apartat 2.5.5) es troba buscant el nombre binari de la magnitud i posant el signe en la posició més significativa de la representació. D'aquesta forma es troba directament que:

A = 00101001
B = 10011100

La suma/resta de dos números en signe i magnitud segueix el següent algorisme:

i) Es cerca el nombre de magnitud major i es posa en la part superior
ii) Es realitza la operació i el signe del resultat és el del nombre posat en la part superior.

D'aquesta manera:

- L'operació A + B implica sumar a un número positiu un número negatiu, pel que es resten les magnituds:

```
  0 0101001
- 1 0011100
  0 0001101  (= 13)
```

- L'operació A - B implica restar a un número positiu un número negatiu. En restar un nombre negatiu aquest es converteix en positiu, pel que es sumen les magnituds i el resultat és positiu:

```
  0 0101001
+ 0 0011100
  0 1000101  (= 69)
```

- L'operació -A + B implica negar un número positiu i sumar-li un altre negatiu. Per tant, el resultat serà negatiu sumant les seves dues magnituds:

```
  1 0101001
+ 1 0011100
  1 1000101  (= -69)
```

- Finalment l'operació -A - B implica convertir a negatiu el nombre major i restar-li un nombre negatiu, fet que es converteix en restar el nombre menor del major quedant negatiu el resultat:

```
  1 0101001
- 0 0011100
  1 0001101  (= - 13)
```

Es pot observar que l'operativitat amb signe i magnitud és complicada. Es veurà ara que el complement a 2 simplifica aquesta operativitat.

Exercici 2.

Amb els dos mateixos números de l'exercici anterior es demana realitzar les mateixes operacions en complement a 2.

Solució.

La suma/resta de dos números representats en C2 és directa. A més és molt fàcil si es considera que l'ordenador que opera en C2 no sap restar, sinó que només suma números representats en complement a 2:

A- B = A + C2(B)

Aleshores, emprant aquesta regla en les operacions demanades, el més adequat és cercar les representacions positives i negatives dels dos nombres anteriors i efectuar sempre l'operació descrita.

Així, els passos són:

2.7 Exercicis resolts.

i) Trobar les representacions positiva i negativa dels dos números.

A = +41 = 00101001 → Representació negativa: -A = C2(A) = 11010111
B = -28 = C2(00011100) = 11100100 → Representació positiva: -B = C2(B) = 00011100

ii) Efectuar les operacions.

A + B = A + B = (recordar que B és un número negatiu!)

```
  00101001
+ 11100100
  00001101   → (+ 69)
```

A – B = A + C2(B) =

```
  00101001
+ 00011100
  01000101   → (+ 69)
```

-A + B = C2(A) + B =

```
  11010111
+ 11100100
  10111011   → (- 69. És fàcil veure que C2(10111011) = 01000101 = +69)
```

-A – B = C2(A) + C2(B) =

```
  11010111
+ 00011100
  11110011   → (- 13. És fàcil veure que C2(11110011) = 00001101 = +13)
```

S'observa, per tant, que l'operativitat en C2 simplifica enormement la suma i resta en operacions simples en el computador.

Exercici 3.

Treballant amb representació de C2 i 8 bits, i donats els números A = 58, B = 75 i C = -55, realitzar les operacions A+B i A+C.

Solució.

i) Primer es troben les representacions dels nombres:

A = 58 = 00111010
B = 75 = 01001011
C = -55 = C2(00110111) = 11001001

ii) Seguidament es realitzen les operacions.

A + B =

```
  00111010
  01001011
  10001101
```

...i s'obté un número negatiu! De fet, en sumar 58+75 = 133 s'excedeix la representació de 8 bits en C2. Es detecta per què el resultat és de signe diferent al dels dos operands.

Això no pot passar mai en sumar números de signe diferent o restar números del mateix signe. Analitzem ara la següent operació:

A + C =

```
   00111010
+  11001001
1) 00000011 (= 3)
```

Cal fixar-se que es genera un 1 de carreteig en el novè número de la representació, però el signe continua sent vàlid (el signe és coherent, puig estem sumant un número negatiu a un positiu!). El bit de carreteig no té més incidència en la suma en complement a 2, i no s'ha de confondre amb l'excés de representació en l'operació.

Exercicis.

Exercici 4.

Treballant amb punt fix de 8 bits en la part entera i 4 en la part fraccionària, realitzar l'operació A-B en C, donats A=16.4 i B=16.25. Hi ha error en l'operació?.

Solució.

i) Primer es troben les representacions binàries dels nombres:

$A = 16.4_{(10} = 00010000.0110_{(2}$

$B = 16.25_{(10} = 00010000.0100_{(2}$

ii) Acte seguit es pot fer la resta.

$A - B = A + C2(B) =$ 00010000.0110
11101111.1100
00000000.0010

iii) L'error que es produeix en el resultat és fruit de l'error de conversió del nombre A que es propaga. Es pot observar que ve donat per:

$Resultat_{(10} - Resultat)$
$= 0.15 - ConversioABase10(00000000.0010) =$
$= 0.15 - 0.125 =$
$= 0.075$

Es tracta, proporcionalment, d'un error molt gran, que es deu a la manca de precisió en el canvi de base.

En certs casos pot ser que l'error entre operands es cancel·li, però no és el cas de l'exemple. En qualsevol cas s'haurà de tenir sempre present que la conversió entre bases pot introduir importants errors.

Exercicis.

1. Donats els següents números binaris, i suposant que es treballa amb 5 bits, donar el seu valor decimal suposant que estan representats en SM, C2 i en notació esbiaixada.

Número	...si en SM	...si en C2	...si esbiaixat
01010			
10010			
10001			
11111			
00000			
01111			
10000			

2. Donats els números A = -39, B = 98, C = -127 i D = 0, trobar la seva representació en SM, C1 i C2.

3. Siguin els números A = 15 i B = -25. Treballant amb representació de 6 bits, realitzar les operacions A+B, A-B, -A+B i –A-B en SM i en C2.

4. Treballant amb representació de punt fix amb 8 bits per la part entera i 4 en la fraccionària...

- Trobar la representació dels nombres A = +25.85 i B = -70.35 en SM i C2.
- Realitzar les operacions en C2 següents: A+B, -A-B.
- Determinar l'error de conversió en els operands i en els resultats.

5. Representar els números $A = 3.25 \cdot 2^{15}$ i $B = -125.125 \cdot 2^{-15}$ en punt flotant normalitzat.

Annex: Pas genèric d'un número decimal a punt flotant.

En els exemples de pas a punt flotant que s'han posat fins ara sempre s'ha procurat que la base fos potència de 2 per a simplificar els càlculs.

El pas genèric d'un nombre decimal a punt flotant requereix d'una transformació matemàtica que, tot i no ser simple, sí que és ferragossa.

Amb l'exemple següent es veuen tots els passos que cal seguir per realitzar aquesta transformació. Es convertirà el nombre $A = 4{,}4668 \cdot 10^{10}$ a punt flotant.

Pas 1. Es posa el nombre en format decimal amb la primera xifra no zero en la primera posició decimal.

$$A = 0{,}44668 \cdot 10^{11}$$

Pas 2. Es converteix la base 10 a base 2. El procediment és:

$$10^x = 2^y$$

$$\log 10^x = \log 2^y$$

$$x \cdot \log 10 = y \cdot \log 2$$

$$y = x \cdot \log 10 / \log 2$$

Substituint:

$$y = 11 \cdot \log 10 / \log 2 = 36{,}5412$$

Pas 3. Es reescriu el número amb potències de 2:

$$A = 0{,}44668 \cdot 10^{11} = 0{,}44668 \cdot 2^{36{,}5412} = 0{,}44668 \cdot 2^{0{,}5412} \cdot 2^{36} = 0{,}44668 \cdot 1{,}455 \cdot 2^3 = 0{,}65 \cdot 2^3$$

D'on el nombre amb mantissa expressada en base 2 queda com:

$$A = 0{,}65 \cdot 2^3 = 0{,}10\underline{1001}\ 2^3 = 1{,}0\underline{1001}\ 2^2$$

Pas 4. ...i ja només queda expressar-lo en punt flotant.

Capítol 3

INTRODUCCIÓ ALS COMPONENTS DIGITALS

Els sistemes digitals són la base física del computador i és una matèria fonamental en l'aprenentatge del funcionament del computador. La matèria de sistemes digitals és, de per sí mateixa, una matèria gran que s'ha d'aprendre metodològicament en un curs i que no es pot veure en només un capítol.

Per altra part, però, en una introducció als computadors, s'han de conèixer un conjunt de conceptes bàsics dels sistemes digitals si es vol aprofundir mínimament en el funcionament de la CPU, el nucli dels computadors. És el que pretén fer aquest capítol: introduir de forma somera aquells conceptes essencials dels sistemes digitals que són indispensables per a entendre el funcionament elemental del processador.

En aquest sentit, i després de llegir aquest capítol, s'espera que quedin clars els següents conceptes:

Circuit combinacional, i els mòduls combinacionals bàsics que s'empraran en els propers capítols.
El concepte de flip-flop (com a punt de memòria) i la seva forma de funcionar.
Els mòduls seqüencials, especialment el registre i el comptador, que formen la base de la lògica que fa funcionar el processador.
I el diagrama de temps, com a mètode mecànic de visualització de l'evolució d'un sistema digital en el temps que permetrà entendre l'evolució temporal que sofreixen els components del processador durant l'execució d'un programa.

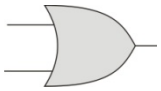
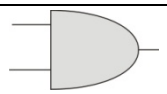
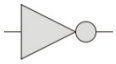
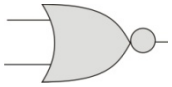
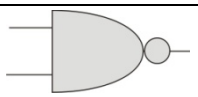
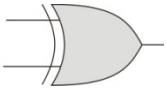
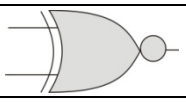
3.1 Introducció a la lògica digital

En els capítols previs s'ha anat introduint que el computador funciona internament mitjançant la lògica digital binària que empra dos símbols com a sistema de numeració base en el seu funcionament.

A nivell matemàtic, el sistema que empra dos símbols es coneix com a *àlgebra de Boole* en honor al seu inventor George Boole, matemàtic i filòsof anglès. Boole era de fet un filòsof que estava obsessionat en voler simplificar l'argumentació de problemes lògics i filosòfics que conduïssin cap a dos estats, el cert i el fals. Per resoldre el problema va desenvolupar un sistemes de normes (descrites en el treball *An Investigation of the Laws of Thought*) que li permetien manipular i

simplificar els arguments a aquests dos estats. Aquestes normes que treballen amb dos estats van ser emprats a començaments del s. XX per desenvolupar la lògica digital moderna.

L'àlgebra de Boole, que a nivell matemàtic es descriu a partir d'un conjunt de postulats i propietats, treballa amb les tres operacions elementals descrites en la taula 3.1: la unió, la intersecció i la negació. La unió i la intersecció són operacions binàries, mentre que la negació és una operació unària.

Operació	**Nom**	**Operació**	**Símbol**
Unió	OR	A+ B	
Intersecció	NAND	A · B	
Negació	NOT	$\overline{A}$	
Operacions combinades			
Unió negada	NOR	$\overline{A + B}$	
Intersecció negada	NAND	$\overline{A \cdot B}$	
Exclusió	XOR, OREX	$A \oplus B$	
Exclusió negada	NXOR	$\overline{A \oplus B}$	

Taula 3.1. Operacions elementals de l'àlgebra de Boole.

El que és meravellós de l'àlgebra de Boole és que aquestes tres operacions amb els postulats de l'àlgebra formen la base sobre la que es desenvolupa tota la teoria dels sistemes digitals, i que és el fonament del computador.

Les tres operacions bàsiques donen lloc a les respectives operacions negades. Emprant la negació en les dues operacions OR i AND, s'obtenen les seves negades NOR i NAND, respectivament. Existeixen, a més, dues operacions addicionals:

- L'exclusió. El resultat és 1 sempre que les dues entrades siguin diferents, si no és 0.
- L'exclusió negada. És la negada de l'exclusió. El resultat és 1 sempre que les dues entrades siguin iguals, si no és 0.

Cadascuna d'aquestes operacions té associat un símbol (darrera columna) que permet identificar fàcilment la operació que es realitza quan es treballa en circuits digitals. A nivell de sistema digital aquests símbols se'ls anomena *portes lògiques*.

A nivell matemàtic aquestes operacions es poden expressar a través de les seves taules de veritat. La taula de veritat s'obté posant 0's i 1's a les entrades i realitzant l'operació indicada.

La taula 3.2 mostra les taules de veritat per a totes les portes lògiques. Es pot observar que totes les portes operen amb dues entrades, excepte la inversió.

A B	NOT A	A OR B	A AND B	A NOR B	A NAND B	A XOR B	A NXOR B
00	1	0	0	1	1	0	1
01	1	1	0	0	1	1	0
10	0	1	0	0	1	1	0
11	0	1	1	0	0	0	1

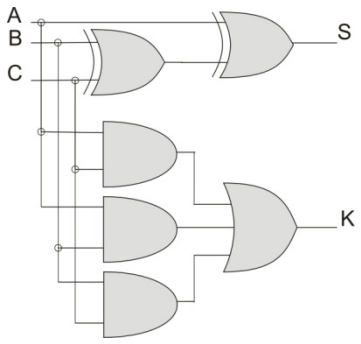
Taula 3.2. Taules de veritat de les portes

3.2 Components combinacionals

A nivell de sistema digital cadascuna de les portes lògiques correspon a un circuit electrònic implementat avui en dia amb transistors. Per unió de portes es construeixen els circuits digitals. Els més simples s'anomenen circuits combinacionals. La porta lògica n'és el component base.

Per tant, un *circuit combinacional* és un circuit digital implementat amb portes lògiques llur resultat és atemporal. Amb això s'entén que, com a tot circuit físic, un circuit combinacional respondrà davant canvis en les seves entrades de manera que, tot canvi en una entrada provoca automàticament un conjunt d'events que poden desembocar en un canvi en el valor de la sortida.

La figura 3.1 mostra un circuit combinacional simple que consta de tres entrades anomenades A, B i C i dues sortides, S i K. Es pot observar que és la connexió entre portes que les lliga i en configura un circuit. Una característica fonamental de tot circuit digital és que una sortida de porta pot anar a diferents entrades però que mai dues sortides es poden connectar entre sí[2].



a)

A B C	S K
0 0 0	0 0
0 0 1	1 0
0 1 0	1 0
0 1 1	0 1
1 0 0	1 0
1 0 1	0 1
1 1 0	0 1
1 1 1	1 1

b)

Figura 3.1. Circuit combinacional sumador complert

El circuit de la figura 3.1a correspon a un circuit digital. És fàcil trobar el seu funcionament. Es pot trobar a partir de:

- La taula de veritat del circuit. Per a construir la taula de veritat del circuit només cal assignar valors a les entrades del circuit i veure quina sortida s'obté aplicant les equacions de funcionament de cada porta. La figura 3.1b mostra la taula de veritat del circuit. Es pot observar que cada sortida dóna lloc a una funció. Cada funció rep el nom de *funció booleana*.

[2] L'etapa de sortida d'una porta lògica està formada per un parell de transistors actuant en mode corrent que exigeix una càrrega d'alta impedància per a poder funcionar correctament, és a dir, que la sortida s'ha de connectar a l'entrada d'una o més portes.
L'excepció a la regla la produeixen uns pocs components que, per construcció de la seva etapa de sortida, poden connectar entre sí les seves sortides. Es tracta de circuits amb etapa de sortida amb col·lector obert o drenador obert o elements *tri-state* o d'alta impedància.

- Analíticament, emprant les equacions de les portes lògiques. Substituint cada porta per la seva equació es troba fàcilment les *equacions booleanes* del circuit:

$$S = A \oplus B \oplus C$$
$$K = A{\cdot}C + A{\cdot}B + B{\cdot}C$$

En la taula de veritat cada sortida es posa en una columna, indicant que les dues sortides són independents entre sí. Per a trobar cadascuna de les funcions booleanes només cal sumar els 1's de la funció. I cada 1 es compleix quan es compleixen els valors de les entrades en aquella fila. Per tant es té:

$$S = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot \overline{C} + A \cdot \overline{B} \cdot \overline{C} + A \cdot B \cdot C$$
$$K = \overline{A} \cdot B \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot \overline{C} + A \cdot B \cdot C$$

Tot i que semblin funcions booleanes diferents a les trobades analíticament, aplicant les propietats de l'àlgebra de Boole és fàcil demostrar que són equivalents. Per tant, ambdós conjunts d'equacions ens representen al mateix circuit, encara que dissenyant el circuit a partir del darrer conjunt d'equacions s'obtingui un 'dibuix' diferent. Funcionalment representen al mateix circuit, un sumador complet o *full-adder*.

Si s'analitza amb més profunditat la funcionalitat d'aquest circuit simple es veu que el sumador complet realitza la suma aritmètica de tres bits (es comprova fent les sumes de la taula de veritat de la figura 3.1). I el circuit té dues sortides per què la suma aritmètica de 3 bits pot arribar a valdre 3 (quan els tres bits valen 1) en decimal, que és 11 en binari (figura 3.2a). I per representar dos bits es necessiten dues sortides o funcions booleanes.

Figura 3.2. a) Exemple de suma emprant sumador complets.
b) Suma aritmètica de quatre dígits.

D'aquí els noms que s'han donat a les entrades i les sortides. A i B corresponen als dos bits que es sumen. C és una entrada que agafa el carreteig del dígit anterior quan es produeix. S i K són les sortides, i corresponen a la sortida suma i carreteig que es propaga, respectivament.

Per unió de sumadors complets es pot fer un sumador de qualsevol nombre de bits. La figura 3.2b mostra la suma aritmètica de dos nombres de 4 bits. Es pot observar que es fa per repetició de l'estructura aritmètica de la figura 3.2a. Quan es produeix carreteig es propaga a l'etapa següent per a ser sumada. Cada etapa té tres bits d'entrada i 2 de sortida.

Quan es vol implementar la suma amb circuits digitals es segueix el mateix procediment: per repetició de sumadors complets es pot implementar un sumador de qualsevol nombre d'etapes. Cada etapa és responsable del seu bit suma i del carreteig que es propaga a l'etapa següent. Com a circuit digital es coneix com a sumador *ripple carr*y i és un mòdul estàndard que es troba implementat en diferents circuits integrats. Si el circuit digital de la figura 3.1 s'empaqueta en la forma de la figura 3.3a, la construcció del sumador *ripple carry* de 4 etapes ve donada per la figura 3.3b. Si es mira el sumador *ripple carry* com a mòdul aleshores es pot empaquetat en la forma de la figura 3.3c.

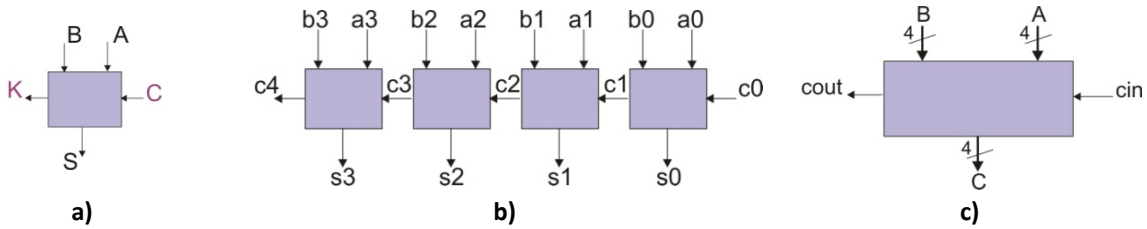


Figura 3.3. a) Sumador complet com a entitat. b) Sumador *ripple carry* de 4 etapes. c) Sumador *ripple carry* de 4 etapes com a entitat.

La figura 3.3c mostra algunes propietats que es segueixen quan es treballa amb mòduls. Cin i cout són simples senyals. El fil que els representa és prim. A, B i C són busos (a nivell matemàtic se'n podria dir vector). En aquest cas representen la unió de 4 fils en cada cas, d'aquí la línia transversal amb el 4. Es pot observar que, de manera molt simple, identifica al circuit de la figura 3.3b.

3.3 Mòduls combinacionals

El sumador és un demostrador perfecte per veure la metodologia seguida en la construcció de sistemes digitals. S'ha vist com a partir de les portes es construeixen circuits digitals més grans. El sumador, vist com a entitat, és un mòdul combinacional emprat per construir circuits digitals més complexes. En aquest apartat s'introdueix el funcionament dels mòduls combinacionals més usats en el disseny de processadors.

3.3.1 El multiplexor.

El multiplexor és un mòdul que selecciona un senyal d'entrada cap a la sortida a través de senyals de control. La figura 3.4a mostra el mòdul més simple, el d'un senyal de control. Amb el senyal de control *c* es pot seleccionar una de dues senyals d'entrada *x1* o *x0* per passar cap a la sortida *z*. La figura 3.4b mostra el circuit combinacional intern al multiplexor d'una entrada de control. Quan *c* val 0 *z* agafa el valor de *x0*. Quan val 1 agafa el valor de *x1*. L'equació booleana que implementa el circuit és

$$z = x1 \cdot c + x0 \cdot \bar{c}$$

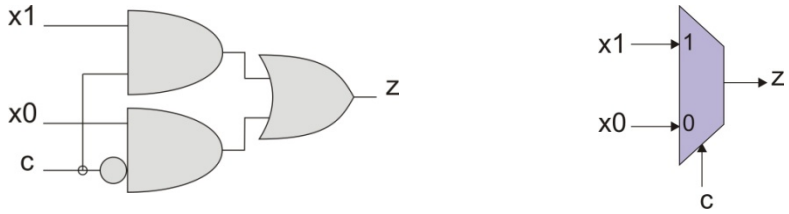


Figura 3.4. a) Multiplexor d'una entrada de control. b) Multiplexor de dues entrades de control. c) Multiplexor de dues entrades de control construït amb multiplexors d'una entrada de control.

El multiplexor s'utilitza fonamentalment en el control de senyals d'accés a un bus. En el computador, per exemple, és normal emprar els busos com a 'autopistes' de transport de dades entre diferents components del processador. Amb el multiplexor es pot controlar el component que té accés al bus per a enviar la seva dada.

La construcció de multiplexors és escalable. La figura 3.5 mostra el multiplexor de dues entrades de control. Amb dos senyals de control es pot seleccionar a un de 4 possibles senyals d'entrada.

L'equació booleana que representa el multiplexor de dos senyals de control és, per tant, una extensió de la del multiplexor d'un senyal de control.

$$z = x3 \cdot c1 \cdot c0 + x2 \cdot c1 \cdot \overline{c0} + x1 \cdot \overline{c1} \cdot c0 + x0 \cdot \overline{c1} \cdot \overline{c0}$$

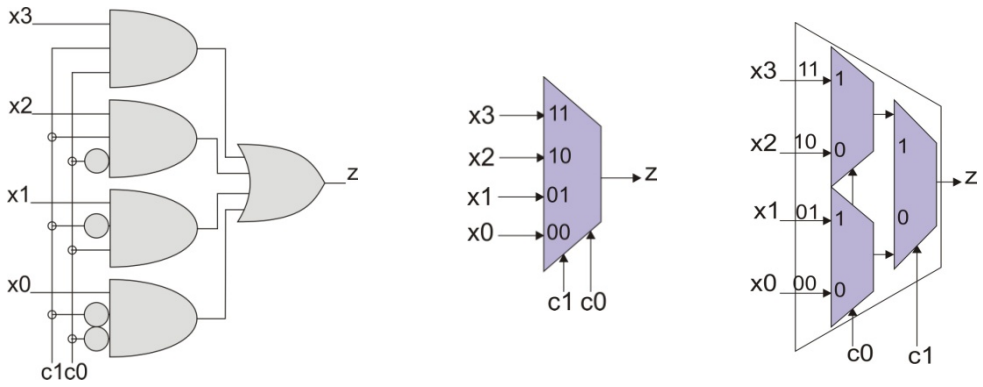


Figura 3.5. Multiplexor de dues entrades de control.

Seguint amb el mateix raonament, amb tres senyals de control es pot seleccionar a una de 8 entrades possibles. En genèric, amb n senyals de control es pot seleccionar a una de 2^n senyals d'entrada (figura 3.6).

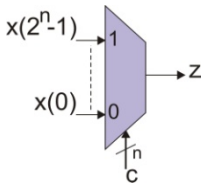


Figura 3.6. Mòdul multiplexor genèric.

Donat que la construcció de multiplexors grans requereix de molta lògica, gràcies a l'escalabilitat del multiplexor, es fa construint arbres multiplexors amb multiplexors d'ordre menor. La figura 3.5c mostra la construcció del multiplexor de dues entrades de control emprant multiplexors d'una entrada de control.

3.3.2 El descodificador.

El descodificador és un circuit combinacional de n entrades i 2^n sortides de manera que totes les sortides valen zero excepte aquella que té per índex l'enter que representa l'entrada. La figura 3.7 mostra el descodificador de 2-a-4, és a dir, dos senyals d'entrada i 4 de sortida. Sovint s'hi posa un senyal de capacitació o *enable* que inhabilita tot el circuit si no està activat.

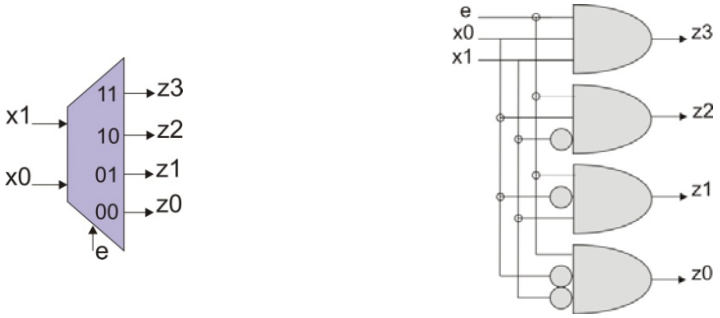


Figura 3.7. Descodificador 2-a-4 amb senyal d'*enable*.

Les equacions booleanes del descodificador són:

$$z3 = e \cdot x1 \cdot x0$$
$$z2 = e \cdot x1 \cdot \overline{x0}$$
$$z1 = e \cdot \overline{x1} \cdot x0$$
$$z0 = e \cdot \overline{x1} \cdot \overline{x0}$$

El descodificador és un circuit molt emprat en els sistemes digitals. S'utilitza fonamentalment en la descodificació de senyals, és a dir, s'utilitza com a selector d'un conjunt de dades: permet seleccionar un d'entre 2^n elements. Entre altres funcionalitats forma el descodificador d'adreces en memòries.

La construcció de descodificadors grans implica l'ús de moltes portes lògiques. Per això els descodificadors grans es fa amb arbres descodificadors, de forma similar als arbres multiplexors introduïts en l'apartat 3.3.2.

3.3.3 La unitat aritmètico-lògica (UAL o ALU)

La UAL és el component digital més important del processador encarregat del procés de dades. Tal com el seu nom indica és una unitat capaç de realitzat operacions aritmètiques i lògiques.

La UAL pot tenir moltes funcionalitats diferents depenent de la orientador del processador. La figura 3.8 mostra un esquema genèric d'una UAL. De forma general consta dels següents ports:

- Els operands, simbolitzats pels ports A i B.
- Els senyals de carreteig d'entrada i de sortida. Apart d'actuar com a carreteig es poden emprar també en operacions d'entrada/sortida.
- El senyal de control d'operació *opcode*. Selecciona l'operació que realitza la UAL.
- Senyals d'estat de l'operació. Informa sobre el resultat de l'operació. Solen ser els bits de:
    - Zero. Indica si el resultat de l'operació ha estat zero
    - Carreteig. Informa de quan es produeix carreteig.
    - Saturació o *overflow*. Indica si el resultat de l'operació ha saturat la capacitat del sistema.
    - Signe. El bit de signe indica si es resultat és positiu o negatiu quan s'opera amb nombres amb signe.

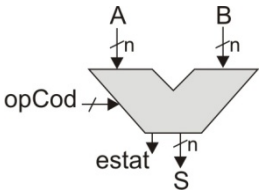


Figura 3.8. Unitat aritmètica lògica.

Entre les operacions que realitza una UAL hi ha operacions aritmètiques (suma i resta), lògiques (and, or, not, or-exclusiva) i operacions de desplaçament (a dreta, a esquerra i rotacions).

3.4 El diagrama de temps

El diagrama de temps mostra en un gràfic temporal l'evolució d'un circuit digital. El diagrama de temps es construeix posant en la gràfica totes les entrades implicades i les sortides i senyals intermedis que es desitja analitzar.

La figura 3.9 mostra el diagrama de temps del sumador complet (apartat 3.2). En el diagrama s'hi han de posar les combinacions possibles de les entrades. Aleshores, analitzant el circuit (s'ha fet sobre la figura 3.1a) es calculen les sortides, S i K en aquest cas. Per exemple, en el cas d'entrades (A, B, C) = (1, 1, 0) s'observa que les sortides agafen el valor (S, K) = (0, 1).

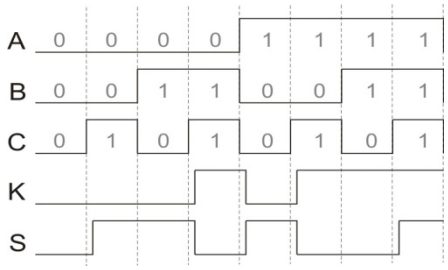


Figura 3.9. Diagrama de temps del circuit sumador complet.

En aquest exemple s'han fet totes les combinacions possibles de les entrades. A més, en aquest cas s'ha posat un petit retard en el càlcul de les sortides indicant el petit retard de propagació que tenen totes les portes lògiques.

El diagrama de temps és una eina important en l'anàlisi de circuits seqüencials. També s'emprarà en capítols posteriors per analitzar la temporització dels senyals interns del processador.

3.5 Components seqüencials

Com s'ha introduït en l'apartat 3.3, els circuits combinacionals són sistemes que evolucionen sense recordar l'evolució que han sofert. Tan bon punt canvia l'entrada d'un circuit combinacional se succeeixen un conjunt d'events que poden conduir a un canvi en la retarda.

Un circuit seqüencial és un sistema digital que la seva evolució en el temps depèn dels estats pels que prèviament ha passat. El sistema seqüencial recorda, fins a un moment determinat, pels estats que prèviament ha passat.

Els circuits seqüencials síncrons funcionen amb un senyal de rellotge. Un senyal de rellotge és un senyal periòdic elemental que sincronitza l'evolució de tot sistema digital síncron. L'ús de senyals de rellotge és fonamental per sincronitzar l'evolució de tot processador.

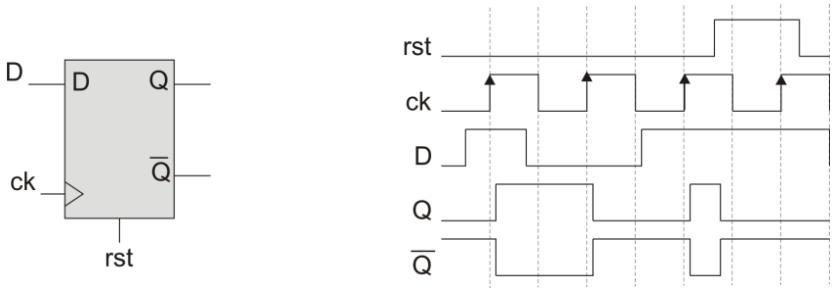
Els components elementals dels circuits seqüencial són els biestables. El *biestable* és un punt de memòria que pot guardar un valor (0-lògic o 1-lògic). Quan el biestable va governat per flanc de rellotge se'l sol anomenar flip-flop. Així, el *flip-flop* és un punt de memòria que, controlat per flanc de rellotge, guarda temporalment un bit d'informació. L'*estat* representa el valor que guarda el flip-flop. L'evolució d'un flip-flop depèn de l'entrada i l'estat en el que es troba (Q). Aquest fet fa que es digui que el flip-flop evoluciona segons la seva història.

Els flip-flops emprats són el D, RS, JK i T. Per simplicitat, en el que segueix només s'utilitzarà el flip-flop D. La figura 3.10 mostra el funcionament del flip-flop D amb el senyal asíncron *reset*. Consta de tres entrades anomenades *rst*, *ck* i *D*. D és l'entrada de dades del circuit; ck és el senyal de rellotge; i rst és un senyal asíncron que inicialitza el flip-flop. La sortida del flip-flop és Q. La sortida $\overline{Q}$ és la inversa de Q. El funcionament del flip-flop és simple:

- En funcionament normal tot canvi en l'entrada D és guardat en el flip-flop en arribar el flanc de pujada de rellotge. El valor queda emmagatzemat fins que arriba el proper flanc de rellotge.

- En el cas del flip-flop D l'estat actual no intervé en el càlcul del nou estat. En aquest sentit es diu que el flip-flop D és el flip-flop transparència, en el sentit que tota entrada és propagada a la sortida amb un cicle de rellotge de retard.
- Quan s'ha d'inicialitzar l'estat del flip-flop es pot emprar el senyal de reset. En posar-lo a 1-lògic el flip-flop D es posa a 0, independentment del que faci el senyal de rellotge.

La figura 3.10b mostra en un diagrama de temps el funcionament del flip-flop D. Cal observar que tot canvi es produeix en el flanc de pujada del rellotge, que es compleixen les característiques acabades d'enumerar i que el senyal de reset reinicialitza el flip-flop.



a) **b)**

Figura 3.10. Flip-flop D.

El funcionament del flip-flop D es pot posar en forma de taula d'estat. La taula 3.3 mostra la taula d'estat del flip-flop D. Les entrades de la taula són l'entrada del flip-flop D i l'estat en el que es troba (Q). Les sortides són el proper estat Q^+ i el proper estat negat $\overline{Q^+}$. Queda explícit que el funcionament del flip-flop D és independent de l'estat en què es troba.

D Q	Q^+ $\overline{Q^+}$.
0 0	0 1
0 1	0 1
1 0	1 0
1 1	1 0

Taula 3.3. Taula d'estat del flip-flop D

3.6 Mòduls seqüencials

Quant s'ha tractat la construcció de circuits combinacionals en l'apartat 3.3 s'ha vist que l'element base de construcció era la porta lògica. Ara, quan s'entra es veurà que els elements indispensables en la construcció de circuits seqüencials són els flip-flops. La figura 3.11 és un esquema genèric de construcció de circuits seqüencials.

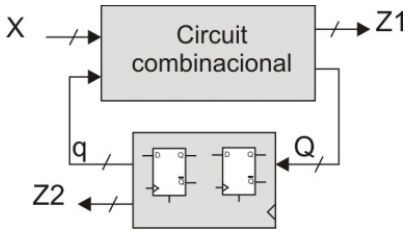


Figura 3.11. Esquema genèric d'un circuit seqüencial

D'acord amb la figura, un circuit seqüencial es pot considerar constituït per dos blocs. El bloc combinacional i el bloc seqüencial. El bloc combinacional es construeix amb portes lògiques i és l'encarregat de calcular el proper estat del circuit. El bloc seqüencial conté només flip-flops i és l'encarregat de guardar l'estat en el que es troba el circuit. El càlcul del proper estat (Q) depèn de l'estat actual en el que es troba el circuit (q) i de les entrades (X). En quant a les sortides, aquestes poden dependre directament de les entrades i de l'estat (cas de la sortida Z1) o ser només dependents de l'estat (la sortida Z2). En el primer cas es parla de *màquina de Mealy* mentre que en el segon es diu que és una *màquina de Moore*.

Aquest apartat analitza els principals mòduls seqüencials que utilitza un processador. Tots compleixen les característiques generals dels circuits seqüencials. El més important és, però, entendre el seu funcionament, ja que formen la base de funcionament de tot processador. Per exemple, quan es parli de registre d'instruccions caldrà pensar que el mòdul seqüencial registre; quan s'introdueixi el comptador de programes la seva construcció es basa en un registre o en un comptador; els ports d'entrada/sortida han de ser registrats; etc. Cal comprendre bé el funcionament d'aquests components.

3.6.1 El registre.

El registre és un circuit seqüencial composat de flip-flops que permet emmagatzemar la informació d'una dada. Un registre conté tants flip-flops com bits necessiti emmagatzemar. Per exemple, si es vol guardar una dada d'un byte, el registre tindrà 8 flip-flops.

La figura 3.12 mostra l'entitat que representa a un registre. Consta dels següents senyals:

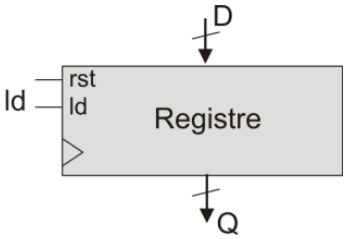


Figura 3.12. Registre.

- Els senyals de reset i rellotge, comuns als circuits seqüencials síncrons amb senyal d'inicialització.
- El senyal de càrrega, *ld*.
- L'entrada de dades *D*.
- La sortida de dades *Q*.

El funcionament del registre és molt simple:

- Per defecte el registre manté la dada. Això és, l'estat proper continua essent l'estat actual, expressat com Q = q.
- Si s'activa el senyal de càrrega *ld*, aleshores la dada *D* es carrega en el registre: Q = D.
- Si s'activa el senyal de reset, aleshores el registre s'inicialitza a 0.

La figura 3.13 mostra el circuit corresponent a un registre de 4 bits. Es pot observar que en aquest cas cada flip-flop és autònom, i que només depèn de les entrades i estat en el que es troba: si el senyal *ld* del multiplexor està a 0-lògic es manté l'estat (la sortida *q* es realimenta cap a l'entrada del flip-flop D); si *ld* passa a 1-lògic, aleshores l'entrada *D* es carrega en el registre.

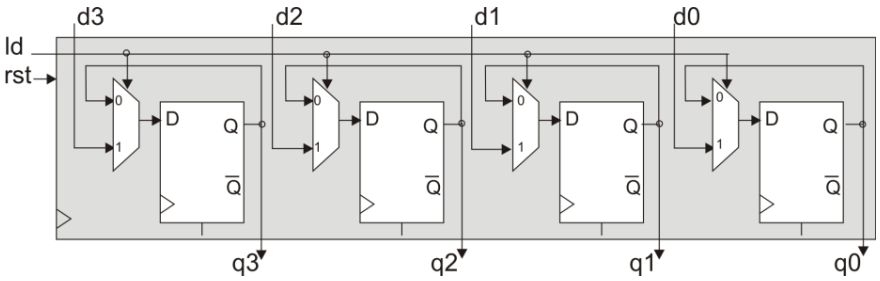


Figura 3.13. Construcció del registre.

La figura 3.14 mostra en un diagrama de temps el funcionament del registre. En el primer flanc de rellotge es carrega la dada d'entrada D(3..0) = (0011) en el registre. Es torna a canviar la dada del registre en el tercer flanc del rellotge. Finalment, una activació del senyal de reset inicialitza el registre.

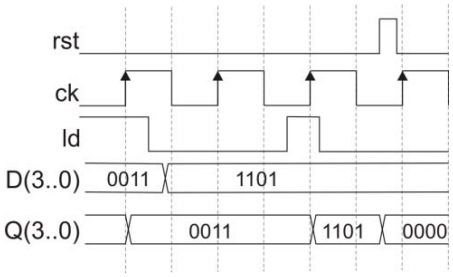


Figura 3.14. Diagrama de temps del registre.

3.6.2 El comptador binari.

El comptador és un circuit seqüencial que presenta una sortida incremental de l'estat en el que es troba. El comptador clàssic és el comptador binari ascendent, tot i que existeixen altres comptadors pel codi que utilitzen, com el codi Gray, etc. La figura 3.15 mostra les taules d'estats d'aquests comptadors en el cas de 3 bits. La taula d'estat posa el proper estat en funció de l'estat actual. En tots els casos, un cop s'arriba al darrer estat el comptador torna a començar per l'estat inicial.

La figura 3.16a mostra l'entitat d'un comptador amb senyal de càrrega paral·lela i reset asíncron. El funcionament, d'acord amb les entrades del mòdul, és el següent:

- El senyal de reset *rst* és el responsable d'inicialitzar el comptador.
- El comptador també té un senyal de càrrega paral·lela *ld* que permet carrega la dada externa *D* al comptador.

Codi binari	
q2 q1 q0	Q2 Q1 Q0
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	1 0 1
1 0 1	1 1 0
1 1 0	1 1 1
1 1 1	0 0 0

Codi Gray	
q2 q1 q0	Q2 Q1 Q0
0 0 0	0 0 1
0 0 1	0 1 1
0 1 1	0 1 0
0 1 0	1 1 0
1 1 0	1 1 1
1 1 1	1 0 1
1 0 1	1 0 0
1 0 0	0 0 0

Codi Johnson	
q2 q1 q0	Q2 Q1 Q0
0 0 0	0 0 1
0 0 1	0 1 1
0 1 1	1 1 1
1 1 1	1 1 0
1 1 0	1 0 0
1 0 0	0 0 0

Figura 3.15. Comptadors segons el codi.

- Finalment, si el senyal de capacitació *e* està activat, aleshores el comptador compta a cada flanc de pujada o baixada de rellotge.
- El comptador té dues sortides: l'estat *Q* i el senyal de fi de compte o *terminal counter TC*. Aquest senyal es posa a 1 quan el comptador arriba al darrer estat.

La figura 3.16b mostra en un diagrama de temps el funcionament del comptador.

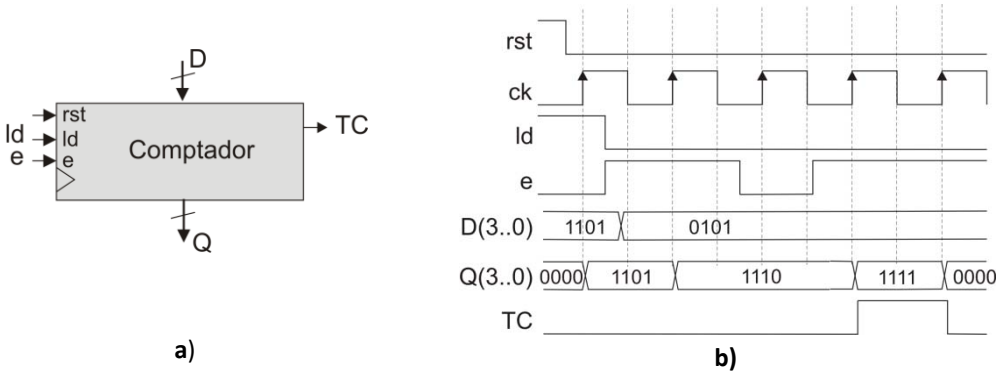


Figura 3.16. a) Entitat del mòdul comptador. b) Diagrama de temps.

3.6 La memòria.

La memòria és el dispositiu encarregat de guardar dades en el computador. És un dispositiu d'entrada/sortida que permet retenir informació i donar-la quan se li demana. En el seu concepte més bàsic la memòria és un dispositiu semiconductor basat en transistors. Vist com a circuit integrat es parla de memòries ROM i RAM, amb moltes variants. Posant múltiples circuits integrats en un mateix substrat es tenen els mòduls de memòria emprats en el computador. L'acrònim ROM prové de *Read Only Memory*, o memòria de només lectura. Té la capacitat de proporcionar informació que té guardada. RAM significa *Random Access Memory*, i és una memòria que permet escriure i llegir informació.

Independentment del tipus de memòria, una memòria és un circuit digital que guarda informació i l'estructura en forma de matriu. La matriu es distribueix en un conjunt de paraules que poden ser llegides. Una paraula és un conjunt de bits, normalment múltiple del byte. La selecció de la paraula a llegir es realitza per mitjà del descodificador d'adreces.

Totes aquestes característiques es troben resumides en la memòria de la figura 3.17, que presenta les següents característiques:

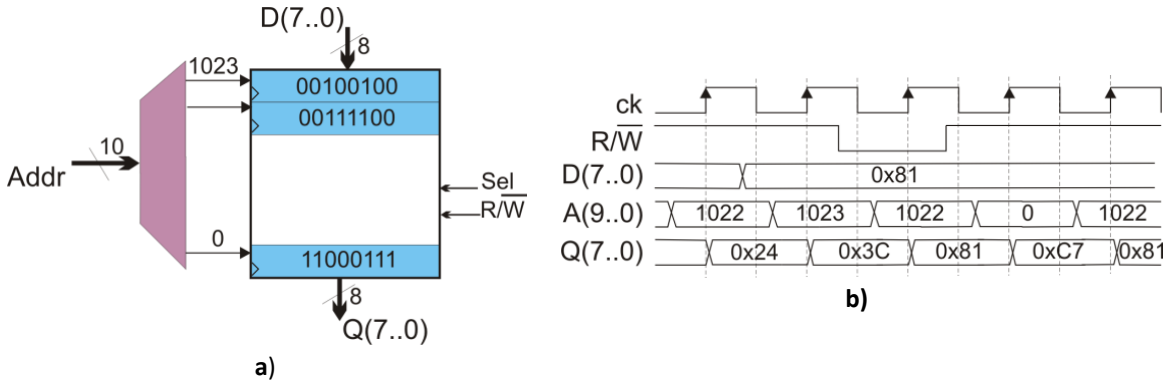


Figura 3.17. a) Memòria RAM. b) Diagrama de temps.

- En la figura s'hi observa clarament la matriu i el descodificador d'adreces.
- La matriu està formada per 1024 paraules. Cada paraula és d'un byte (8 bits). Cada paraula es pot entendre com a un registre. Normalment, tota operació d'escriptura va, doncs, sincronitzada amb un senyal de rellotge. La lectura pot no ser sincronitzada.
- En tot moment només una paraula pot ser accedida. El descodificador d'adreces és el mòdul encarregat de controlar-ho (recordar que el descodificador només té una sortida activada mentre que totes les altres estan desactivades). Donat que s'han d'adreçar 1024 posicions, el descodificador necessita una entrada Addr(9..0) de 10 bits. Aquesta entrada Addr(9..0) s'anomena *bus d'adreces*.
- Es mostra un senyal de lectura/escriptura R/$\overline{W}$. Quan es realitza una lectura (R/$\overline{W}$ = 1) la paraula adreçada surt pel bus de dades Q(7..0). Quan es realitza una escriptura (R/$\overline{W}$ = 0) la dada D(7..0) es carrega en la paraula adreçada.
- El senyal de selecció *Sel* permet seleccionar el mòdul de memòria amb què es treballa.

La figura 3.17b mostra un diagrama de temps resumint el funcionament de la memòria. El diagrama de temps exemplifica el funcionament de la memòria. El diagrama mostra la lectura (sincronitzada) de diferents posicions de memòria i, enmig en el tercer cicle) executa una escriptura de la posició 1022.

Les característiques d'una memòria són:

- L'*espai adreçable*. És el nombre de posicions de memòria que es poden adreçar, i depèn del nombre de bits que té l'adreça.
- La *capacitat de la memòria*. És el número de posicions de memòria que té.
- En un computador el *mapa de memòria* és l'espai adreçable total pel computador. És a dir, si un computador empra un bus d'adreces de 10 bits, el seu mapa de memòria té una capacitat de 2^10 paraules.

3.7 Resum del capítol

El capítol introdueix el lector en els fonaments dels sistemes digitals des d'un punt de vista molt pràctic de manera que es pugui aprendre el funcionament dels components digitals essencials sense haver d'aprofundir en la base matemàtica de l'àlgebra de Boole.

Del capítol és important:

- Diferenciar el concepte de circuit combinacional del de seqüencial.
- Haver entès el funcionament de tots els mòduls, combinacionals i seqüencials introduïts.
- Entendre i saber realitzar diagrames de temps, ja que permeten visualitzar ràpidament el funcionament dels sistemes digitals. En propers capítols es fan servir per veure el sincronisme que s'estableix en els diferents components del processador en el cicle d'instrucció.
- Finalment s'introdueixen les memòries com a circuit integrat, com a element que en propers capítols formarà la base de la memòria principal del computador.

3.8 Exercicis resolts

Exercici 1. El 7-segments.

El 7-segments (figura 3.18) és un dispositiu optoelectrònic que consta de 8 leds, 7 d'ells disposats de manera que permeten representar qualsevol número hexadecimal i un 8è per a representar el punt decimal. Per exemple, en la figura 3.18 el 7-segments representa el número 2 encenent els leds a, b, d, e i g. Per a poder representar el número 2, codificat en hexadecimal, en el set-segments fa falta un circuit que converteixi el número 2 en l'encesa dels leds corresponents. De forma general, qualsevol número que es vulgui representar necessita de la conversió del número hexadecimal a l'encesa dels pertinents leds en el 7-segments. A aquest circuit se l'anomena codificador.

Es demana i) donar la taula de conversió de hexadecimal a 7-segments i ii) trobar el circuit combinacional que l'implementa.

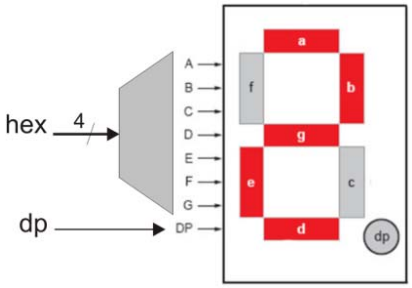


Figura 3.18

Solució.

El circuit que realitza la conversió d'un número hexadecimal en el 7 segments és un codificador: transforma una entrada en les sortides necessàries que encenen els leds de forma adequada.

La forma de construcció de tot circuit combinacional s'ha introduït en l'apartat 3.2. Aplicant la mateixa metodologia s'observa que el problema es resol en dos passos:

i) Trobar la taula de veritat.
ii) Convertir la taula en el corresponent circuit combinacional.

Donat que el codi hexadecimal té 16 elements i el 7-segments consta de 7 leds, la taula de veritat ve donada per la taula 3.4.

Codi	h3 h2 h1 h0	G F E D C B A
0	0000	0111111
1	0001	0000110
2	0010	1011011
3	0011	1001111
4	0100	1100110
5	0101	1101101
6	0110	1111101
7	0111	0000111

Codi	h3 h2 h1 h0	G F E D C B A
8	1000	1111111
9	1001	1100111
a	1010	1110111
b	1011	1111100
c	1100	0111001
d	1101	1011110
e	1110	1111001
f	1111	1110001

Tala 3.4. Taula de veritat de la codificació hexadecimal a 7-segments.

Per a trobar el circuit combinacional només cal cercar els uns de la taula. Les equacions booleanes que en resulten són la suma dels següents termes:

Led G = 2+3+4+5+6+8+9+a+b+d+e+f
Led F = 0+4+5+6+8+9+a+b+c+e+f
Led E = 0+2+6+8+a+b+c+d+e+f
Led D = 0+2+3+5+6+8+b+c+d+e
Led C = 0+1+3+4+5+6+7+8+9+a+b+d
Led B = 0+1+2+3+4+7+8+9+a+d
Led A = 0+2+3+5+6+7+8+9+a+c+e+f

Per transformar a circuit combinacional només cal traduir cada equació anterior en una suma de productes.

Exercici 2. Analitzant un circuit seqüencial.

Amb l'anàlisi de circuit seqüencials es descobreix la funcionalitat del circuit. Com a exemple, en aquest apartat es mostren els passos que s'han de seguir per a realitzar l'anàlisi. Es realitzarà emprant el circuit simple de la figura 3-19.

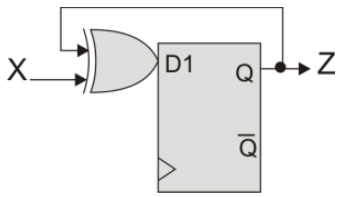


Figura 3.19. Circuit seqüencial.

Solució.

Per analitzar un circuit seqüencial el primer que cal és identificar els senyals de què es composa. Observant a la figura 3.19 (on s'hi ha posat tots els noms essencials que calen per l'anàlisi) s'observa que es composa de:

- Senyals d'entrada. En aquest cas és X.
- Senyals de sortida. És el senyal Z
- Els senyals d'entrada als flip-flops. Com que només n'hi ha un, és D0.
- La sortida del flip-flop representa l'estat actual q0.
- Finalment, el que no mostra el circuit són els propers estats. El proper estat representa l'estat que esdevindrà actual un cop s'apliqui el senyal de rellotge. Com que només hi ha un flip-flop, el proper estat serà Q0, i donat que es tracta d'un flip-flop D, es compleix que Q0 = D0.

Amb aquestes condicions de partida, l'anàlisi passa per les següents etapes:

i) Es cerquen les entrades dels flip-flops i es posen en funció de l'estat actual i les entrades. En aquest circuit es veu que:

$D0 = q0 \oplus X$.

ii) Es busquen els propers estats. Com que es treballa amb flip-flops D

$Q0 = D0 = q0 \oplus X$.

iii) Es crea la taula d'estats. La taula d'estats calcula els propers estats (que és la incògnita en l'anàlisi) en funció dels estats actuals i les entrades (taula 3.5).

Entrades i estats actuals	Propers estats	Sortides
X q0	**Q0**	**Z**
0 0	0	0
0 1	1	1
1 0	1	1
1 1	0	0

Taula 3.5. Taula d'estats.

S'observa que en aquest exemple la sortida coincideix amb el proper estat, fet que no té per què succeir en tots els circuits.

iv) Es posa els estats en format d'alt nivell. És a dir, es dóna nom als estats codificats amb 0's i 1's.

En aquest cas, i de la taula es dedueix que la sortida val 1 sempre que ha entrar un nombre senar d'1s. Així s'anomenen els estats com a *parell* i *senar*, depenent del nombre d'uns que han rebut. La taula 3.6 mostra el resultat. Cal observar que la taula s'ha posat en un format més adequat, on a l'esquerra hi ha els estats actuals i enmig els propers estats.

Entrades i estats actuals	Propers estats		Sortides
	X		
q0	**0**	**1**	**Z**
Parell	Parell	Senar	0
Senar	Senar	Parell	1
	Q0		

Taula 3.6. Taula d'estats d'alt nivell.

v) I finalment es posa la taula a format de graf dirigit (figura 3.20), en el que els cercles representen els estats actuals i les fletxes a transicions. El graf representa el mateix que la taula, i es llegeix de la següent manera. Suposant que ens trobem en l'estat parell, *si l'entrada val 1, el circuit passa a l'estat Senar amb sortida 1; si del mateix estat entre un 0, ens quedem en el mateix estat i la sortida val 0*. De forma semblant es llegeix per a cada estat restant.

Analitzant el graf s'observa clarament que el circuit és un detector de paritat senar. Cada cop que ha arribat un nombre senar d'1s, la sortida val 1.

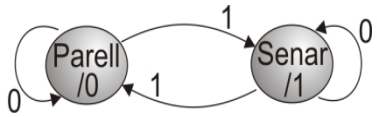


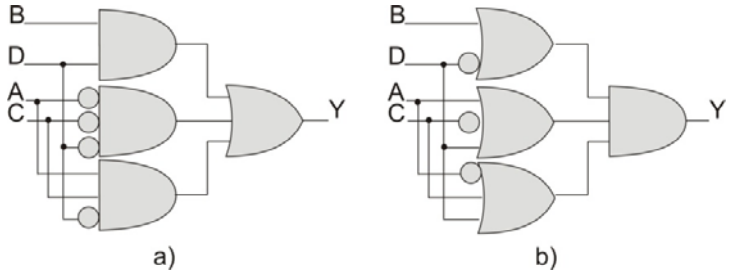
Figura 3.20. Graf d'estats de l'exercici.

3.9 Exercicis.

1. Donats els circuits combinacionals següents es demana:

- Trobar la seva funció booleana.
- Trobar la taula de veritat.
- Donar un diagrama de temps.

Nota: Observar que els dos circuits implementen la mateixa funció booleana. Són circuits duals.

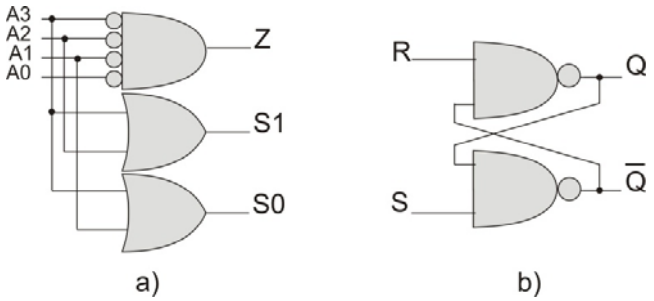


Exercici 1. Circuits combinacionals.

2. Trobar la funció booleana dels circuits següents.

Notes: i) El circuit 3.19a correspon a un codificador.

ii) El circuit de la figura 3.19b correspon a un punt de memòria. És el circuit base de construcció dels flip-flops. En el seu anàlisi realitzar una taula de veritat posant les sortides Q i Q en funció de les entrades S i R. Observar que en un cas cal deixar la sortida en funció de les sortides anteriors: el circuit es comporta com a guarda d'estat. Observar, per tant, que el circuit es pot inicialitzar a 0 i a 1 i que pot mantenir l'estat.



Exercici 2. a) Codificador. b) Biestable SR:

3. Donar el circuit combinacional que implementa un multiplexor de tres entrades de control.

4. Donar el circuit combinacional que implementa un descodificador de tres entrades de control

5. Implementar una unitat aritmètico-lògica que realitzi les operacions suma, and, or i not de nombres de 4 bits.

Nota: Seguir les següents pautes per a realitzar el circuit: i) Pensar en la següent implementació modular. Ii) Dissenyar un mòdul processador que tregui una sortida per a cadascuna de les operacions treballant amb nombres d'un bit. iii) Per iteració, construir la unitat capaç de treure aquestes operacions per nombres de 4 bits. Iv) Seguidament, amb un multiplexor a la sortida s'esculli l'operació desitjada.

6. El conjunt d'equacions d'estat següents representen a un circuit seqüencial. X és l'entrada del circuit i Z la sortida. Es demana trobar el circuit seqüencial que l'implementa.

3.9 Exercicis.

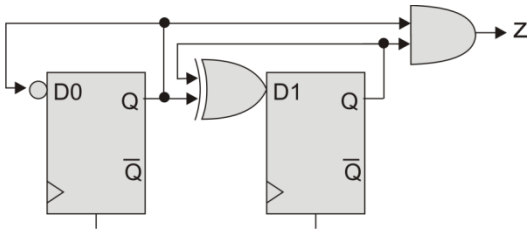
$Q3 = q2 \oplus q1$

$Q2 = q1 \cdot q3$

$Q1 = X + q3$

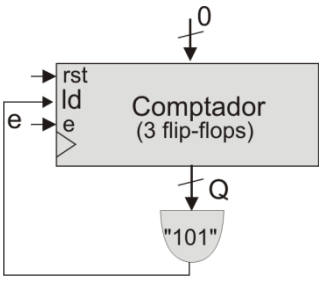
$Z = \overline{q3 \cdot q2 \cdot q1}$

7. Analitzar el circuit seqüencial de la següent figura. Correspon a algun circuit conegut?



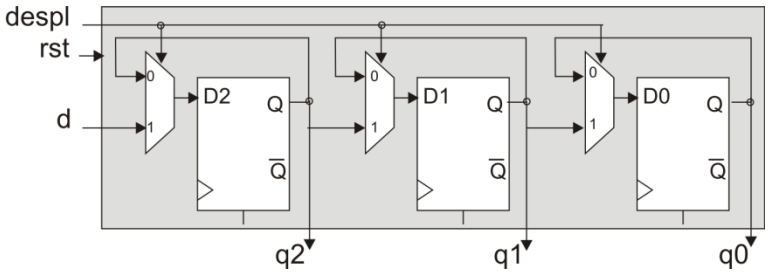
Exercici 7. Circuit seqüencial.

8. El circuit de la figura correspon a un comptador de 6 estats. Trobar la taula de veritat.



Exercici 8. Comptador de 6 estats.

9. El circuit de la següent correspon a un registre de desplaçament a la dreta. Trobar la seva taula de veritat i explicar el seu funcionament.



Exercici 9. Registre de desplaçament.

Capítol 4

EL SISTEMA ORDENADOR

El capítol 4 presenta una introducció a l'arquitectura del computador basada en l'Arquitectura von Neumann. Es veuran amb cert deteniment la CPU, el nucli constitutiu de l'ordinador, i el seu mecanisme de funcionament basat en l'Arquitectura von Neumann. S'introdueix l'estructura del computador i el seu mecanisme elemental de funcionament basat en el cicle d'instrucció.

En els apartats finals s'analitzen els paràmetres emprats en el còmput de prestacions en el computador i es presenta l'arquitectura Hardware com a variació de l'arquitectura von Neumann.

El capítol previ d'Introducció als sistemes digitals és important per entendre bé l'estructura i el funcionament del computador.

4.1 Estructura general del computador: màquina Von Neumann

John von Neumann va ser un matemàtic que feu nombroses aplicacions en diferents camps de la ciència. Entre les seves aportacions destaquen els treballs fets en el camp de la computació. L'any 1945, després d'entrar a treballar en la computadora ENIAC amb J.P. Eckert i J.W. Mauchly va establir l'arquitectura general del computador i el seu mecanisme bàsic de funcionament. La seva aportació va ser tal que molts dels computadors construïts fins avui en dia encara es basen en la seva aportació.

La màquina von Neumann descriu una arquitectura capaç d'executar un conjunt d'instruccions elementals (instruccions màquina) que s'han de guardar en la memòria principal i que poden ser llegides i executades. Consta de les cinc unitats funcionals unitat aritmètico-lògica, unitat de control, memòria, unitat d'entrada/sortida i bus de dades que comunica les altres unitats.

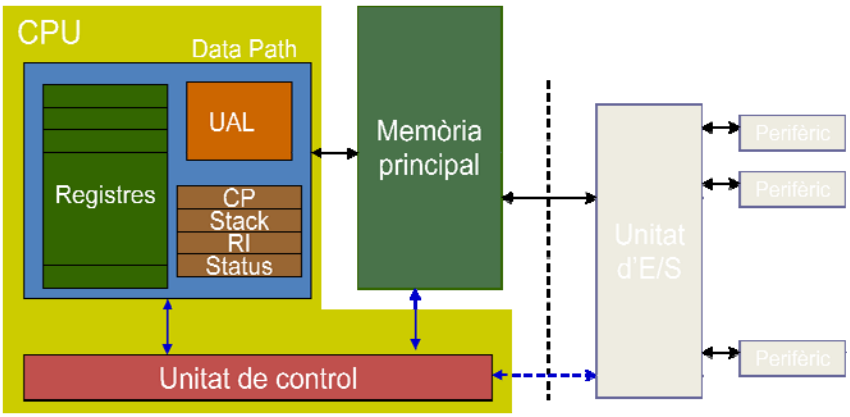


Figura 4.1. Arquitectura de la màquina von Neumann

La memòria principal està constituïda per un conjunt de cel·les idèntiques (nombre fix de bits) organitzades de forma consecutiva i amb adreça específica. Les instruccions i dades s'obtenen de la memòria a través del procés de lectura i s'hi guarden amb el procés d'escriptura. L'arquitectura von Neumann es caracteritza per compartir en el mateix espai de memòria programes i dades

La UAL és la unitat que permet realitzar operacions concretes sobre dades. Les dades (operands) provenen de la memòria principal o de registres que guarden dades de forma temporal

La unitat de control és l'encarregada de controlar (a través dels senyals de control) les accions que es realitzen en el processador: lectura de les instruccions de memòria, execució de les instruccions i control del seqüenciament.

El conjunt unitat aritmètico-lògica i unitat de control forma la Unitat Central de Procés (CPU). És el nucli del processador. La CPU també conté un conjunt de registres

La comunicació amb el computador es realitza a través dels perifèrics. Els perifèrics són dispositius físics que permeten la comunicació amb el computador. Segons el tipus de comunicació es té:

- Comunicació home-màquina, en les que les persones interaccionen amb el computador
- Comunicació màquina-màquina, en les que els computadors interaccionen entre ells
- Comunicació màquina-home, en les que el computador interacciona amb les persones

Cada perifèric imposa els seus propis requisits. Per exemple, la comunicació del computador amb l'home es sol fer mitjançant terminals i impressores, fet que obliga a fer operacions de traducció donat que els llenguatges emprats són totalment divergents (el computador treballa a nivell digital). Un aspecte important a tenir en compta en la comunicació de la CPU amb perifèrics és la velocitat de transferència de dades. Donat que normalment el funcionament del computador és molt més ràpid que no pas, per exemple, una impressora, sol tenir un impacte gran en el rendiment Per evitar la reducció de prestacions del computador s'empren dispositius específics d'entrada/sortida que alleugereixen al processador d'aquestes tasques

4.1.1 Funcionament intern de la CPU

John von Neumann va descriure una seqüència de cinc passos que seguia la CPU durant l'execució d'un programa que estava carregat en memòria:

1. Cerca i càrrega de la instrucció. En engegar, es va a la posició de memòria apuntada pel comptador de programes a cercar la instrucció que ha d'executar la CPU. La instrucció es carrega en el registre d'instruccions.
2. Increment del comptador de programes. El comptador de programes es prepara per a la propera instrucció que s'haurà d'executar.

3. Descodificació de la instrucció. La unitat de control descodifica la instrucció i prepara la resta de components per a executar la instrucció.
4. Execució de la instrucció. Les accions a executar depenen del tipus d'instrucció a executar. En una instrucció aritmètico-lògica intervindran els elements de la unitat de procés. En una instrucció de salt es carregarà en el comptador de programes una nova posició de memòria.
5. Es torna al primer pas.

En tot el procés seguit per la CPU s'ha de tenir present que la CPU no fa res més que interpretar instruccions escrites en llenguatge màquina. El llenguatge màquina s'ha carregat prèviament en memòria. El programa encarregat de posar el codi màquina en memòria s'anomena carregador. La CPU llegirà el programa de posicions consecutives de memòria a partir d'una posició de memòria inicial predeterminada.

Aquest esquema de funcionament introduït en els primers computadors continua essent totalment vàlid avui en dia. L'arquitectura von Neumann continua essent la base de la major part dels computadors actuals. Les unitats funcionals, encara que presentin variacions segons la implementació, contenen pràcticament els mateixos components. A continuació es presenta amb més deteniment la composició d'aquestes unitats funcionals.

4.2 La CPU

La Unitat Central de Procés (UCP o CPU) és la unitat física responsable de la interpretació de les instruccions contingudes en els programes, del procés de dades aritmètiques i lògiques del computador i de l'entrada/sortida de dades amb l'entorn. Tot i que des dels inicis dels computadors la forma, el disseny i la fabricació ha canviat, la funció principal de la CPU continua essent la mateixa. Consta de la unitat de procés i la unitat de control.

4.2.1 La unitat de procés

La unitat de procés (UP) conté tots els elements d'execució de la CPU. Tot i que pot variar segons l'arquitectura seguida, en la unitat de procés s'hi troben els següents components (recordar la figura 4.1):

- La unitat aritmètico-lògica (UAL). És la unitat responsable d'executar les operacions aritmètico-lògiques (tal com diu el seu nom) del processador.
- El comptador de programes. És un registre (pot ser un comptador) que guarda l'adreça de la propera instrucció a executar.
- El registre d'instrucció. És el registre que té la instrucció que s'executa. La instrucció que s'executa es carrega en la fase de cerca de la instrucció.
- Un banc de registres de propòsit general. Per a facilitar la tasca de la ALU és normal incloure en la UP un conjunt de registres que guarden les dades amb les que opera la UAL. Arquitectures molt simples que en la instrucció només poden adreçar un únic operand requereixen almenys d'un registre, anomenat acumulador en aquests casos, per a poder fer operacions la UAL. Avui en dia, lo més normal, és que la UP disposi de un conjunt registres que poden realitzar funcions diverses. Per exemple, el Intel 8086 disposava d'un conjunt de 4 registres de propòsit general, on cadascun d'ells feia una tasca determinada segons la instrucció que s'executava. En contrapartida és fàcil trobar avui en dia microcontroladors de amb un banc de 16 registres de propòsit molt general.
- Registre apuntador a pila. En el processador hi ha instruccions de salt a subrutina que requereixen continuar (un cop s'acaba l'execució del subprograma) en la instrucció següent a la de salt. Per fer això possible cal guardar en la pila o *stack* l'adreça de retorn. L'apuntador a pila indica posició a la que cal guardar l'adreça de retorn.

- El registre d'estat. És un registre associat a la UAL que guarda l'estat de la CPU. Està format per un conjunt de bits (anomenats *flags* o banderes) que poden indicar tant l'estat en què està treballant la CPU com contenir informació relativa a la darrera operació realitzada per la UAL. Aquesta informació sol incloure els bits de zero, si hi ha hagut excés (*overflow*), si 'ha produït carreteig (*carry*), el bit de signe en operacions amb operands amb signe, si estan activades les interrupcions, etc.
- Depenent de l'arquitectura i comunicació amb la memòria a vegades es parla dels registres d'accés a memòria (*Memory Address Register* o MAR) i del *buffer* de memòria (*Memory Buffer Register* o MBR). Són registres que fan d'interfície entre la CPU i la memòria en la transferència de dades i que tenen com a missió principal adequar la velocitat de transferència de dades entre ambdós components.

4.2.2 La unitat de control (UC)

La unitat de control és la unitat responsable de tot el sincronisme que s'estableix en el processador. S'encarrega d'establir les connexions en la CPU que garanteixen l'execució del cicle d'instrucció i l'execució de les instruccions.

En la realització de la unitat de control hi ha dues implementacions clàssiques:

- Basada en màquina d'estats finits. La UC es construeix sobre una màquina d'estats finits on cada estat defineix l'actuació de cada element de la unitat de procés.
- Basada en una màquina microprogramada. Cada instrucció s'implementa en base a un conjunt de microinstruccions que s'executen de forma seqüencial. Permet reduir el circuit global però complica tot modificació posterior de la unitat de control.

4.3 El repertori d'instruccions

El repertori d'instruccions (*instruction set*, en anglès) és el conjunt de comandes que la CPU pot interpretar. El repertori d'instruccions ha d'estar pensat per a la màquina que s'ha definit. L'execució d'una determinada instrucció implica una personalització del processador emprant el conjunt d'elements de què es composa: operació, registres, adreçament, interrupcions, etc. Conjuntament amb l'estructura establerta de la CPU en defineix l'arquitectura que té el computador.

Diferents màquines tenen diferents tipus d'instruccions. Cada màquina té el seu format d'instrucció. Cada instrucció incorpora informació essencial per a la configuració de la màquina quan s'executa. La interpretació de la instrucció es fa en base a uns camps que identifiquen aspectes del funcionament de la CPU. Aquests camps són:

- La codificació de la instrucció. Tota instrucció té un codi únic que la identifica davant la màquina per a ser interpretada
- Els operands o dades de treball. Ha d'especificar sobre quin tipus de dades treballa: registres, constants, ports.
- Adreçament. La màquina ha de saber on es troben els operands i els ha d'anar a cercar. Cada màquina té els seus modes d'adreçament.

Cada instrucció és a baix nivell el codi màquina que personalitza el computador quan s'executa. Els camps que conté cada instrucció s'associen als components del computador i a la seva personalització per a cada instrucció.

Exemple 1.

Enmig de l'execució d'un determinat programa pel processador EduP12 es troba el següent conjunt d'instruccions (l'apèndix A1 conté el conjunt d'instruccions d'EduP12):

```
cerca: LD R16, +X
CP R16, R17
BRCC cerca
```

Quines implicacions tenen aquestes instruccions en la forma de treballar de la CPU?

Primer de tot cal veure el codi màquina del conjunt d'instruccions. Cada instrucció es codifica amb un format particular establert en el repertori d'instruccions. Tot i que no s'entra en aquest moment a analitzar en detall el format, s'ha de posar per introduir la complexitat que hi ha darrera la codificació de cada instrucció.

La taula 4.1 mostra com es codifica cadascuna d'aquestes instruccions. A través del seu anàlisi es veu com influeix cada instrucció en la CPU. Es poden analitzar una a una (fem-ho de més simple a més complexa):

Instrucció	Format	Codi instrucció	Codop	Camp 1	Camp 2	Camp 3
CP R16, R17	cccc ccdd dddd ssss	0001 1011 0000 0001	000110	10000	10001	
BRCC cerca	cccc kkkk kkkk kbbb	1111 1111 1110 1000	1111	111111101	000	
LD R16, +X	cccc zyxd dddd ccbc	1100 0011 0000 0001	1100 - 00-1	001	10000	0

Taula 4.1. Codificació de les instruccions de l'exemple 1.

- Instrucció *CP R16, R17*. Es llegeix com a *Compare with Register*. És una instrucció de comparació que resta del registre R16 el valor del registre R17, sense guardar el valor final. Les implicacions que té en la CPU són:
    - Com a instrucció aritmètico-lògica, empra la UAL. Per tant, altera el valor del registre d'estat.
    - Instrucció que consta de tres camps: codi d'operació, registre amb què opera i constant amb què opera. La primera fila de la taula 4.1 especifica aquests camps.
- Instrucció *BRCC cerca*. Es llegeix com a *Branch if Carry Clear*. És una instrucció de salt condicional. Si el resultat del bit de carreteig no s'ha activat (és zero) salta a la instrucció etiquetada amb la paraula *cerca*; en cas contrari continua en seqüència. Les implicacions que té en la CPU són:
    - És una instrucció de salt i -en principi- no treballa amb la UAL[3].
    - Consta de tres camps: codi d'operació, un operand (l'adreça a saltar) i el bit sobre el què es realitza la comparació. El codi d'operació ve predeterminat. Els darrers tres bits (camp bbb) indiquen l'estat sobre el que es compara (en aquest cas el de carreteig, i per això el seu valor és 000). Finalment la constant k indica el salt relatiu (respecte a la posició actual del PC) a fer. Com que s'ha d'anar a 3 posicions prèvies, val -3 que, en complement a 2 és el valor corresponent al camp 2 de la instrucció BRCC.
- Instrucció *LD R16, +X*. Correspon a la instrucció *Load from Indirect X with preIncrement to R16*. És una instrucció de càrrega que té dos passos. Primer incrementa el valor del registre X.

[3] De fet sí que s'empra la UAL ja que es fa un salt relatiu respecte a la posició actual. Però en no ser una instrucció aritmètico-lògica no s'actualitza el registre d'estat.

Després va a cercar la dada que es troba en la posició de memòria adreçada pel registre X i la carrera en el registre R16. Les implicacions que té en la CPU són:

- És una instrucció d'entrada amb la memòria de dades a on va a cercar la dada. Tot i que empra la UAL per incrementar el registre X, no és una instrucció aritmètico-lògica.

  Instrucció que consta de quatre camps: el codi d'operació, un registre índex (X, en aquest cas), el registre receptor (R16) i el camp de pre-increment/decrement del registre índex. La instrucció també porta implícit un adreçament indirecte per anar a cercar la dada.

El codi màquina d'aquesta instrucció s'especifica en la darrera fila de la taula 4.1.

Es veu com l'estructura de la CPU i el repertori d'instruccions són part essencial de l'arquitectura del processador. S'ha introduït, també, que hi ha diferents maneres d'accedir a les dades. En la instrucció LD s'ha dit que s'utilitza un adreçament indirecte. En aquest cas implica que el registre conté l'adreça de memòria que conté la dada que s'ha de carregar.

4.3.1 Classificació de les instruccions.

En l'exemple 1 s'han vist tres tipus d'instruccions. De forma general, les instruccions que es troben en un processador solen pertànyer als següent tipus:

- Instruccions aritmètico-lògiques. Operen amb la unitat aritmètico-lògica. Segons l'arquitectura poden ser instruccions amb immediats (constants), entre acumulador i dada de memòria, de doble registre i de registre únic.
- Instruccions de salt. Es divideixen en dos tipus fonamentals:
  - De salt condicional. Se salta a una posició si es compleix una condició (com la instrucció BRCC en l'exemple 1).
  - De salt incondicional. La instrucció implica un salt en la seqüència del programa. Dintre d'aquest tipus d'instrucció hi trobem les instruccions pròpiament de salt incondicional i les de salt a subrutina.
- Instruccions de transferència. Són instruccions que treballen amb dades de memòria. Fonamentalment hi ha les instruccions de càrrega de dades, que cerquen una dada de memòria i la carreguen en registre, i de guarda de dades, que guarden una dada que es troba en un registre a memòria. Les instruccions de transferència admeten diferents modes d'adreçament (s'introdueix en el proper apartat).
- Instruccions d'entrada/sortida. Realitzen la transferència de dades amb perifèrics d'entrada/sortida.
- Altres instruccions. Hi ha instruccions específiques, com ara d'aturada de procés, d'establiment de bit de registre d'estat, de treball amb pila, etc. Solen ser instruccions específiques.

4.3.2 El processador segons el repertori d'instruccions

De l'apartat es dedueix clarament que la informació incorporada en el repertori d'instruccions està associada directament a l'arquitectura del processador. D'acord amb el repertori d'instruccions, per tant, s'estableix una classificació dels processadors en els termes següents:

- Processadors basats en acumulador. Són arquitectures molt basades en el moviment de dades entre memòria i un registre acumulador. La realització de tota operació també implica la cerca d'un operand de memòria mentre l'altre es troba en l'acumulador.
- Arquitectures basades en pila (*stack*). En aquest cas una dada es troba en la capçalera de la pila, mentre que l'altra es cerca de memòria.
- Processadors basats en banc de registres de propòsit general (RPG). És el cas més corrent avui en dia, molt emprat en microcontroladors i en processadors per a sistemes embeguts. Actualment es distingeixen :
    - Arquitectures basades en memòria de registre. És el cas dels processadors 680x0. Un operand es troba en memòria i l'altre en un registre. La dada del registre sol ser font i destí al mateix temps.
    - Arquitectures denominades *load/store*. Exigeixen el traspàs de dades entre memòria i registres. Els operands de la UAL es troben en els registres. Quan els operands es troben en registres, un d'ells actua com a font i destí.

4.4. Modes d'adreçament

El processador treballa amb dades que s'adrecen a partir dels operands de les instruccions i que es troben guardades en memòria. L'adreça real en la que es troba una dada en memòria s'anomena **adreça física**.

Hi ha dos factors que fan difícil l'accés directe a l'adreça física per part de moltes instruccions:

- Les instruccions estan acotades per un nombre determinats de bits. Les instruccions, per tant, no tenen prou número de bits com per poder adreçar sempre l'adreça física de la memòria.
- Encara que es podés accedir directament a tota la memòria, no resulta pràctic a efectes reals. Es deu al fet que sovint és molt més pràctic i recomanable accedir a memòria des d'adreces relatives o indirectes que no pas absolutes.

Suposem, per exemple, l'accés a una memòria de 4 Gparaules. Una instrucció que adreci directament una dada d'aquesta memòria necessita dedicar-hi 32 bits.

Així una instrucció normalment no treballa amb l'adreça física de la dada, sinó que disposa de diversos mecanismes, o **modes d'adreçament**, que li permeten determinar l'adreça física de l'operand. El mode d'adreçament determina l'operació que cal executar per obtenir l'adreça física en la que es troba l'operand. A aquesta adreça física se l'anomena **adreça efectiva.**

De forma general, es pot establir la següent classificació dels modes d'adreçament:

- Adreçament implícit.
- Adreçament immediat.
- Adreçament directe.
- Adreçament indirecte.
- Adreçament relatiu.
- Adreçament indexat.

Una mateixa instrucció pot admetre diferents tipus d'aquests modes d'adreçament. També sol passar que una màquina mescli entre sí aquests modes d'adreçament.

4.4.1 Adreçament implícit

Es troba en aquelles instruccions que, per pròpia construcció, tenen implícit l'operand amb el que treballen. És el cas d'instruccions de retorn de subrutina, per exemple, que saben que han d'anar a cercar l'adreça de retorn en la capçalera de la pila.

4.4.2 Adreçament immediat

L'adreçament immediat el tenen les instruccions que expliciten l'operand en la pròpia instrucció. No calen, per tant, referències a memòria

La figura 4.2 mostra un esquema d'adreçament immediat. La instrucció conté una constant *k* que és la dada que fa falta per operar. En el processador EduP12, per exemple, totes les instruccions que es tractaran en el grup d'instruccions aritmètico-lògiques amb immediat duen aquest adreçament.

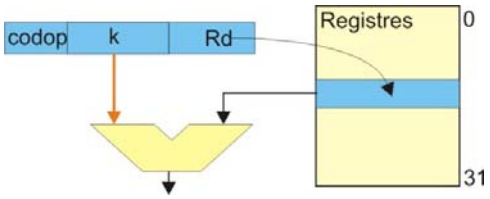


Figura 4.2. Adreçament immediat

4.4.3 Adreçament directe

En l'adreçament directe la instrucció conté l'adreça real de l'operand. Per tant, la instrucció té l'adreça efectiva de la dada. La figura 4.3 mostra que l'operand de 12 bits de la instrucció és l'adreça efectiva a la que es cerca la dada en memòria.

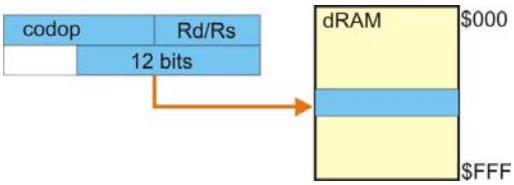


Figura 4.3. Adreçament directe

4.4.4 Adreçament indirecte

En l'adreçament indirecte la instrucció conté un apuntador a l'adreça que conté l'adreça efectiva de l'operand. L'adreçament indirecte implica realitzar un doble adreçament per trobar l'operand.

És un adreçament útil per accedir a taules d'apuntadors. Sovint es combina amb un increment o decrement de l'operand que es pot fer abans o després de la captura de la dada.

La figura 4.4 mostra un esquema de la indirecció que es duu a terme amb aquesta instrucció.

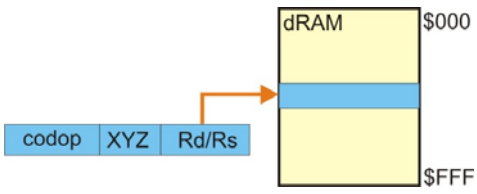


Figura 4.4. Adreçament indirecte

4.4.5 Adreçament relatiu

En un adreçament relatiu l'adreça efectiva es troba sumant a una adreça de referència un desplaçament u offset. L'adreça efectiva s'obté sumant el desplaçament a l'adreça de referència.

Depenent d'on es troba l'adreça de referència es parla de:

- *Adreçament relatiu a la base*. Un registre base conté l'adreça de referència. L'adreça efectiva s'obté sumant l'adreça de referència amb el desplaçament donat en la instrucció. A vegades al registre base se li'n diu registre índex i aleshores es parla d'**adreçament indexat**. Els sistemes operatius en solen fer un bon ús ja que facilita l'accés a diferents camps en una estructura de dades.
- *Adreçament relatiu a comptador de programes*. L'adreça efectiva s'obté sumant el contingut del PC amb el desplaçament donat en la instrucció. És útil en instruccions de salt respecte a la posició actual.
- *Adreçament relatiu a segment*. L'adreça efectiva s'obté sumant a un registre de segment (com podria ser un apuntador a pila) el desplaçament especificat en la instrucció.

La figura 4.5 mostra un adreçament relatiu des de registre índex. L'adreça efectiva es troba sumant el valor del registre índex amb un desplaçament q.

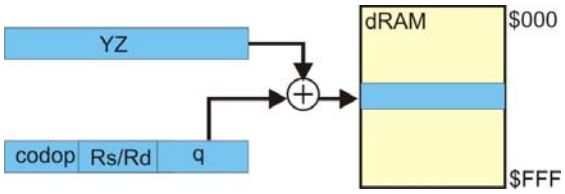


Figura 4.5. Adreçament indirecte relatiu

4.5 La memòria

La memòria és un component fonamental del computador que té la funció de guardar la informació amb la que tracta el computador. Conjuntament amb la CPU forma part del nucli dur de l'arquitectura von Neumann.

El terme memòria inclou un conjunt molt gran i divers de dispositius. Per exemple, fa referència tant a la memòria dinàmica interna RAM, memòria ràpida i propera a la CPU, com a tots els dispositius externs d'emmagatzemament de dades, estàtics i més lents.

4.5.1 Classificació de la memòria

La memòria en el computador es divideix en els següents tipus:

- Memòria primària. És la memòria que ha d'estar connectada a la CPU per a què funcioni correctament el computador. Consisteix en tres tipus principals de memòria:
    - Els registres de la CPU. Els registres de la CPU, com a dispositius de guarda de la informació molt propers a la CPU, configuren la memòria més interna i més ràpida del computador. La seva construcció és a base de registres, com diu el seu nom.
    - La memòria principal. Com indica el nom és la memòria que conté la informació base amb la que treballa el computador en tot moment. Degut a què emmagatzema el programa i dades ha de ser més gran que els propis registres i es comunica amb la CPU a través de busos. És, per tant, més lenta. Per fer-la lo més rapida possible solen

ser memòries d'accés aleatori (RAM) dinàmiques, motiu pel que quan es desendolla el computador perden la informació que contenen.

  - La memòria cau o *cache*. És una memòria redundant, en tant que duplica informació de la memòria principal, que està molt propera a la CPU. La seva missió és fer d'enllaç entre la memòria principal i els registres per tal d'accelerar la velocitat de procés del computador.

- Memòria secundària. També anomenada memòria massiva. És memòria de major capacitat que guarda de forma permanent la informació. Es comunica amb la CPU a través dels canals d'entrada/sortida, fet que la fa més lenta. Per exemple, la lectura d'un byte des de disc dur es mesura en unitats de milisegons, mentre que la lectura d'un byte d'un registre triga pocs nanosegons. L'exemple més clar de memòria massiva és el disc dur. La memòria secundària avui en dia està formada principalment pels discs durs que empren un sistema magnètic per guardar dades i discs òptics.
- Es parla també de memòria fora de línia. Es tracta de sistemes d'emmagatzemament en dispositius externs al computador. Seria l'exemple de les actuals memòries que es connecten per USB, els CDs i DVDs, o les antigues cintes magnètiques, per exemple.

4.5.2 Característiques i jerarquia de memòria

Les característiques principals de la memòria en el computador són:

- Capacitat. S'entén com a la quantitat d'informació que pot emmagatzemar una unitat de memòria. Sol ser mesurada en paraules o en bytes.
- La paraula és la unitat base d'organització de la memòria.
- Unitat adreçable. És la mida de la dada que es pot adreçar. Normalment es mesura en paraules, tot i que en determinats casos també es pot fer l'accés per bytes.
- Al número d'elements que es transfereixen s'anomenen unitats de transferència. La transferència de dades des de, o cap a, la memòria secundària sol ser molt massiva. Aleshores la unitat és el bloc (un conjunt de paraules de cop).
- Velocitat de transferència. És la velocitat a la que es transfereixen les dades des de la unitat de memòria.
- El temps d'accés té diferent significat segons el tipus de memòria. En memòries d'accés aleatori és el temps que es triga en llegir o escriure una dada. En disc durs i altres memòries secundàries és el temps que es triga en situar el mecanisme de lectura o escriptura.
- A vegades es parla de temps de cicle de memòria. Es calcula com el temps d'accés més tot altre temps que es necessita per a realitzar una nova lectura o escriptura.

Davant d'aquestes propietats queda clar que és difícil tenir una memòria ideal per computador. Aquesta memòria hauria de ser ràpida, tenir molta capacitat, i a més hauria de ser ràpida, característiques que entre elles són incompatibles. Això fa que el computador hagi d'utilitzar un conjunt molt divers de memòria. En el nucli central utilitza memòries molt ràpides que poden guardar poca quantitat d'informació, mentre que la memòria massiva es troba allunyada del nucli i necessita temps d'accés grans. Aquesta disposició de la memòria no és casual i es deu a la necessitat de compatibilitzar diferents prestacions. A la disposició de la memòria en el computador se l'anomena jerarquia de memòria.

La jerarquia de memòria representa i classifica la memòria emprada en el computador mitjançant una piràmide (figura 4.6) on els costats representen les magnituds capacitat, velocitat d'accés i cost. Els diferents tipus de memòria utilitzada es posen en ordre dintre aquest diagrama.

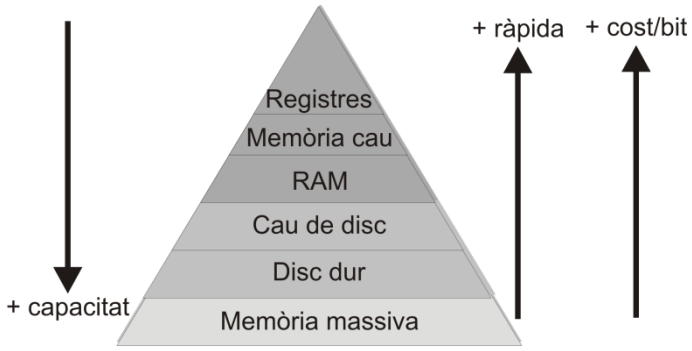


Figura 4.6. Jerarquia de memòria

La lectura del diagrama és clara:

- En la punta de la piràmide hi tenim els registres i la cau just dessota seu. Es tracta de la memòria més ràpida del computador i també és la més cara (de més cost/bit). D'aquí que sigui la que n'hi ha menys.
- En la base de la piràmide hi ha la memòria massiva. És memòria lenta i de baix cost/bit. Per això és la que n'hi ha més

El propòsit de la jerarquia de memòria és trobar la raó ideal de cada tipus de memòria per balancejar correctament la velocitat, la capacitat i el cost de la memòria emprada en el computador.

4.5.3 La memòria RAM

Per norma general s'entén com a memòria RAM la memòria principal del computador encarregada de guardar els programes i les dades de forma temporal durant el funcionament del mateix. Mirat des del punt de vista de la jerarquia de memòria és una memòria força ràpida, d'un cost mig-alt per bit, motiu pel que és més aviat escassa si es compara amb la memòria secundària.

La memòria RAM, com ja s'ha introduït en el capítol 3, és una memòria d'accés aleatori (prové de *Random Access Memory*), el que indica que totes les dades es poden obtenir directament a partir del bus d'adreces i que totes tenen el mateix temps d'accés, fet que fa que no calgui el posicionament de cap braç o altre mecanisme (com passa amb la memòria secundària que empra un accés seqüencial).

Una memòria RAM, com a tal, és un circuit integrat que conté memòria d'accés aleatori. Pot ser memòria RAM estàtica o dinàmica. La memòria RAM estàtica manté les dades emmagatzemades mentre el circuit estigui alimentat. En canvi la memòria dinàmica només conserva les dades durant uns milisegons, fet que fa que s'hagi de refrescar per mantenir els valors guardats. Es parla de SRAM i DRAM, respectivament.

La memòria principal del computador està formada per mòduls SIMM, DIMM, SO-DIMM, entre altres similars. Són mòduls de memòria formats per circuits DRAM. La comunicació amb la CPU es fa mitjançant tres tipus de senyals:

- El bus de dades. És el bus emprat per rebre i enviar a la CPU les dades de la memòria.
- El bus d'adreces, que s'encarrega de portar l'adreça amb la que es treballa amb la memòria.
- Senyals addicionals de treball amb la memòria com ara les línies de selecció de mòdul i de lectura/escriptura.

4.5.4 Memòria ROM

La memòria ROM (*Read Only Memory*) és una memòria d'accés aleatori de només lectura que s'empra per emmagatzemar dades de forma permanent, fins i tot quan se'n va l'alimentació.

La seva missió en el computador és doble. Per una part conté una rutina d'inici del computador que determina el hardware de la màquina, detecta el correcte funcionament dels dispositius i arrenca la màquina: llegeix el registre *boot record* (de disc dur o altre dispositiu) que s'encarrega de posar el sistema operatiu en la RAM. En segon lloc té una rutina anomenada BIOS (*Basic Input-Output System*) que activa els dispositius d'entrada/sortida (teclat, rateta, ...).

4.5.5 La memòria cau

La memòria cau es troba entre la memòria principal i els registres formada per memòria SRAM, més ràpida que la DRAM.

La memòria cau forma és un *buffer* entre la memòria principal i els registres per permetre funcionar més ràpid la CPU. S'utilitza per guardar els programes i les dades que s'estan usant. Quan es necessita algun recurs (programa o dada) que no està en la cau s'ha de cercar en la memòria principal (i fins i tot en memòria secundària si tampoc està en memòria principal). Això fa que s'estableixi un protocol d'intercanvi de dades que és tot un repte. Hi ha múltiples estudis sobre quins són els recursos que s'han d'intercanviar.

La comunicació entre CPU i memòria cau es fa paraula a paraula. La comunicació entre cau i memòria principal es fa per blocs.

4.5.6 El disc dur

El disc dur (HDD o *Hard Disk Drive*) és un sistema no volàtil d'emmagatzemament que empra un sistema magnètic per guardar les dades.

Internament el disc dur conté un conjunt de discs concèntrics (figura 4.7) que giren a gran velocitat amb un capçal en cada cara del disc. Els capçals estan situats en uns braços que van endins i enfora. Juntament amb el moviment rotatori del disc fa que el capçal pugui llegir qualsevol part del disc. Els discs giren a una velocitat molt alta (és normal la velocitat de 7200 rpm). El gir tan ràpid del disc crea una capa d'aire entre capçal i disc que fa que el capçal no toqui mai el disc dur. De fet, de tocar, es destruirien les dades del disc.

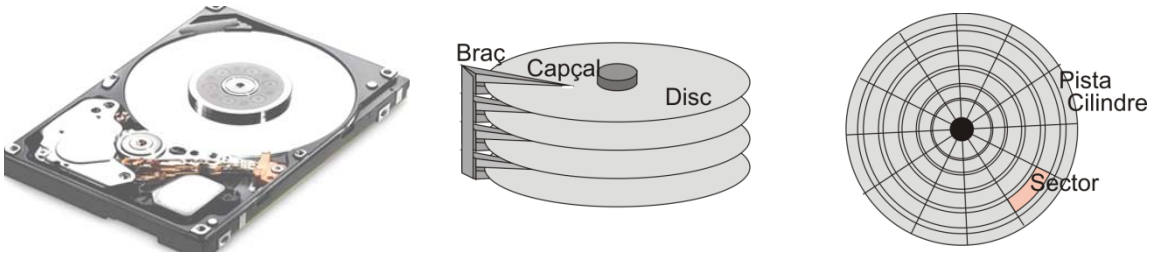


Figura 4.7. Estructura del disc dur.

L'estructura del disc dur és la següent:

- Disc. Cadascun dels discs magnètics que conté el disc dur.
- Cara. Cadascuna de les dues cares de cada disc
- Capçal. Dispositiu lector que es troba en la punta del braç. Cada disc té dos capçals, un per cada cara del disc.
- Pista. Cadascuna de les unitats concèntriques en què el disc es divideix. La pista 0 es troba en l'exterior. Entre pista i pista hi ha un espai de separació d'informació.
- Cilindre. Conjunt de pistes alineades verticalment que ocupen la mateixa posició en cada disc.

- Sector. Cadascuna de les divisions que es realitza en una pista. Actualment cada sector té 512 bytes.
- El sector conté la informació, que es troba en tres camps: preàmbul, informació i control d'errors. El preàmbul conté bits d'inici de sector i el número de pista i sector. Les dades estan en el camp d'informació. El camp de control d'errors conté informació per a la recuperació d'errors.

Amb aquesta distribució la informació per pista és fixa, i la densitat d'informació és alta en el centre però baixa a l'exterior. Noves tècniques, com la gravació de bits per zona que millora el nombre de sectors en el disc, intenten optimitzar la capacitat del disc dur.

La temporització establerta en un disc dur consta dels següents temps:

- Temps de cerca de pista. És el temps d'accés a una pista-
- Latència rotacional. Temps d'espera fins que el sector passa sota la pista
- El temps d'accés és la suma del temps de cerca de pista més el temps de latència rotacional.
- Temps de transferència de bloc. És el temps total que es triga en transferir un bloc.

El disc dur es connecta amb la placa base amb diferents connexions:

- IDE/ATA - *Integrated Device Electronic / Advanced Technology Attachment*. És un controlador de dispositius d'emmagatzemament massiu com el disc dur. Va ser l'estàndard fins el 2004 per la seva versatilitat i facilitat d'ús.
- SCSI – *Small Computer System Interface*. Interfícies preparades per discs durs de gran capacitat i velocitat. Un controlador SCSI pot manejar fins a 7 discs durs, amb velocitats de transferència de dades de fins a 20Mbits/s
- SATA – *Serial ATA*. Empra un bus sèrie. Més ràpid i eficient que els IDE. Actualment arriba als 600 MB/seg.
- ASA – *Advanced Serial SCSI*. Interfície de transferència de dades sèrie successor del SCSI amb millor velocitat de transferència. Pot augmentar la velocitat de transferència a major nombre de dispositius connectats.

4.5.7 Disc òptic

És un disc planer que pot enregistrar dades binàries a través d'alts i baixos en la seva superfície que codifiquen 0's i 1's. L'escriptura es fa amb làsers a través de la inserció de solcs en la pista. En la lectura el làser reconeix la dada emmagatzemada a través de la reflexió diferent que fan els 0's i els 1's.

El primer disc òptic va ser el làser-disc, introduït a finals dels 70. Actualment els discs òptics més emprats són el CD (*Compact Disc*) i el DVD (*Digital Versatil Disc*).

Tant el CD com el DVD tenen una única pista disposada en espiral que s'inicia en el centre i va fins a l'exterior. A diferència del disc magnètic llegeixen a velocitat lineal constant, si bé a velocitats superiors a 12x es manté la velocitat angular constant.

CDs estàndards actuals admeten una capacitat 700 MB, el que els hi permet reproduir so durant 80 min. La velocitat base d'un CD d'àudio permet llegir 150 KB/s. El factor 54x d'alguns CDs actuals indica que poden ser llegits a una velocitat de 150·54 KB/s.

El fet que el DVD empri un làser de longitud d'ona més petita que el CD fa que les osques puguin ser més petites permeten una capacitat d'emmagatzemament sis vegades superior a la del CD. La capacitat estàndard d'un DVD està en 4.7GB. Els DVDs de capacitat doble són DVDs de doble cara. La

velocitat de transferència de dades base del DVD és de 1350 KB/s. Cal tenir present el format en els DVDs. La diferència entre DVD+ i DVD- està en la separació entre osques.

El BD o Blue-Ray Disc és un disc òptic de nova generació d'alta capacitat d'emmagatzemament de dades i vídeo d'alta definició. Pot emmagatzemar 25 GB d'informació.

4.5.8 Memòria d'estat sòlid

La memòria d'estat sòlid (SSD o *Solid State Drive*) és un dispositiu d'emmagatzemament secundari no volàtil d' informació fet amb memòria d'estat sòlid amb l'objectiu de substituir els discs durs.

Donat que no tenen parts mòbils són més immunes als cops. Són menys sorolloses. Els temps de latència són menors, fet que les fa més ràpides. Tenen menor pes. Per això són millors són ideals en computadors portàtils. Per altra part també són més cares, no arriben a les capacitats d'emmagatzemament dels discs durs i en cas de pèrdues de dades són més difícils de recuperar.

Una memòria SSD consisteix de:

- Un controladora que és un dispositiu processador que s'encarrega de la interfície entre la unitat SSD i la memòria principal.
- I una cau que s'encarrega de controlar la transferència de dades cap a memòria principal.
- També disposa de dispositiu (condensador) de manteniment puntual de l'alimentació que li permet acabar la transferència actual en cas que es produeixi una aturada inesperada de l'alimentació.

4.5.9 Gestió de la memòria

Durant el funcionament normal del computador la necessitat de memòria que es necessita per poder córrer tots els programes que s'estan executant en un moment determinat pot ser molt major a la memòria cau i principal de què disposa el computador. Aleshores cal que el sistema operatiu gestioni els programes que en un moment determinat han d'estar carregats. Així, la gestió de memòria és l'ús que el computador en fa de la memòria per a proveir de recursos els programes conforme s'han d'anar executant. La gestió de la memòria:

- Ha de subministrar l'espai de memòria per a la gestió de múltiples processos amb un rendiment acceptable des del punt de vista de l'usuari.
- Si es pot ha de compartir recursos entre programes.
- Ha de protegir els recursos dels programes.
- Ha de fer tota aquesta gestió de memòria de manera transparent a l'usuari.

Per tant, la gestió de memòria ha de portar un control dels recursos de memòria que calen als programes que s'estan executant, alliberant la memòria quan no faci falta i assignant-la a altres programes que en tinguin necessitat.

En la gestió de memòria és important el concepte de memòria virtual. La **memòria virtual** és un sistema de gestió de la memòria dins del sistema operatiu que 'fa creure' al programa que està en posicions consecutives de la memòria quan, en realitat, pot estar trossejat en diferents parts de la memòria. Això és pràcticament obligat en programes grans que no entren en les posicions de memòria principal assignades quan s'han d'executar.

El maquinari que s'encarrega de gestionar l'accés a memòria des de la CPU s'anomena **unitat de gestió de la memòria**, i la seva funció és la de convertir les adreces virtuals en adreces físiques.

4.6 Entrada/sortida

En un computador s'entén per entrada/sortida al conjunt de sistemes responsables de coordinador l'intercanvi d'informació entre els perifèrics i el processador. D'aquesta manera el computador pot reaccionar a canvis de l'entorn i hi pot intervenir si és el cas. En el sistema d'entrada/sortida s'hi troben involucrats tots els perifèrics i els diferents dispositius que s'encarreguen del control de l'entrada/sortida.

Una classificació inicial de perifèrics els divideix en:

- Perifèrics d'entrada. Són els dispositius que capten i envien dades des del dispositiu cap al processador per a ser tractades. Entre els perifèrics d'entrada més comuns hi ha el teclat, el ratolí (en les seves diferents versions: de bola, de control –Joystick-, aeri, etc.), el micròfon, l'escàner, la càmera, lectors de codis de barra, etc.
- Perifèrics de sortida. Són dispositius que reben dades de la CPU. Aquí hi ha la targeta gràfica (i subseqüent monitor), la impressora, la tarja de so (i corresponents altaveus), el plotter, etc.
- I perifèrics d'entrada/sortida. Es tracta dels dispositius que tant poden enviar informació cap a l'ordinador com rebre-la des de l'ordinador. És el cas més comú del sistema de memòria secundari: disc dur, gravadores/lectors de CD/DVD, cintes, pantalles tàctils, etc.

En l'entrada/sortida hi intervenen els següents components:

- Els dispositius d'entrada i sortida.
- Les targetes controladores dels perifèrics.
- I la connexió entre aquests elements que empra els busos d'adreces, de dades i de control del processador.

La comunicació entre processador i perifèrics no és simple, ja que la cada perifèric té una manera concreta de funcionar. Entre les característiques que s'han de tenir presents hi ha:

- El sincronisme entre CPU i perifèric. La velocitat d'entrada o sortida de dades de cada perifèric ve predeterminada. Per la seva part la CPU normalment necessita treballar a velocitats altes per respondre a totes les necessitats de procés que té. Convé doncs establir un protocol de comunicació que permeti a la CPU no deixar desatès el perifèric i que, al mateix temps, no li faci perdre cicles de CPU esperant al perifèric. Per altra part, perifèrics molt ràpids no haurien de saturar la CPU, ja que aquesta aleshores no podria atendre les necessitats de procés requerides.
- El sistema d'interconnexió amb la CPU és específic per a cada perifèric. Un controlador de perifèric ha de permetre compatibilitzar l'entrada/sortida de dades del perifèric amb la CPU.
- El sistema operatiu se'n sol responsabilitzar d'establir la comunicació entre perifèrics i processador.
- El tipus de transferència que s'estableix amb el perifèric. Hi ha dispositius que transfereixen quantitats molt petites d'informació, com pot ser el cas de la comunicació sèrie amb una UART en la que la unitat de transferència és el byte. Altres han de transferir grans quantitats d'informació, motiu pel que solen tenir controladors específics que transfereixen la informació per blocs, com és el cas de la memòria.

Segons el tipus de transferència hi ha diferents mecanismes emprats en l'entrada/sortida:

- Transferència elemental. S'usa en comunicacions simples en les que la comunicació s'estableix per a un conjunt molt petit de dades i que es poden realitzar a la velocitat d'interconnexió del bus del sistema. En aquest cas s'estableix una comunicació física entre la CPU i el perifèric i es poden transmetre bits, bytes o un conjunt reduït de paraules.

La transferència es realitza per hardware i pot involucrar la conversió de dades en sèrie a paral·lel i viceversa.

- Transferència per bloc. Quan el conjunt de dades a transferir és gran és usual preparar la comunicació per passar un *bloc* de dades de manera sincronitzada amb el perifèric. Es sol tenir un dispositiu d'emmagatzemament temporal de la informació per evitar la pèrdua de dades durant la transmissió.

  La transferència de dades es pot realitzar controlada per software o per DMA (*Direct Memory Acces*).

4.6.1 Model de perifèric

El perifèric és tot component que no forma part del nucli central del processador i es connecta a ell emprant els busos del sistema. La connexió al sistema es sol fer emprant controladora que fa d'interfície entre perifèric i CPU. La controladora disposa d'una part electrònica llur funció és la d'adaptar la velocitat de procés i el format de les dades amb la CPU.

Per tant, el mecanisme d'E/S de cada perifèric realitza les següents operacions:

- Sincronitza la transferència de dades. Per exemple, la comunicació sèrie requereix de velocitats de transmissió entre 300 i 115200 bauds. Una controladora típica per a la comunicació sèrie és una USART (*Universal Synchronous/Asyncrhronous Receiver and Transceiver*). La CPU sol treballar a velocitats molt superiors a la USART. Mitjançant bucles d'espera i interrupcions s'adequa la velocitat de transmissió entre la USART i CPU.
- I controla la transferència de dades entre perifèric i CPU. El control es realitza a través de senyals d'estat, que indiquen a la CPU com es troba el perifèric, i senyals de control que indiquen al perifèric les accions que ha de fer. En el cas de la USART, per exemple, un senyal d'estat típic és indicar al processador que acaba de rebre una dada. En sentit contrari, quan la CPU coneix que l'etapa de transmissió està inactiva, pot enviar una dada al mòdul transmissor per a iniciar la comunicació. En el cas d'una impressora, els senyals de control s'empren per encendre o apagar la impressora, posicionar el capçal de la impressora, etc

La figura 4.8 mostra un esquema genèric d'un perifèric en el que s'observa que el controlador forma la interfície entre el dispositiu d'E/S i la CPU. La comunicació entre CPU i controladora es fa a través de tres busos: el bus de dades, encarregat de la transmissió de la informació; el bus d'adreces, encarregat de determinar el perifèric amb el què s'estableix la comunicació; i el bus de control que entre altres, conté les línies de control del perifèric específic amb què s'està treballant.

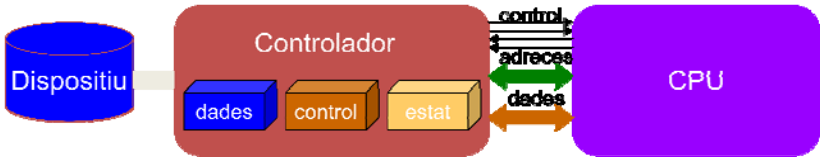


Figura 4.8. Model genèric de perifèric

El fet que en la connexió entre perifèric i CPU hi hagi un bus d'adreces indica, directament, que la identificació del perifèric per part de la CPU es fa a través d'una o més adreces específiques. Aquest fet permet que a un processador se li puguin connectar una quantitat indeterminada de perifèrics, doncs la identificació passa per proporcionar les adreces que requereixi la controladora. A partir d'aquí serà la controladora la responsable d'entendre's amb la CPU davant de cap petició d'informació.

Entre perifèric i CPU s'estableixen tres mecanismes bàsics de comunicació:

- Comunicació per entrada/sortida programada
- entrada/sortida per interrupció
- I entrada/sortida per DMA

4.6.3 Entrada/sortida programada

L'entrada/sortida programada realitza la transferència de dades entre la CPU i el perifèric emprant les instruccions d'entrada/sortida del processador. Aquesta comunicació té les següents característiques:

- La transferència triga el que dura una instrucció.
- Típicament s'utilitzen instruccions IN i OUT per a implementar instruccions d'entrada/sortida amb perifèrics que es comuniquen amb l'entorn, i les instruccions de càrrega i guarda de dades, LD i ST respectivament, quan es tracta de transferències amb memòria.
- Les resposta d'un perifèric d'entrada/sortida depèn de:
    - L'adreça. Un perifèric només respondrà si l'adreça correspon a la adreça d'entrada/sortida que se li ha assignat.

      Hi ha CPUs, sol ser típic en microcontroladors, que no distingeixen entre el mapa d'adreces d'E/S i la memòria principal.
    - El tipus d'operació que s'exigeix. Els senyals de control especifiquen l'acció a realitzar pel perifèric adreçat i el camí per on passen les dades.
    - La temporització, factor que influeix en la transferència de dades. Pot ser síncrona o asíncrona.
- Prioritat del perifèric. Quants diversos perifèrics comparteixen la CPU cal establir un mecanisme de prioritat en la comunicació.
    - Es parla de gestió distribuïda quan els peticionaris es posen d'acord per determinar qui es queda amb el recurs. Un mecanisme es pot fer emprant lògica distribuïda, en la que la pròpia lògica del perifèric que es connecta al bus estableix, entre tots els concursants, les prioritats d'accés.
    - Quan la gestió és centralitzada hi ha un mestre que dirigeix la cessió del recurs. És, per exemple, el cas de la connexió en *daisy-chain* que connecta tots els perifèrics amb una línia comuna de petició de servei i una de concessió. El sistema ve controlat per un mestre que dóna la cessió del bus de dades.

4.6.4 La interrupció

La interrupció és un mecanisme emprat en els computadors que indica al processador que ha d'interrompre l'execució actual que fa d'un procés per a atendre un servei especial. La interrupció passa aleshores a executa el que s'anomena una *rutina de servei de la interrupció* que sol estar governada per la BIOS.

Les interrupcions, depenent del mecanisme que les produeix, es poden classificar en dos tipus:

- De la mateixa CPU quan es produeix un fet important que requereix ser tractat per la pròpia CPU. En aquest cas es sol parlar de dues variacions possibles:
    - *Traps* o trampes produïdes per errors durant l'execució del programa (com per exemple, una divisió per 0) que es podrien evitar si s'analitzés en compte el programa.

    - Errors software o *excepcions* són interrupcions produïdes per una operació no permesa no controlada directament pel programador.

- Pot ser deguda a un event extern que demani l'atenció de la CPU. Per exemple, en un processador dedicat al procés en temps real de senyals externs, es corrent programar interrupcions per demanar l'atenció del processador davant canvis en senyals externs.

La interrupció involucra els següents mecanismes:

- El cicle de petició de servei d'interrupció. S'informa a la CPU que s'ha produït un event que requereix atenció especial per part de La CPU.
- El cicle de concessió de la interrupció. Un cop s'ha acabat la instrucció que s'està executant, la CPU identifica la interrupció que s'ha produït i realitza un salt en l'execució normal del programa per atendre la interrupció.
- S'atén la interrupció. S'executa la rutina de servei de la interrupció que atén la interrupció produïda.
- Finalment es desactiva la interrupció produïda.

Quan es produeixen vàries interrupcions a un mateix temps hi ha diferents alternatives que les tracten:

- Inhibició d'interrupcions. No es permet cap interrupció fins que s'hagi atès la que es tracta. Hi ha diferents possibilitats, com ara desactivar la possibilitat de qualsevol nova interrupció, o desactivar només les interrupcions de prioritat inferior.
- Interrupcions simultànies. Permet que hi hagi diferents interrupcions actives a un mateix temps. El tractament de cadascuna pot seguir un sistema de prioritat o per *polling*, és a dir, es va comprovant si s'ha produït una interrupció.
- Interrupcions anidades. Mentre es tracta una rutina de servei se succeeix i tracta una nova interrupció. Es pot seguir un sistema de tractament d'interrupció per prioritat en la interrupció, de manera que només puguin suspendre la rutina de servei d'interrupció actual aquelles interrupcions més prioritàries.

La interrupció és un potent mecanisme que permet al sistema operatiu emprar la CPU per servir una segona aplicació mentre s'espera que acabi una operació d'entrada/sortida. El hardware s'encarrega d'avisar quan el dispositiu d'entrada/sortida ha acabat. Aleshores el sistema operatiu, si és el cas, pot passar a continuar la tasca que estava esperant que s'acabés l'entrada/sortida.

Un exemple d'operació d'entrada/sortida per interrupció es l'ús del teclat del computador. Quan s'introdueix un caràcter es codifica en el registre de dades del dispositiu i s'activa un interrupció en el hardware. El processador passa a atendre la rutina d'interrupció corresponent. El caràcter es guarda temporalment en el buffer del dispositiu i desperta el procés d'espera de l'entrada/sortida.

4.6.45 Augment de prestacions en E/S: DMA

De forma força general els dispositius d'entrada/sortida són dispositius lents comparats amb la velocitat de la CPU, prou lents com per què la CPU no pugui estar preguntant contínuament per l'estat del dispositiu. Per descarregar de feines al processador els perifèrics tenen mecanismes que els hi donen certa llibertat d'actuació de forma que treballen amb paral·lel amb el processador. Per exemple, en la lectura de dades de disquet la CPU pot informar al controlador corresponent per tal que arrenqui el dispositiu mentre la CPU continua fent tasques de procés. En acabar, el dispositiu interromp a la CPU per notificar-li que està preparat.

Dos mecanismes genèrics de transferència de dades que permeten descarregar a la CPU del control de perifèrics són:

- L'accés directe a memòria o DMA
- L'establiment de canals d'E/S. En general, un canal és una unitat d'entrada/sortida que funciona per DMA.

El *Direct Memory Accés* o DMA és una tècnica emprada actualment per molts dispositius d'entrada/sortida que permet l'accés directe a la memòria per part del perifèric sense necessitat d'interrompre contínuament el processador.

En el DMA el controlador del perifèric es comunica directament amb la memòria principal del computador i en porta el control de la transferència sense intervenció de la CPU. La CPU informa al controlador de la tasca a realitzar. En acabar, el controlador interromp a la CPU per donar-li la informació sol·licitada.

L'operació de DMA es fa per blocs de dades. Per a realitzar la transferència de dades el controlador de perifèric necessita conèixer característiques concretes de la transmissió a realitzar:

- L'adreça amb què ha de treballar en memòria principal, per a poder donar les ordres de control correctes
- El tipus d'operació a realitzar, si és d'entrada o de sortida
- El nombre de dades a transferir
- I l'adreça del perifèric

Donat que l'accés a memòria es pot portar a terme per diferents usuaris, cal establir mecanismes de control de l'accés. Dos mecanismes bàsics són

- A través d'un control multiporta de la memòria. La CPU es connecta a una de les múltiples portes de què disposa la memòria per poder ser accedida. Cada porta és un controlador d'accés a la memòria.
- Per robatori de cicle. En aquest cas la memòria té una única porta d'accés que és compartida per tots els usuaris, i normalment amb control de la CPU. Aleshores, quan el perifèric necessiti realitzar una transferència a memòria haurà de dialogar amb la CPU per què li cedeixi un cicle o fase per a realitzar la transferència.

4.7 Comunicacions i busos

En el computador el bus és un sistema digital que s'empra per comunicar els components dintre del computador o per comunicar diferents computadors entre sí.

Les primeres arquitectures de computadors transferien dades entre els diferents components (CPU, memòria i perifèrics) a través de busos particulars entre els components, fet que comportava complicacions cada cop que es volien introduir millores en l'arquitectura.

Per solucionar aquests problemes, una de les primeres arquitectures amb busos va establir la compartició de connexions entre els dispositius del computador per transferir dades amb un arbitratge per part de la CPU. La CPU se'n encarrega de generar els senyals de control que asseguren la correcta connexió de components en la transferència de dades tot indicant a través de l'adreça els dispositius que intervenen en la comunicació. El sistema de busos que transporta dades paral·leles entre els dispositius del computador està constituït fonamentalment pels busos de dades, d'adreça i de control (figura 4.9). Per aquest menester els primers computadors empraven una placa de circuit imprès base, anomenada *backplane*, que només tenia el connexionat del bus i en la que s'hi connectaven tots els components. D'aquest temps és el bus ISA aparegut a finals dels anys 70.

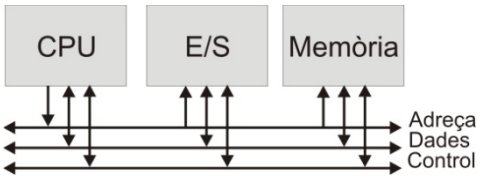


Figura 4.9. Sistema simple de busos

A la que els computadors varen començar a anar més ràpid i a la CPU se li requeria més temps de procés es va veure la necessitat d'alliberar a la CPU de la tasca d'arbitratge del bus. Es va crear el *chipset* que té per funció jerarquitzar els busos del computador creant el bus de sistema que s'encarrega de la connexió de la CPU i la RAM i el bus d'expansió per al a comunicació amb els perifèrics. El bus ISA, per exemple, va passar a ser un bus d'expansió. En els busos PCI i posterior AGP se'ls hi va introduir majors capacitats de procés autònom.

Actualment en un computador s'estableix una gama àmplia de comunicacions que obliga a fer una distinció entre els busos, creant la jerarquia de busos. S'estableixen els següents nivells:

- Busos de tipus 0. Fa referència als busos interns al xip. No són accessibles a l'usuari, però en determinen la connexió del xip al sistema, ja que en determinen l'ample de paraula
- Busos de tipus 1. És el bus d'interconnexió en una placa de circuit imprès. Les seves característiques venen determinades per les prestacions dels busos de xip i solen ser interns a la pròpia placa. Comuniquen dispositius separats per desenes de centímetres.
- Busos de tipus 2. Connecten diferents plaques en un mòdul, formant el *backplane*. A nivell de computador, avui en dia passa a ser el bus de sistema, arribant al centenar de senyals de control i estableixen comunicacions de longitud inferior al metre. Els busos ISA, VME, PCI en són exemples.
- Busos de tipus 3. Connecten diferents mòduls i poden arribar als 10 metres de longitud, fet que obliga a tractar-lo com a línia de transmissió. Necessiten utilitzar terminals o *buffers* en la seva connexió.
- Busos de tipus 4. En formen part els busos de connexió paral·lela entre perifèrics. Poden tenir una configuració molt diferent a la del bus de sistema (com el port *Centronics* –avui pràcticament desaparegut- que s'empra per establir la comunicació amb impressores). Un port paral·lel empra un connector de 25 pins, transfereix paraules de 8 bits i arriba a velocitats de 2Mbit/s
- Busos de tipus 5. Actualment els busos sèrie del computador constitueixen el tipus elemental de comunicació en sistemes informàtics. Sovint la informació es transmet en codis ASCII i s'empren en xarxes locals de baixa velocitat. La velocitat de transmissió és d'1 a 100Mbit/s, i poden comunicar components distants a pocs kilòmetres. Entre les normes més emprades hi ha la IEEE 802.

  La comunicació en el bus sèrie pot ser simple, semidúplex (un fil, dos sentits alternants) o full-dúplex (dos fils). Entre les normes més comunes de comunicació sèrie entre terminals hi ha la RS232C. Permet tant la connexió d'equips informàtics amb mòdems com sense mòdem. La temporització ve definida per la velocitat de transmissió que pot agafar valors entre 110 i 921600 bauds. Actualment està en desús.

  El relleu l'est¡a agafant el bus USB, amb velocitats de 1.5Mb/s a 12Mb/s, començant a entrar el USB2 que pot arribar a velocitats de transmissió de 480Mb/s. És un bus sèrie destinat a la connexió de perifèrics lents: ratolí, teclat, escàner, etc.

Les connexions actuals de xarxa com ara Ethernet van més enllà del bus del PC i es tracten en ordre diferent. Altres busos, com ara el I2C poden ser tractats com busos interns o externs.

La figura 4.10 mostra un esquema de la jerarquia de busos que s'estableix en el computador. Els busos estan jerarquitzats formant diferents nivells per a la connexió dels diferents dispositius. L'objectiu final de la jerarquia de busos és obtenir una millor eficiència del sistema.

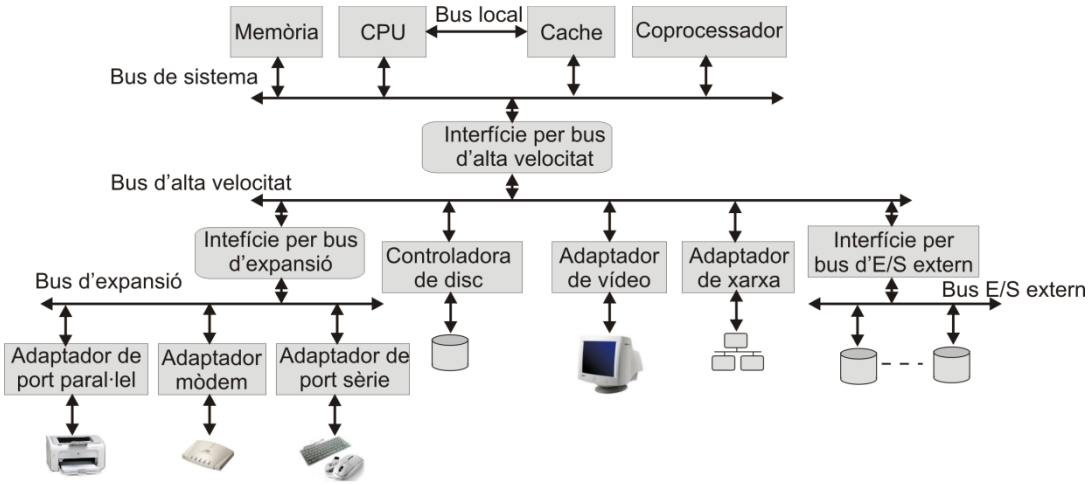


Figura 4.10. Jerarquia de busos

4.8 Màquines Harvard i Von Neumann

L'arquitectura Harvard és una arquitectura molt utilitzada en microcontroladors que presenta les següents variants respecte a l'arquitectura von Neumann:

- L'arquitectura Harvard emmagatzema en memòries diferents el programa i les dades
- Els espais d'adreces de les memòries de programa i de dades són diferents
- Les arquitectures de les memòries de programa i de dades poden ser diferents

El fet que l'arquitectura Harvard pugui llegir instrucció i dades a un mateix temps fa que sigui una arquitectura més ràpida. En contra, el dimensionament de les memòries s'ha de preveure adequat per evitar que cap d'elles quedi infrautilitzada.

Tot i aquestes diferències, el principi de funcionament és similar al de l'arquitectura Von Neumann. La figura 4.9 mostra un esquema de l'arquitectura Harvard on es veu que la similitud amb l'arquitectura von Neumann.

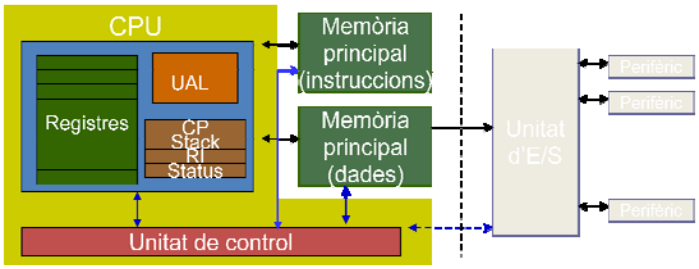


Figura 4.9. Arquitectura Harvard

4.9 Classificació bàsica dels computadors

L'any 1966 M.J. Flynn va formular una classificació bàsica dels computadors que es coneix avui en dia com a taxonomia de Flynn. La classificació estableix com a paràmetres de classificació el nombre d'operacions que són executades concurrentment i el flux de dades que arriben al processador. Classifica els processadors en quatre tipus:

- SISD o *Single Instruction, Single Data Stream*. És una arquitectura simple sense cap mena de paral·lelisme. És l'exemple clàssic de processador simple, com els computadors personals en els seus inicis. Avui en dia els computadors personals exploten el paral·lelisme emprant múltiples processadors.
- SIMD o *Single Instruction, Multiple Data Stream*. Són processadors que una única arquitectura permet explotar el paral·lelisme en dades. Pot ser el cas d'arrays de processadors emprats en unitats de procés gràfiques (GPU o *Graphic Processing Unit*), un processador especialitzat en la realització d'operacions gràfiques o operacions en punt flotant que permet alleugerir la càrrega del processador en aplicacions gràfiques i de videojocs.
- MISD o *Multiple Instruction, Single Data Stream*. És un cas especial en el que una dada és processada en múltiples unitats, normalment només emprat en casos de tolerància de falles.
- MIMD o *Multiple Instruction, Multiple Data Stream*. En aquest cas hi ha diferents processadors executant múltiples instruccions sobre dades diferents. Es tracta de càlculs distribuïts que poden treballar sobre un mateix espai de dades o espais de dades diferents. En aquest classificació s'hi compten els supercomputadors.

A nivell de CPU és normal parlar d'arquitectures CISC (*Complex Instruction Set Computer*) i arquitectures RISC (*Reduced Instruction Set Computer*).

La terminologia CISC neix sobre els anys 80 en aparèixer l'arquitectura RISC i per diferenciar-se d'aquesta, tot i que l'arquitectura CISC té el seu naixement a començaments dels 70 amb la introducció del microprocessador per part d'Intel, essent el precursor del què seria el nucli de la tecnologia emprada posteriorment en el PC.

La característica principal dels processadors CISC és que tenen un codi màquina ampla per permetre la realització d'operacions complexes en un nombre petit d'operacions. A nivell de nucli de CPU es caracteritzen per tenir cada instrucció màquina microprogramada. Això és, cada instrucció executa un petit codi microprogramat en una memòria interna a la CPU. La conseqüència principal és que es redueix el nombre d'instruccions assemblador. Aquest fet redueix els costos de creació de programes, compacta el codi i facilita la tasca del compilador.

En contrapartida els processadors RISC es basen en instruccions senzilles que poden ser executades ràpidament pel processador. Per això sempre realitza operacions amb operands que es troben en un banc de registres que es troba proper a la CPU, fet que implica que una operació CISC pot necessitar diverses instruccions RISC. Per això, un codi RISC necessitarà de més memòria per a emmagatzemar un programa determinat i el compilador tindrà una tasca més feixuga alhora de trobar el codi màquina. En contrapartida té un conjunt d'avantatges importants:

- Les instruccions tenen un format fix i s'encabeixen en pocs formats.
- Les operacions són ràpides per què s'opera amb el banc de registres. Només les instruccions de càrrega i guarda accedeixen a memòria.
- En ser les instruccions molt simples al final el temps d'execució és similar al de les instruccions complexes d'un processador CISC.
- El hardware per implementar les instruccions RISC és més simple.
- S'optimitza l'ús dels registres de propòsit general i facilita el paral·lelisme intern.

L'arquitectura RISC facilita la segmentació i el paral·lelisme quan s'executen les instruccions en el processador. Forces microprocessadors actuals estan basats en aquesta arquitectura, des del PowerPC fins al ARM que és molt usat en sistemes embeguts.

4.10 Resum del capítol

Aquest capítol ha donat una visió general d'un sistema ordenador genèric. Tots es basen en un conjunt d'elements (que pot ser molt divers) com ara CPU, perifèrics, memòria, comunicació, entrada/sortida, etc. Aquests components estan tots lligats a través de la CPU que mana i ordena les ordres a executar i que fan funcionar l'ordenador.

El capítol ha servit com a introducció del capítol 5 que es centra ja en un processador específic, l'EduP12. Molts dels conceptes teòrics introduïts en aquest capítol es posaran a la pràctica. El conjunt d'instruccions i l'adreçament suportat forma el llenguatge elemental de la CPU que serveix per a la seva programació.

4.11 Exercicis resolts

A partir de dos exemples concrets es mostra el plantejament d'algorismes de resolució de programes en assemblador. És una metodologia fàcil d'aprendre que convé seguir sempre. Com es veu a continuació, els passos a resoldre són, primer trobar el diagrama de flux i després passar-lo a assemblador. Aquesta manera de resoldre problemes en assemblador s'emprarà a partir del proper capítol.

Exercici resolt 1. El primer algorisme.

Es volen sumar dos nombres que s'agafen d'un perifèric d'entrada i es vol mostrar el resultat en un perifèric de sortida.

Solució

Plantejament

El processador es troba connectat a l'entrada/sortida del sistema a través de ports d'entrada. En aquest exercici s'empra un port d'entrada i un de sortida anomenats PortA i PortB respectivament.

El plantejament en aquest exercici és molt simple. Es guarden en dos registres diferents les dues dades que entren pel PortA. Aleshores es sumen, guardant el resultat en un dels registres, i es treu pel PortB.

Diagrama de flux

El diagrama de flux és un mètode gràfic d'expressió de l'algorisme. Es basa en la construcció d'un graf amb símbols predefinits que indica la tipologia de tasca a realitzar. En general es solen emprar els següents símbols:

- Rectangle → Indica execució.
- Rectangle inclinat → Implica acció d'entrada/sortida
- Rombe → Pregunta per una condició, actuant de diversa manera segons el valor d'aquesta.
- Oval → S'empra per indica inici i fi de programa
- Cercle → Emprat per connectar diferents parts del programa.

El diagrama de flux de l'exercici es mostra en la figura 4.10. Es pot observar que en aquest cas és molt directe. Cal observar com cada símbol indica l'acció que es realitza. L'execució de l'algorisme soluciona el problema plantejat.

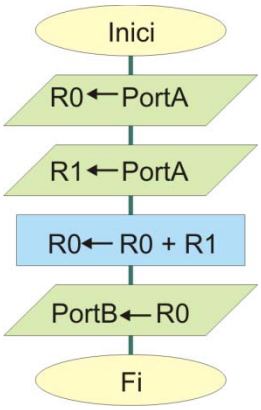


Figura 4.10. Diagrama de flux de l'exercici 1.

Un diagrama de flux per a un programa en assemblador sol ser convenient fer-lo a baix nivell. Com que les instruccions assemblador es basen en transferència de dades entre registres, les accions convé especificar-les en llenguatge transferència de registres. És útil emprar la nomenclatura establerta pel processador.

Programa en assemblador

La traducció del diagrama de flux en un programa assemblador és directa. Donat que les instruccions ja s'han posat en format adequat pel processador, només cal traduir els símbols en les instruccions assemblador adequades. La figura 11 mostra el programa assemblador d'aquest exercici.

```
IN R0, PortA;
IN R1, PortB;
ADD R0, R1;
OUT PortB, R0;
```

Figura 4.11. Codificació en assemblador de l'exercici 1.

Exercici resolt 2. Cap a un algorisme genèric

Treure pel PORTB el resultat de sumar tots els nombres primers entre 0 i 100.

Solució

Plantejament

L'exercici en aquest cas empra una iteració sobre una variable: la variable *som* (que tindrà el resultat final) anirà sumant la variable índex *a*, fins que aquesta arribi al final de la iteració. Per tant, es realitzen les operacions:

som ← som + a
a ← a+2

Un punt fonamental és trobar les condicions inicials i finals. En aquest cas les condicions inicials són: la variable *som* ha de partir de 0 i la variable índex del primer valor a sumar *a* val 1. La condició final es troba quan la variable índex sobrepassa el límit, que en aquest exercici és 100.

Diagrama de flux

El diagrama de flux es dóna en la figura 4.12. S'hi identifiquen clarament les variables som i a. Es pot observar que, tot i seguir el format RTL, en aquest cas s'han emprat variables normals enlloc de registres del processador. El seu ús permet identificar més fàcilment les variables emprades. Un cop es passa a l'assemblador és fàcil assignar registres del processador a les variables normals.

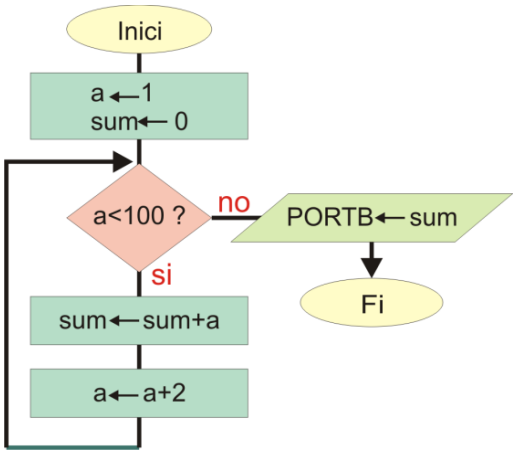


Figura 4.12. Diagrama de flux de l'exercici 2.

Programa en assemblador

La traducció en assemblador del programa es dóna en la figura 4.13.

```
.DEF som = R10;
.DEF a = R11;
inici: LDI som, 0;
       LDI a, 1;
proper: CPI a, 100;
       BRPL fi;
       ADD som, a;
       ADDI a, 2;
       RJMP proper;
fi: OUT PORTB, som;
       RJMP fi;
```

Figura 4.13. Codificació en assemblador de l'exercici 2.

Es veu clarament com l'ús de variables genèriques facilita la comprensió del programa. La conversió de variables genèriques a registres la realitza la directiva *.DEF*. Per altra part, el salt a parts del programa es fa amb instruccions de salt com *RJMP* o *RCALL*, i el lloc on es salta s'identifica amb etiquetes, aquí *inici*, *proper* i *fi*.

4.12 Exercicis

1. Trobar la velocitat lineal a què gira les pistes interna i externa d'un disc dur. I en el cas d'un DVD?

2. La UART és una unitat de transmissió sèrie d'informació entre dos dispositius. La figura següent mostra el format que se segueix: quan un byte es vol enviar, aquest s'empaqueta amb un 0 en la posició menys significativa i un 1 en la més significativa, formant un paquet de 10 bits que s'envien a través del port sèrie. Quan no es transmet res la línia queda en repòs a 1-lògic. Adjunt en la taula es mostren velocitats de transmissió típiques.

Donat per suposat que la transmissió de cada bit dura 16 cicles de rellotge base del sistema, es demana:

i) Calcular la freqüència de rellotge mínima per a cada *baudrate*. S'entén com a *baudrate* la velocitat de transferència ò nombre de bits enviats per segon.

ii) Calcular quan temps es triga en transmetre un byte d'informació per a cada *baudrate*.

iii) Quants bytes es transmeten, en cada cas, per segon?

Velocitats típiques (*Baudrate*)
2400
4800
9600
19200
38400
57600
115200

Exercici 2. Format de transmissió sèrie.

3. Donats els següents algorismes descrits en assemblador:

i) Trobar els corresponents diagrames de flux.
ii) Dir que realitza cadascun d'ells.
iii) Treballant amb una freqüència de rellotge de 20MHz i suposant que cada instrucció dura un cicle de rellotge, quan triga l'execució de cada algorisme?

Nota: Tot i que la funció que realitza cada instrucció és bastant evident, els apèndixs A1 i A2 detallen la seva funcionalitat.

```
--ALGORISME 1
inici:     LDI R0, 100;
bucle:     OUT PORTB, R0;
           DEC R0;
           BRPL bucle;
continuar:      ...
```

```
--ALGORISME 2
.DEF A = R0;
.DEF B = R1;
.DEF tmp = R2;

inici:     LDI A, 1;
           LDI B, 0;
bucle:     OUT PORTB, A;
           MOV tmp, A;
           ADD A, B;
           MOV B, tmp;
           CPI A, 1000;
           BRMI bucle;
continuar:      ...
```

```
--ALGORISME 3
.DEF tmp0 = R0;
.DEF tmp1 = R1;

inici:     LDI tmp0, 0xFFF;
bucle2: LDI tmp1, 0xFFF;
bucle1: DEC tmp1;
           BRNZ bucle1;
           DEC tmp0;
           BRNZ bucle2;
continuar:      ...
```

Exercici 3. Algorismes.

Capítol 5

EL PROCESSADOR EDUP12

5.1 Introducció.

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

- Necessitat de tenir un processador complert però potent, i al mateix temps simple, que servís per a l'aprenentatge de fonaments de computadors.
- 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 per a ser emprat en tasques educatives i en aplicacions no educatives.
- Envoltat d'eines d'aprenentatge adequades: assemblador, simulador elemental del nucli del processador, i simulador de programes en assemblador.
- Descrit com a nucli software, en VHDL, per a ser implementat sobre FPGA per a aplicacions diverses.
- I amb codificació d'instruccions clàssica, no massa diferenciada del repertori d'instruccions de processadors actuals, que permeti a qui s'introdueixi en el món dels processadors 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 per a l'ensenyament de fonaments de computadors.

Amb EduP12 s'aconsegueix una versió simple però pràctica d'un processador 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)[4].

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

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

de la CPU es veuen als ports d'E/S com a registres de memòria que són accedits a través d'adreces prefixades.

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

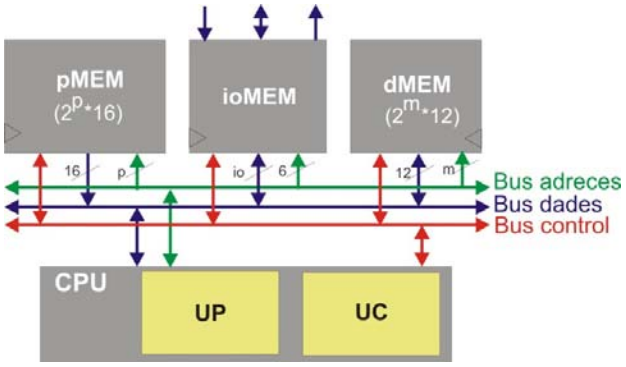


Figura 5.1. Estructura del processador

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 seqüència 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.

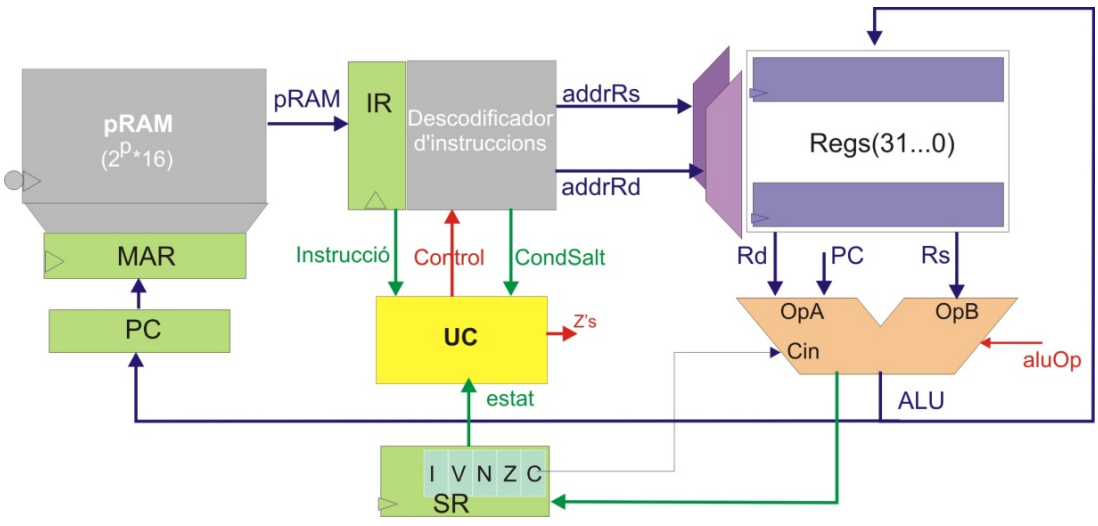


Figura 5.2. La unitat central de procés

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 UAL. 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 UAL, 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 intermedis 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[5]:

- 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 UAL, 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'incrementa el PC.

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

- 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 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 intermèdia 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.

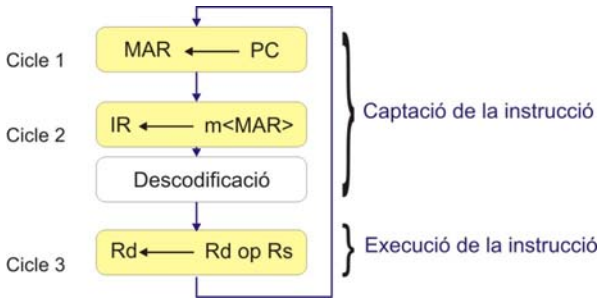


Figura 5.3. Cicle d'instrucció per a una instrucció aritmètico-lògica

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.

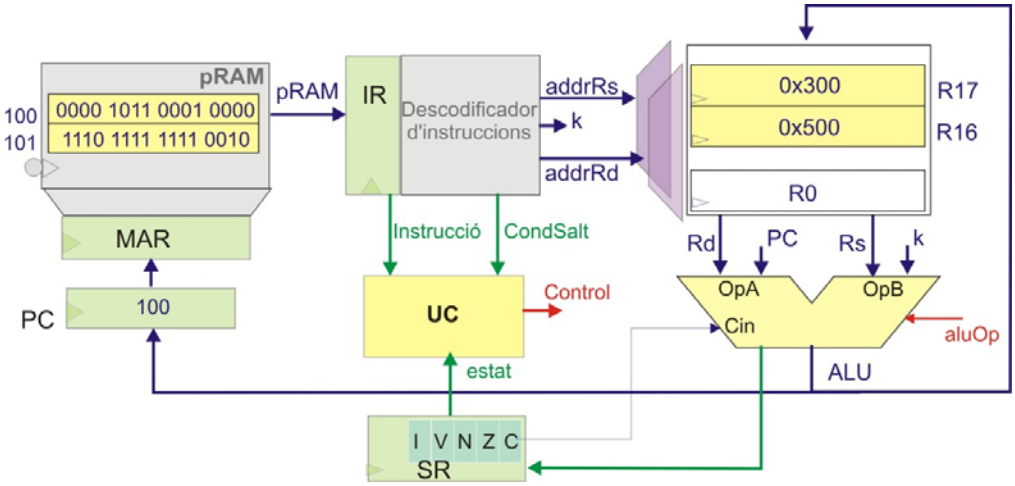


Figura 5.4. Estat de la CPU en el moment d'iniciar l'execució de la instrucció ADD R0, R1

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).

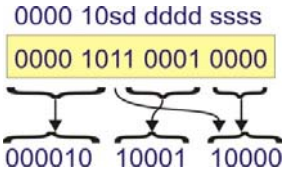


Figura 5.5. Correspondència de bits en la instrucció ADD R17, R16.

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 UAL, 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.

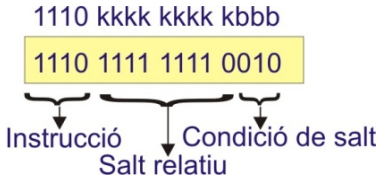


Figura 5.6. Anàlisi de la instrucció BRMI

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.

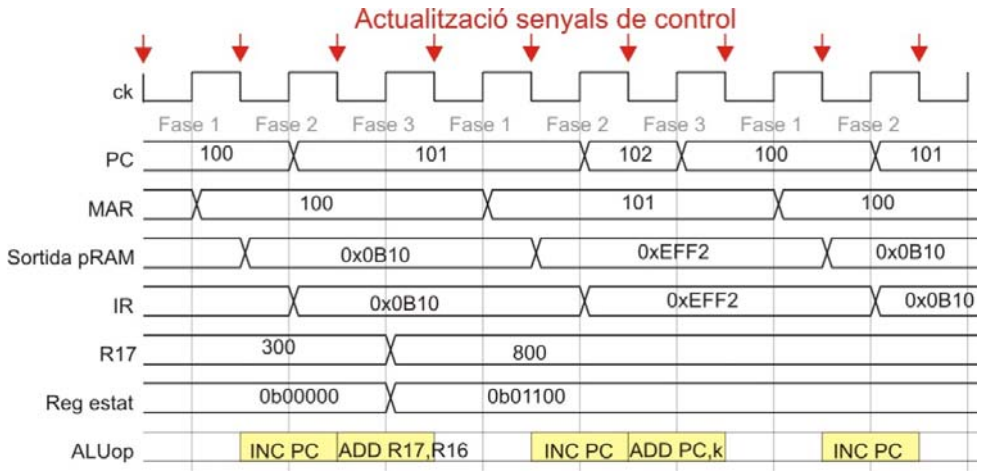


Figura 5.7. Diagrama temporal d'evolució del processador

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. 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.

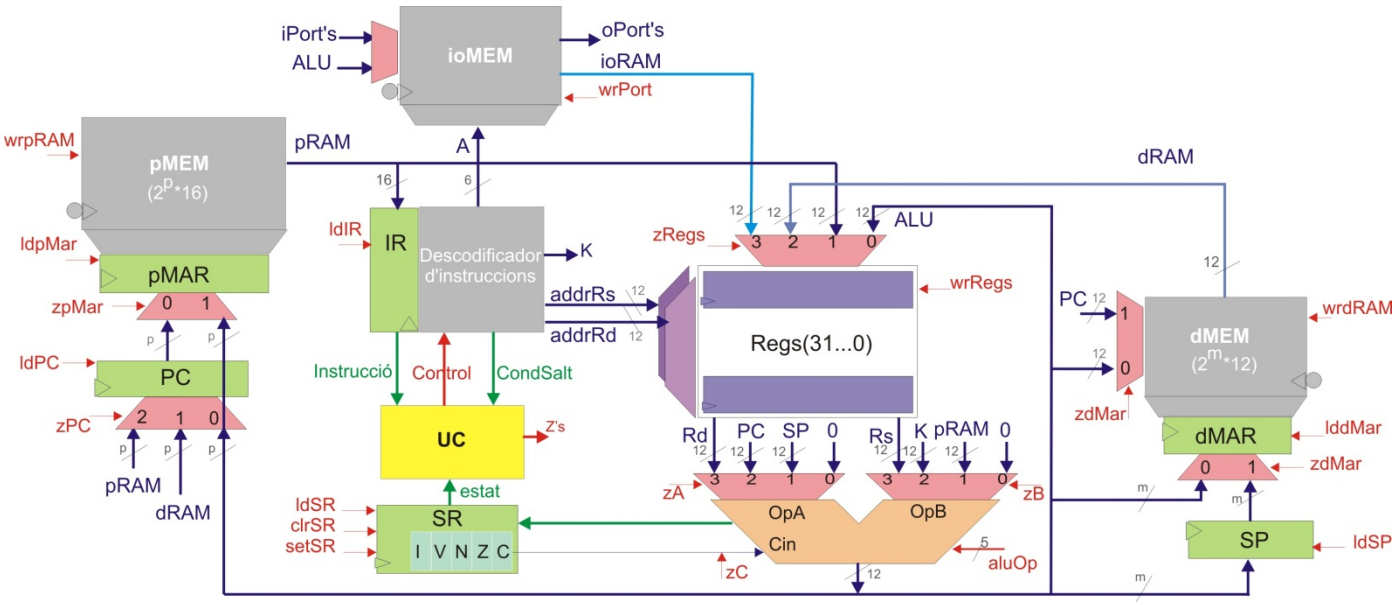


Figura 5.8. Arquitectura de EduP12

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.

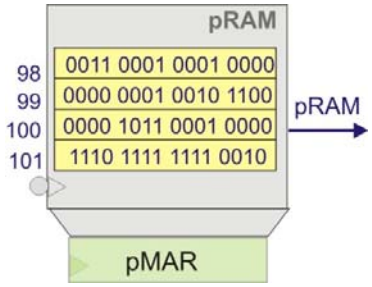


Figura 5.9. Introducció de la instrucció LDI R17, 300 en la posició 98 de la memòria.

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.

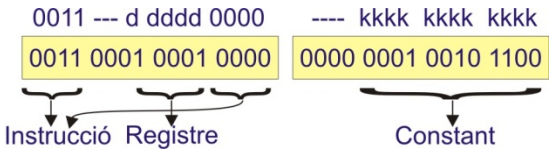


Figura 5.10. Descodificació de la instrucció LDI R17, 300.

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.

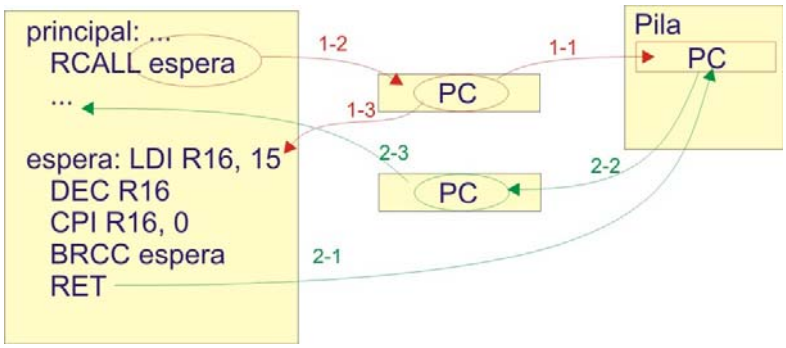


Figura 5.11. Execució instrucció RCALL.

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.

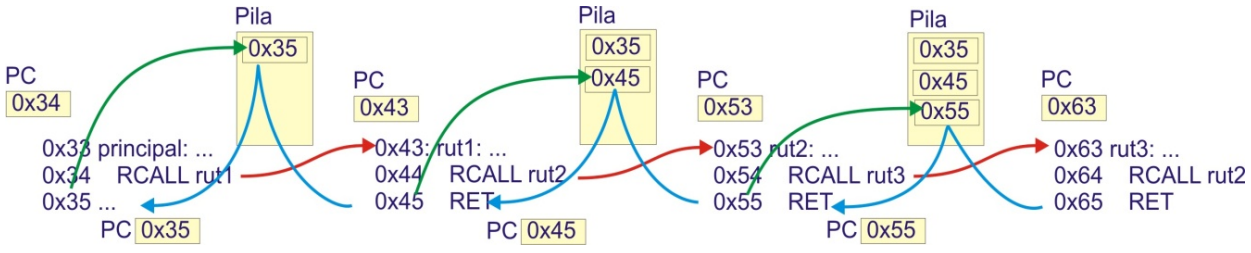


Figura 5.12. Anidament de subrutines: La pila guarda les adreces de retorn que es recuperen en executar *RET*.

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 dRAM ← PC	zddMAR←1, lddMAR←1, zA←1, opALU←DEC, ldSP←1 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.

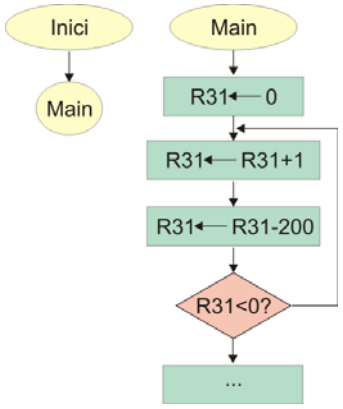
5.5 Exercici resum

La figura 5.14a correspon a un determinat programa que realitza una rutina de retard. Agafant aquest programa com a base es demana (entre parèntesis s'ha posat, per clarificar l'exercici, la posició de memòria en la que es s'inicia cada tros de programa):

i) Donar un diagrama de flux (en format RTL) del programa.
ii) Trobar el codi màquina del programa. Ajudeu-vos de l'Apèndix A1.
iii) Trobar els cicles de rellotge en els que es descompon la instrucció SUBI. Donar l'activitat dels senyals de control que s'activen durant la seva execució.
iv) Calcular el retard (nombre de cicles de rellotge) de la subrutina de retard suposant que cada cicle dura exactament un cicle del rellotge base i que la freqüència del rellotge és de 20 MHz.

5.5 Exercici resum

```
(0)              RJMP main
(32) main:       LDI R31, 0
(33) bucle:      INC R31
(34)             SUBI R31, 200
(35)             BRMI bucle
(36)             ...
```



a) **b)**

Figura 5.14

Apartat i).

La figura 5.14b mostra el diagrama de flux. S'observa que el programa escrit mostra un salt relatiu inicial abans de passar a executar una altra part. En capítols posteriors s'explica el motiu d'aquest salt.

Apartat ii).

El codi màquina de les instruccions es troba de forma directa analitzant el format de cada instrucció que ve donat en l'Apèndix A.

Programa			Format instrucció	Codi màquina	
(0)		RJMP main	0100 kkkkkkkkk	(0)	0100000000100000
(32)	main:	LDI R31, 0	0011 --- ddddd 0000	(32)	0011000111110000
			0	(33)	0000000000000000
	bucle:	INC R31	0011 --- ddddd 1000	(33)	0011000111111000
		SUBI R31, 200	0011 --- ddddd 0100	(33)	0011000111110100
			200	(33)	0000000011001000
		BRMI bucle	1110 kkkkkkkkk 010	(34)	1110111111110010
		...	...	(35)	...

La transformació de totes les instruccions és directa. Com a càlcul a realitzar només s'ha de trobar el salt relatiu des de la instrucció BRMI, que és simple: donat que es troba en la posició 34, que quan s'executa el PC ja ha incrementat (i val PC = 35), i que el salt a fer és a la instrucció 33, el salt relatiu és de -2. Cal recordar que s'ha de posar en complement a la base.

Apartat iii).

- Cicles de rellotge de la instrucció SUBI i activació de senyals de la unitat de procés per a la seva execució.

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	R31 ← R31 – mem<MAR>	zA←3, zB←1, aluOp←SUB, zRegs←0, wrRegs←1

Apartat iv).

El càlcul de la duració de la rutina d'espera es troba determinant el nombre de cicles totals de rellotge que triga en executar el bucle. Es troba fàcilment que:

Instrucció	#cicles	#cops que s'executa	Total (cicles)
INC	3	200	600
SUBI	4	200	800
BRMI	3	200	600
		Total	2000

Per tant, la duració total del bucle és de 2000 cicles, que tenint present una freqüència de rellotge de 20MHz implica que triga en executar 2000 / ($2*10^7$) = 100µseg

5.6 Exercicis plantejats

1. Trobar el codi màquina del conjunt d'instruccions següent:

a. MOV R0, R1
b. CLR R16
c. ANDI Z, 0x88
d. BRID 0x444
e. LPM +X, R16
f. ST +X, R16
g. LDD R24, Y+50

2. Trobar els cicles de rellotge en què es descomponen el conjunt d'instruccions de l'exercici 1.

3. Es té el següent programa (entre parèntesis s'ha posat l'adreça d'inici de cada tros de programa).

```
        ( Suposar que R31 agafa inicialment el valor 1000 )
(32)    main:   RCALL b1
                RJMP main
                ...
(100)   b1:     RCALL b2
                RET
(200)   b2:     INC R31
                RET
```

Es demana:

i) Donar un diagrama de flux del seu funcionament.
ii) Donar un diagrama d'evolució temporal del processador en el que es vegi l'evolució dels registres de la CPU involucrats en l'execució del programa.

4. Donat el següent programa determinar:
   i) El codi màquina del programa.
   ii) Determinar la duració de la rutina d'espera .

```
espera: LDI R0, 0xFF
b0:     LDI R1, 0xFF
b1:     DEC R1
        BRCC b1
        DEC R0
        BRCC b0
        RET
```

Nota: Entre parèntesis s'ha posat l'adreça d'inici de cada tros de programa.

Capítol 6

LLENGUATGE MÀQUINA DE EDUP12

En aquest capítol s'introdueix el repertori d'instruccions del processador EduP12 i els modes d'adreçament amb què treballa.

6.1 El repertori d'instruccions.

El *repertori* o *joc d'instruccions* és el conjunt de comandes que implementa una CPU. El joc d'instruccions sol incloure diferents tipus d'instrucció com aritmètico-lògiques, de transferència de dades, d'entrada/sortida, de salt i d'altres de genèriques.

La instrucció forma part de l'arquitectura del processador ja que porta immersos detalls sobre el cablejat que s'ha d'establir durant la descodificació de la mateixa.

Diferents màquines tenen diferents tipus d'instruccions. Cada màquina té el seu format d'instrucció. Però sempre la interpretació de la instrucció es fa en base a uns camps que identifiquen aspectes del funcionament de la CPU. Aquests camps són:

- La codificació de la instrucció. Tota instrucció té un codi únic que la identifica davant la màquina per a ser interpretada
- Els operands o dades de treball. Ha d'especificar sobre quin tipus de dades treballa: registres, constants, ports.
- Adreçament. La màquina ha de saber on es troben els operands i els ha d'anar a cercar. Cada màquina té els seus modes d'adreçament.

En el processador EduP12 la longitud de la instrucció és de 16 bits. Les instruccions es poden agrupar per categories, mantenint certa uniformitat dintre les categories. A més el repertori d'instruccions és variat i incorpora un conjunt de modes d'adreçament que el fan útil per a aplicacions diverses.

6.2. Modes d'adreçament

EduP12 treballa amb el següent conjunt de modes d'adreçament:

- Adreçament implícit.
- Adreçament immediat.
- Adreçament directe.
- Adreçament indirecte.
- Adreçament relatiu.
- Adreçament indexat.

Una mateixa instrucció pot admetre diferents tipus d'aquests modes d'adreçament. També sol passar que una màquina mescli entre sí aquests modes d'adreçament. Per exemple, es veurà que en EduP12 hi ha instruccions que mesclen els modes d'adreçament indirecte i relatiu.

6.2.1 Adreçament implícit

Es troba en les instruccions de retorn de subrutina (RET, RETI) que han de recuperar de la pila l'adreça de la propera instrucció a executar.

6.2.2 Adreçament immediat

En el processador EduP12 totes les instruccions que es tractaran en el grup d'instruccions aritmètico-lògiques amb immediat duen aquest adreçament. La figura 6.1 mostra un esquema del format d'aquestes instruccions. La dada o *immediat* ve especificada per la constant *k* de la instrucció.

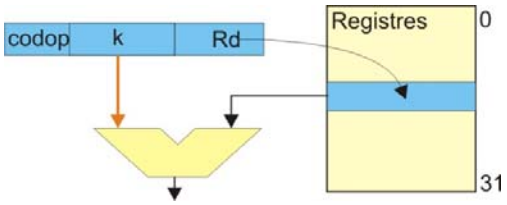


Figura 6.1. Adreçament immediat

6.2.3 Adreçament directe

En EduP12 es poden considerar tres tipus d'adreçament directe:

- Adreçament directe en registre. Es troba en les instruccions que treballen amb dades que es troben en el registre. L'adreça de l'operand és el propi registre. Per exemple, és el cas de la instrucció *ADD R16, R17*: els dos operands estan en registres.
- Adreçament directe en memòria d'entrada/sortida. Quan es treballa amb un port d'entrada/sortida l'adreça de l'operand és el port, que és qui conté la dada amb la que es treballa. Per exemple, en l'execució de la instrucció *IN Rd, PORT*, PORT és l'adreça efectiva on hi ha l'operand.
- Adreçament directe en memòria de dades. De fet és el mateix cas de l'adreçament directe amb memòria d'entrada/sortida, només que ara l'adreça efectiva es troba en una segona paraula. Com a exemple en la instrucció *LDS Rd, 0x123*, 0x123 és l'adreça (efectiva) de memòria que conté l'operand (figura 6.2).

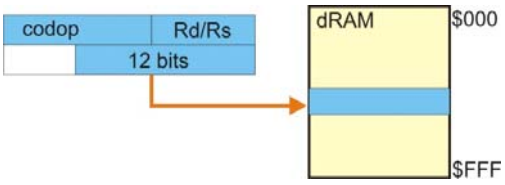


Figura 6.2. Adreçament directe

6.2.4 Adreçament indirecte

EduP12 implementa l'adreçament indirecte per registre. És a dir, un registre conté l'adreça efectiva de l'operand. Els registres d'indirecció només poden ser els registres R29, R30 o R31, anomenats X, Y i Z, respectivament. També admet el pre-increment o pre-decrement del registre.

Exemple: La instrucció LD Rd, -Y realitza una càrrega indirecte amb el registre Y pre-decrementat sobre Rd. Les accions que duu a terme són, així, dues:

Primer decrementa Y: Y ← Y-1.

Després carrega Rd: Rd ← dRAM<Y>

La figura 6.3 mostra un esquema de la indirecció que es duu a terme amb aquesta instrucció.

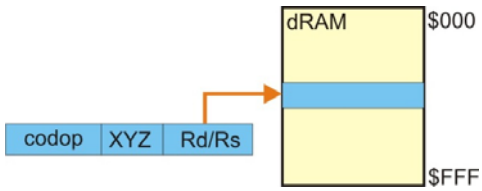


Figura 6.3. Adreçament indirecte

EduP12 també admet un adreçament indirecte a memòria de programa emprant el registre Z que el duu a terme amb les instruccions IJMP i ICALL.

6.2.6 Adreçament relatiu

En EduP12 es poden considerar dos tipus d'adreçament relatiu:

- Adreçament indirecte amb desplaçament. Es suma un desplaçament prefixat a una adreça de referència continguda en un registre (el Y o el Z). És el cas de la instrucció LDD Rd,Y+q, que carrega al registre Rd la dada continguda en la posició Y+q de la memòria de dades (figura 6.4).

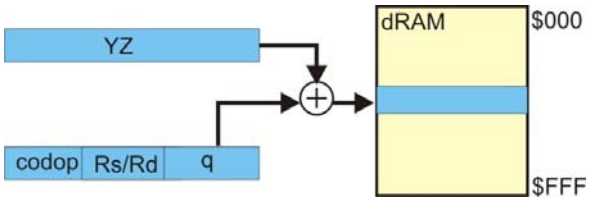


Figura 6.4. Adreçament indirecte relatiu

- Adreçament relatiu a comptador de programes, que es produeix en el cas de les instruccions de salt relatiu. Per exemple, en *RJMP despl*, s'efectua un salt a l'adreça de memòria de programa PC+despl (figura 6.5).

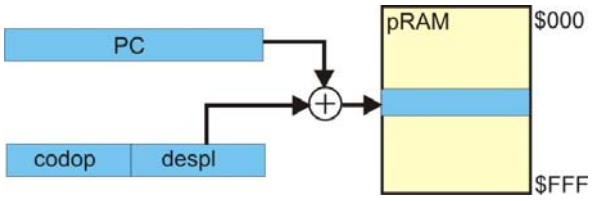


Figura 6.5. Adreçament relatiu a comptador de programes

6.3 Repertori d'instruccions en EduP12

El repertori d'instruccions d' EduP12 es pot classificar en els següents tipus d'instruccions:

- Instruccions aritmètico-lògiques
  - De doble registre: MOV, ADD, ADC, SUB, SBC, CP, CPC, AND, OR, EOR, TST.
  - De registre simple: INC, DEC, COM, NEG, CLR, ASR, LSR, LSL, ROR, ROL, SWAP.
  - Amb immediats: LDI, ADDI, ANDI, ORI, SUBI, SBCI, CPI, TSTI.
- Instruccions de transferència de dades.
  - Amb memòria de programa.
    - Instruccions de càrrega: LPM, LPM+
  - Amb memòria de dades.
    - Instruccions de càrrega: LDS, LD, LD+, LD-, LDD
    - Instruccions de guarda: STS, ST, ST+, ST-.
- Instruccions de salt.
  - De salt incondicional: RJMP, RCALL, RET, RETI.
  - De salt incondicional indirecte: IJMP, ICALL.
  - De salt condicional. BRBC, BRBS, BRCC, BRCS, BRSH, BRLO, BRZE, BRNZ, BRMI, BRPL, BRVC, BRVS, BRIE, BRID.
- Instruccions d'entrada/sortida: IN, OUT
- Altres.
  - De pila o *stack*: POP, PUSH, SAVE, RESTORE.
  - No fer res: NOP.

A continuació es fa un repàs a les instruccions de eduP12 per tipologia d'instrucció. La codificació particular de cada instrucció es pot trobar en l'apèndix A.

6.3.1 Instruccions aritmètico-lògiques de doble registre

Conjunt d'instruccions

El conjunt d'instruccions aritmètico-lògiques de dos registres inclou el conjunt d'instruccions següent:

- Aritmètiques: ADD, ADC, SUB, SBC,
- Lògiques: AND, OR, EOR
- De comparació de valors: CP, CPC, TST
- De transferència entre registres: MOV

Aquestes instruccions actualitzen el registre d'estat.

Format de les instruccions

El format de les instruccions és del tipus ADD Rd, Rs, que correspon a l'operació Rd ← Rd + Rs.

El codi màquina de la instrucció és:

cccc	ccsd	dddd	ssss

En el que

- cccccc correspon a la codificació de les instruccions, que va de 000001 a 001011
- ddddd representa el registre destí, de R0 a R31
- sssss representa el registre font, de R0 a R31

Exemple

ADD R16, R17 queda codificat com 0000 1011 0000 0001 = 0x0B01

6.3.2 Instruccions aritmètico-lògiques de registre simple

Conjunt d'instruccions

Les instruccions aritmètico-lògiques de registre simple treballen amb la dada d'un únic registre. Està format pel conjunt d'instruccions següent:

- Aritmètiques: INC, DEC, NEG.
- Lògiques: COM.
- De desplaçament: ASR, LSR, ROR, ROL, SWAP.

El desplaçament lògic a l'esquerra (LSL) s'implementa via ADD Rd, Rd.

Aquestes instruccions actualitzen el registre d'estat.

Format de les instruccions

El format de les instruccions és del tipus ADDI Rd, k, que correspon a l'operació Rd ← Rd + k.

El codi màquina de la instrucció es composa de dues paraules, i té el format[6]:

cccc	--wd	dddd	1ccc

I a on:

- cccc-ccc correspon a la codificació de les instruccions, que va de 0011-000 a 0011-111
- ddddd representa el registre destí, de R0 a R31
- kkkk kkkk kkkk kkkk representa la constant amb la que es treballa
- Totes les instruccions tenen el bit w = 0, excepte SWAP que val 1.

Exemple

ADDI R16, 0xF0F queda codificat com

0011 0001 0000 0001 | 0000 1111 0000 1111= 0x3101 | 0x0F0F

[6] Sempre que aparegui – en un bit indica que aquest valor no està emprat, tot i que en la descodificació s'agafa valor 0.

6.3.3 Instruccions aritmètico-lògiques amb immediats

Conjunt d'instruccions

El conjunt d'instruccions aritmètico-lògiques amb immediats conforma el conjunt d'instruccions que treballen amb constants, i inclou el conjunt d'instruccions següent:

- Aritmètiques: ADDI, SUBI, SBCI
- Lògiques: ANDI, ORI
- De comparació de valors: CPI, TSTI
- De càrrega: LDI

Aquestes instruccions, a excepció de LDI, actualitzen el registre d'estat.

Format de les instruccions

El format de les instruccions és del tipus ADDI Rd, k, que correspon a l'operació Rd ← Rd + k.

El codi màquina de la instrucció es composa de dues paraules, i té el format:

cccc	---d	dddd	0ccc	----	kkkk	kkkk	kkkk

I a on:

- cccc-ccc correspon a la codificació de les instruccions, que va de 0011-000 a 0011-111
- ddddd representa el registre destí, de R0 a R31
- kkkk kkkk kkkk kkkk representa la constant amb la que es treballa

Exemple

ADDI R16, 0xF0F queda codificat com

0011 0001 0000 0001 | 0000 1111 0000 1111= 0x3101 | 0x0F0F

6.3.4 Instruccions de transferència de dades amb memòria de programa.

La memòria de programes conté el programa que executa el processador.

A nivell de guarda de dades només es permet l'escriptura en el moment de càrrega del programa, pel que pot ser útil per emmagatzemar dades rellevants pel programa, especialment important si són taules de dades.

Per tant, amb la memòria de programa només es permet la lectura de dades.

Conjunt d'instruccions

La instrucció de transferència de dades (de memòria de programa a registre) és:

- Càrrega de programa a registre, només X, Y o Z: LPM Rd, XYZ
- Càrrega de programa a registre, només X, Y o Z amb pre-increment o pre-decrement: LPM Rd, +/-XYZ

Format de les instruccions

- El codi màquina de la instrucció és:

1100	ZYXd	dddd	11ab

a on:

- ddddd és el registre
- ZYX val 100, 010 o 001 segons s'empri Z, Y o X, respectivament
- ab és la codificació de la indirecció:

  -0 → Indirecció normal (sense pre-decrement ni pre-increment)

  01 → pre-increment

  11 → pre-decrement

Exemple

- LPM R17, -Z. Realitza l'operació

  ...primer decrementa Z: Z←Z-1

  ...i després mem<Z>←R20

  es codifica amb

  1100 1001 0001 1111

6.3.5 Instruccions de transferència de dades amb memòria de dades.

Conjunt d'instruccions

El conjunt d'instruccions de transferència de dades transporten les dades de memòria a registre i al revés.

Les instruccions que transfereixen dades de registre a memòria s'anomenen instruccions de guarda (*store*). Les que transfereixen dades de memòria a registre s'anomenen instruccions de càrrega (*load*).

Les instruccions de transferència de dades admeten diferents modes d'adreçament.

El conjunt d'instruccions de transferència de dades és el següent:

- Instruccions de transferència de dades directa.
    - S'adreça directament la posició de memòria
    - Necessita d'una segona paraula que conté l'adreça de memòria
    - Està format per les instruccions LDS i STS.
- Instruccions de transferència de dades indirecta.
    - La indirecció es realitza via un dels registres X, Y o Z, que corresponen als registres R29, R30 i R31, respectivament.
    - Els registres X, Y o Z contenen l'adreça (indirecció) a la que s'ha d'anar a cercar la dada.
    - Hi ha un triple format:
        - Indirecció normal
        - Indirecció amb pre-increment: primer incrementen el registre i després fan la cerca
        - Indirecció amb pre-decrement: primer decrementen el registre i després fan la cerca
    - El conjunt d'instruccions està format per
        - Indirecció normal: LD XYZ, ST XYZ, on XYZ fa referència a un dels tres registres
        - Indirecció amb pre-increment: LD +XYZ, ST +XYZ
        - Indirecció amb pre-decrement: LD -XYZ, ST -XYZ
- Instruccions de transferència de dades indirecta amb desplaçament
    - En el moment de fer la indirecció sumen una constant al registre.
    - Els dos registres permesos per fer la indirecció són Y i Z.
    - El conjunt d'instruccions és: LDD YZ+q, STD YZ+q.

Aquestes instruccions, a excepció de LDI, actualitzen el registre d'estat.

Format de les instruccions

El format de les instruccions de transferència de dades és el següent:

- Instruccions de transferència de dades directa.
    - LDS Rd, K. Efectua la càrrega Rd ← mem<K>

STS K, Rs. Efectua la guarda mem<K>← Rs, a on

Rd correspon a un registre

K és la posició de memòria de dades amb la que es treballa

- El codi màquina de la instrucció es composa de dues paraules, i té el format:

110p	---d	dddd	01--		----	KKKK	KKKK	KKKK

a on:

- p és 0 si és LDS i 1 si és STS
- ddddd és el registre
- K...K és la posició de memòria amb la que es treballa

- Instruccions de transferència de dades indirecta.
  - LD Rd, XYZ. Efectua la càrrega Rd ← mem<XYZ>

    ST XYZ, Rs. Efectua la guarda mem<XYZ>← Rs, a on

    Rd correspon a un registre

    XYZ és el registre amb el que es treballa: X, Y o Z

    Hi ha la possibilitat de pre-increment o pre-decrement del registre XYZ.
  - El codi màquina de la instrucció és:

110p	ZYXd	dddd	00ab

a on:

- p és 0 si és LDS i 1 si és STS
- ddddd és el registre
- ZYX val 100, 010 o 001 segons s'empri Z, Y o X, respectivament
- ab és la codificació de la indirecció:

  -0 → Indirecció normal (sense pre-decrement ni pre-increment)

  01 → pre-increment

  11 → pre-decrement

- Instruccions de transferència de dades indirecta amb desplaçament
  - LDD Rd, Y+q, LDD Rd, Z+q. Efectua la càrrega Rd ← mem<XYZ+q>

    STD Y+q, Rs, STD Z+q, Rs. Efectua la guarda mem<XYZ+q>← Rs, a on

    Rd correspon a un registre

    q és la constant de desplaçament

    YZ és el registre Y o Z que s'empra en la instrucció
  - El codi màquina de la instrucció és:

11(Y/Z)p	qqqd	dddd	qqqq

a on:

- p és 0 si és LDS i 1 si és STS
- Y/Z val 1 quan s'empra Y, 0 quan s'empra Z.

- ddddd és el registre.
- qqq qqqq és la constant de desplaçament, entre 0 i 127.

Exemples

- LDS R31, 0x842, que realitza la càrrega directa R31←mem<0x842> és codificat amb

  1100 0001 1111 0100 | 0000 1000 0100 0010

- ST +Y, R20, que realitza l'operació

  ...primer s'incrementa Y: Y←Y+1

  ...i després mem<Y>←R20

  es codifica amb

  1101 0101 0100 0001

- STD Z+65, R0, que realitza l'operació mem<Z+65>←R0 es codifica amb

  1101 1000 0000 0001

6.3.6 Instruccions de salt

Conjunt d'instruccions

Les instruccions de salt poden ser:

- **Incondicionals**. Efectuen un salt en l'ordre d'execució del programa sempre.

  El conjunt d'instruccions està format per:

  - RJMP k. Efectua un salt relatiu a k posicions respecte a la posició actual del comptador de programes
  - RCALL k. Guarda el contingut actual del comptador de programes en la pila i efectua un salt relatiu a una subrutina que es troba a k posicions respecte a la posició actual del comptador de programes.
  - RET. Efectua el retorn de subrutina. Per fer-ho va a cercar l'adreça de retorn de la pila.
  - RETI. És com RET però pel retorn de rutina de servei d'interrupció.

- **Incondicionals indirecte**. Són instruccions de salt incondicional en les que el desplaçament es troba en el registre Z. Està format per les instruccions:

  - IJMP. Efectua un salt relatiu, amb desplaçament donat en el registre Z, respecte a la posició actual del comptador de programes
  - ICALL. Guarda el contingut actual del comptador de programes en la pila i efectua un salt relatiu a una subrutina que es troba a Z posicions respecte a la posició actual del comptador de programes.

- **Condicionals**. Efectuen un salt quan es compleix una condició predeterminada sobre algun bit del registre d'estat.

  Segons el bit del registre d'estat estigui activat o no es té dues instruccions genèriques:

  - BRBS k. La instrucció ve de *Branch if Register Bit is Set*. És adir, saltar sobre bit activat.
  - BRBC k. La instrucció ve de *Branch if Register Bit is Clear*. És adir, saltar sobre bit no activat.

  La condició de bit activat o no es realitza sobre els bits del registre d'estat: carreteig (C o *carry*), zero (Z), negatiu (N), excés (V o *overflow*) i I (interrupció).

  Aquestes dues instruccions es divideixen en instruccions particulars que actuen sobre un bit concret:

  - BRCS k, BRCC k. Pregunten sobre si el bit de carreteig és 1 o no, respectivament.
  - BRLO k, BRSH k. Com en el cas anterior pregunten sobre si el bit de carreteig és 1 o no, respectivament.
  - BRZE (BREQ) k, BRNZ (BRNE) k. Pregunten sobre si el bit de zero és 1 o no, respectivament.
  - BRMI k, BRPL k. Pregunten sobre si el bit de nombre negatiu és 1 o no, respectivament.
  - BRVS k, BRVC k. Pregunten sobre si el bit d'excés és 1 o no, respectivament.
  - BRIE k, BRID k. Pregunten sobre si el bit d'interrupció està habilitat o no, respectivament.

Format de les instruccions

- El codi màquina de les **instruccions incondicionals** ve donat per:

010p	kkkk	kkkk	kkkk

  - p val 0 per la instrucció RJMP i 1 per la instrucció RCALL.
- Per **instruccions incondicionals amb adreçament indirecte** el codi màquina de la instrucció ve donat per:

0110	zyx-	----	001p

  - ZYX val 100, 010 o 001 segons s'empri Z, Y o X, respectivament
  - p val 0 per la instrucció IJMP i 1 per la instrucció ICALL.
- Per instruccions de salt condicional el codi màquina de la instrucció és:

  Per les instruccions de bit activat o BRBS és

1110	kkkk	kkkk	kbbb

  Per les instruccions de bit no activat o BRBC és

1111	kkkk	kkkk	kbbb

  a on:
  - La codificació d'instrucció de salt ocupa els 4 bits més significatius
  - K...k fa referència al salt relatiu, que va de +255 a 256 posicions de memòria
  - bbb és la codificació del bit de salt, codificat en la forma següent:

    0 → Condició sobre bit C

    1 → Condició sobre bit Z

    2 → Condició sobre bit N

    3 → Condició sobre bit V

    7 → Condició sobre bit I

Exemple

- BRNE 45 (que és equivalent a BRBC 1, 45). Realitza l'operació

  Si el bit Z val 0 aleshores salta a PC+45, si no continua en seqüència

  es codifica amb

  1111 0001 0110 1001
- BRMI -45 (que és equivalent a BRBS 2, -45). Realitza l'operació

  Si el bit Z val 0 aleshores salta a PC+45, si no continua en seqüència

  es codifica amb

  1110 1110 1001 1001

  Observar que -45 = 111010011 quan es codifica en complement a la base sobre 9 bits.

6.3.7 Instruccions d'entrada/sortida

Són instruccions de treball amb la memòria d'entrada/sortida

Conjunt d'instruccions

Hi ha dues instruccions:

- IN Rd, PORT. Realitza l'operació Rd←PORT.

  Comunica l'exterior amb el processador a través d'un port d'entrada.
- OUT PORT, Rs. Realitza l'operació PORT←Rs.

  Comunica el processador amb l'exterior a través d'un port de sortida.

Format de les instruccions

- El codi màquina de les instruccions és:

0111	pAAd	dddd	AAAA

  a on:
  - P val 0 quan la instrucció és IN, i 1 quan és OUT
  - ddddd és el registre de treball
  - AA-AAAA és l'adreça del port d'entrada/sortida. Val entre 0 i 63.

Exemple

- OUT 32,R0, treu pel port de sortida 32 el valor de R0 i es codifica com

  0111 1100 0000 0000
- IN 32, R0 guarda en R0 el valor del port d'entrada 32. Es codifica com

  0111 0100 0000 0000

6.3.8 Instruccions de pila (*stack)* i de guarda d'estat

Quan es treballa amb la pila cal recordar dos detalls:

- L'operació sempre s'executa amb la posició de memòria a la que apunta l'apuntador de pila. En l'operació de PUSH primer es posa la dada i després s'incrementa l'apuntador de pila. Quan s'executa POP primer es decrementa l'apuntador i després es treu la dada.
- S'ha d'anar en compta en emprar la pila quan es treballa amb subrutines i interrupcions ja que s'empra la pila per guardar dades de retorn i dades temporals.

Conjunt d'instruccions

Hi ha dues instruccions que permeten treballar amb la pila:

- PUSH Rd. Posa el valor del registre Rd en la pila.
- POP Rs. Recupera el valor de la pila i el posa en el registre Rs.

Apart n'hi ha dues més que empren la pila per guardar i recuperar l'estat (compta que només fan referència als bits V, N, Z i C i no al bit I d'establiment d'interrupció):

- SAVE. Posa el valor del registre d'estat en la pila.
- RESTORE. Recupera el valor de la pila i el retorna en el registre d'estat.

Format de les instruccions

- El codi màquina de la instrucció és:

0110	e00d	dddd	011p

a on:

  - ddddd és el registre de treball
  - p val 1 per posar en pila (PUSH i SAVE) i 0 per treure de pila (POP i RESTORE).
  - e val 1 en les instruccions SAVE i RESTORE

Exemple

- PUSH R0 es codifica amb

  0110 0000 0000 0111

- POP R0 es codifica amb

  0110 0000 0000 0110

- SAVE es codifica amb

  0110 1000 0000 0111

- RESTORE es codifica amb

  0110 1000 0000 0110

6.3.9 No fer res

Finalment queda la instrucció de no fer res o NOP.

S'empra per fer rutines d'espera.

Format de la instrucció

- El codi màquina de la instrucció és:

0000	0000	0000	0000

Exemple

NOP

6.4 Resum del capítol

El capítol ha introduït el repertori d'instruccions que s'executen en EduP12 i ha analitzat en detall els diferents tipus d'instruccions que el configuren, especificant en cada instrucció el conjunt de camps que la configuren.

Associat amb cada instrucció que treballa amb dades hi ha l'adreçament. S'ha vist que EduP12 admet un conjunt important de modes d'adreçament i s'han detallat en cada instrucció.

Els apèndixs del llibre complementen la informació donada del processador EduP12 en els capítols 5 i 6.

- La taula de l'apèndix A1 condensa el repertori d'instruccions complert del processador EduP12, indicant el nombre de cicles de rellotge que necessita cada instrucció, la codificació, el nombre de paraules que requereix i els bits del registre d'estat que queden afectats per la seva execució
- L'apèndix A2 és la guia d'usuari de les instruccions. Desglossa per totes les d'instrucció la transferència de dades que s'estableix en la CPU quan s'executa la instrucció.

6.5 Exercicis

1. Donats els següents programes, donar el valor del registre d'estat durant l'execució del programa.

.con tot = 0xfff;
.con res = 0x000;
.def a = r31;
.def b = r30;
 	
ldi a, res;
ldi b, res;
tst a, b;
and a, b;
sub a, b;
	.con tot = 0xfff;
	.con res = 0x000;
	.def a = r31;
	.def b = r30;
	
	ldi a, tot;
	ldi b, tot;
	tst a, b;
	and a, b;
	sub a, b;
		.con tot = 0xfff;
		.con res = 0x000;
		.def a = r31;
		.def b = r30;
		
		ldi a, tot;
		ldi b, res;
		tst a, b;
		and a, b;
		sub a, b;
			.con tot = 0xfff;
			.con res = 0x000;
			.def a = r31;
			.def b = r30;
			
			ldi a, res;
			ldi b, tot;
			tst a, b;
			and a, b;
			sub a, b;

2. Donats els següents dos programes es demana:

i) Donar el diagrama de flux del programa.
ii) Trobar el codi màquina del programa.
iii) Per a cada instrucció, indicar quin mode d'adreçament s'empra.

El nombre entre parèntesis al començament d'una instrucció indica la posició en la que comença aquella instrucció.

PROGRAMA 1. Realitzar la suma aritmètica dels 50 primers nombres i donar el valor pel PORTB[7]. En aquests programes és important entendre bé els límits dels índexs per acabar just en el nombre límit que es demana.

```
-- SUMA ARITMÈTICA
(0)     LDI R2,
        0
        MOV R1, R2
loop:   INC R1
        ADD R2, R1
        CPI R1, 50
        BRMI loop (-5)
        OUT PORTB, R2
fi:     RJMP fi (-1)
        NOP
```

PROGRAMA 2: Càlcul del sinus per detecció de valor en taula

```
--Càlcul tabular del sinus
-- La taula del sinus es guarda a partir de la posició 1000, i salta de 10º en 10º
-- Per evitar fraccionaris el valor del sinus es multiplica per 100 i s'agafa l'enter més pròxim
-- Nota: Per simplicitat es mostren els primers i darrers valors de la taula. Acabeu d'omplir-la.
        LDI Z, 0x999
loop:   CPI Z, 1035
        BRPL fi (+3)
        LPM R0, +Z
        OUT PORTB, R0
        RJMP loop (-6)
fi:     RJMP fi (-1)
        NOP

(1000)  "0000000000000000"      sin(0)
        "0000000000010001"      sin(10)
        "0000000000100010"      sin(20)
        ...
        "1111111111101111"      sin(350)
```

3. Explicar si és coherent i què fa el següent programa:

```
--Programa exemple amb ijmp
.con inici = 10;
.con fi = 20;

.org 0;
```

[7] La instrucció IN, amb format *IN Registre, PORTA* és una instrucció d'entrada que permet agafar dades d'un port extern a la CPU. Respectivament, La instrucció OUT, amb format *OUT PORTB, Registre* és una instrucció de sortida que envia dades cap a un port extern a la CPU. En aquests exemples es fan servir com a ports d'entrada i sortida de dades. En el capítol 8 s'introdueix de manera més formal el treball amb ports d'entrada/sortida.

```
        ldi x, inici;
        ldi z, fi;
    .org inici;
        out PORTB, x;
        ijmp z;

    .org fi;
        out PORTB, z;
        ijmp x;
```

4. Pel PortA s'entren 10 nombres. Treure pel PortB el nombre major.

5. Realitzar un programa que doni la suma dels 25 primers múltiples de 3. Donar el resultat pel PortB.

6. Donat un nombre de 12 bits (entrat pel PortA) treure 1 pel PortB si és múltiple de 100, i si no treure 0.

7. (Problema una mica més llarg). Donat dos nombres A i B de 12 bits (es poden entrar com a immediats en el programa), realitzar l'operació la seva multiplicació i guardar el resultat en els registres R1:R0.

Capítol 7

INTERRUPCIONS EN EDUP12

El capítol introdueix el concepte d'interrupció que consisteix en l'acceptació d'events per part del processador que alteren la seqüència normal d'execució d'un programa.

En el primer apartat s'exposa el concepte d'interrupcions i els diferents tipus que poden existir. Posteriorment es passa a veure l'ús d'interrupcions en EduP12 i el tractament que es fa de les mateixes. Seguidament s'explica com s'han de considerar dintre el programa les interrupcions.

7.1 El concepte d'interrupció

La interrupció és un senyal que indica al processador un salt en l'execució normal d'instruccions que està executant. La interrupció indica al processador que ha de prestar atenció a un event que s'ha produït i que per tant ha d'alterar la seqüència de funcionament.

Per tal de prestar atenció a la interrupció el processador passa a executar una rutina de servei de tractament de la interrupció. Com que després de l'execució de la rutina de servei ha de tornar al punt de programa on es trobava, previ al salt a la rutina de servei d'interrupció el processador guarda l'estat en el que es troba.

La interrupció és una tècnica molt comuna de multi tasca en els processadors molt emprada en processadors que han de treballar amb temps real. En aquestes aplicacions durant l'execució d'un programa és normal que hi interfereixin events externs que han de ser tractats. Se'n encarrega la interrupció. Per exemple, l'execució d'un programa que depèn de senyals externs es podria dur a terme creant un bucle d'espera mentre no arriben aquests senyals. Però és molt més pràctic deixar al programa fent tasques de procés i només tractar el senyal extern quan aquest interromp el processador.

En un processador es poden produir diferents tipus d'interrupcions. Poden ser externes, degudes a events que es produeixen fora del processador, o internes, degudes, per exemple, a excepcions de procés com podria ser el cas d'una divisió per zero.

En qualsevol cas una interrupció ha de portar sempre el processador a un estat precís que permeti no perdre la seqüència d'execució normal del programa.

El tractament d'una interrupció de forma precisa compleix quatre propietats:

- En esdevenir una interrupció es guarda el comptador de programa en lloc segur.
- La instrucció que s'està executant s'acaba d'executar. No es parteix cap instrucció.
- En guardar el comptador de programes no s'inicia cap instrucció posterior a la que s'estava executant.
- L'execució de la rutina de servei ha de portar a un estat (adreça) conegut.

En el cas del processador EduP12 el comptador de programes es guarda en la pila, que s'inicia en la darrera posició de la memòria de dades.

La subrutina que s'executa quan s'atén una interrupció s'anomena *rutina de servei de la interrupció* o *ISR*.

EduP12 disposa de dues instruccions particulars per tractar amb interrupcions:

- La instrucció SEI (*Set Interrupts*) activa el senyal (*flag*) general d'interrupcions. Amb aquest *flag* activat s'entén que el programa accepta una interrupció quan aquesta es produeix.
- La instrucció CLI (*Clear Interrupts*) inactiva el flag general d'interrupcions, fet que impedeix l'acceptació de cap interrupció per part de la CPU.

El flag general d'interrupció és el bit *I* del registre d'estat.

7.2 El mòdul d'interrupció en EduP12.

La figura 7.1 mostra el mòdul d'interrupcions implementat en EduP12. És un mòdul parametritzable amb una variable genèrica n que admet 2^n-1 interrupcions, enumerades des de 1 fins a 2^n-1, respectivament.

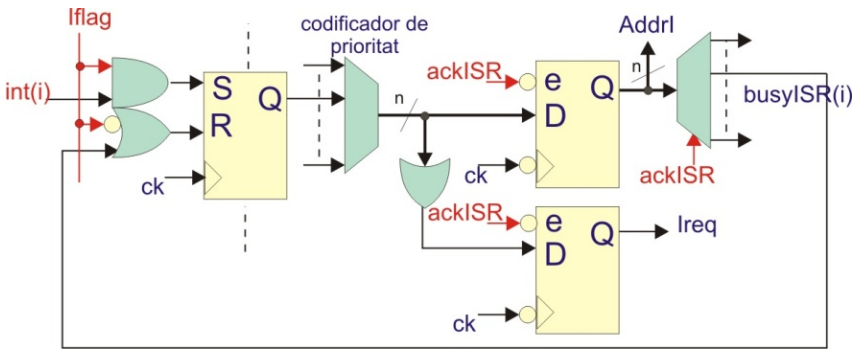


Figura 7.1 El mòdul d'interrupcions

El funcionament del mòdul és simple:

- El senyal del registre d'estat *Iflag* (bit I del registre d'estat) controla l'habilitació d'interrupcions.
- Amb *Iflag* habilitat el mòdul està en estat d'espera fins que arriba una interrupció. Tota interrupció posa en estat habilitat el biestable SR (i).
- Un codificador de prioritat prepara el flip-flops de requeriment d'interrupció i registre d'adreça d'interrupció. El reconeixement d'acceptació d'interrupció per part de la unitat de

control (amb el senyal *ackISR*) clava el mòdul d'interrupcions inhabilitant l'activació de nous requeriments d'interrupció.

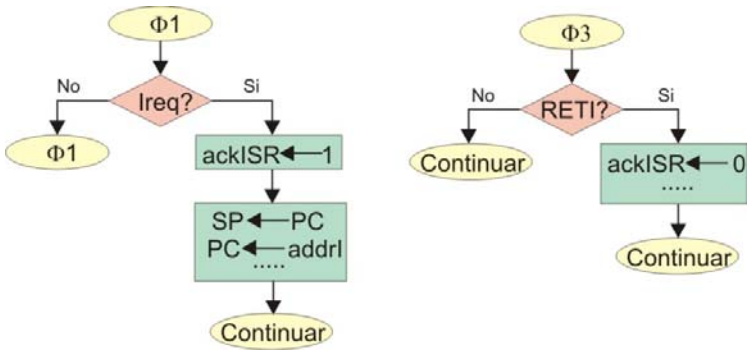
- El senyal *busyISR(i)* permet que es puguin acceptar noves interrupcions però evita el servei d'aquestes així com l'acceptació per part de la interrupció que s'està servint.

7.3 La interrupció en EduP12

Les interrupcions acceptades per la CPU són peticions de servei per part de perifèrics acoblats a la CPU. El mòdul d'interrupció queda acoblat en EduP12 i la unitat de control és l'encarregada de controlar la petició de servei i el servei de les interrupcions.

El funcionament de les interrupcions en EduP12 és el següent (figura 7.2):

- En entrar en la fase de descodificació, i en el cicle inicial, la UC pregunta pel senyal *Ireq*.
- Davant d'una activació d' *Ireq* per part d'una interrupció la unitat de control passa a determinar la rutina de servei d'interrupció que s'ha d'executar. L'adreça proporcionada pel mòdul d'interrupcions, *AddrI* .es l'adreça de programa en la que es troba la referència a l'adreça de la rutina de servei de la interrupció.
- Davant l'activació d'*Ireq* la UC carrega *AddrI* al comptador de programes per anar a cercar la rutina de servei d'interrupció. Al mateix temps ha de guardar l'adreça actual del comptador de programes en la pila ja que és l'adreça de retorn. També activa el senyal *ackISR* de reconeixement de servei d'interrupció. De fet, s'actua com en una instrucció de salt a subrutina.
- *Addri* adreça la posició de memòria on es troba l'adreça a la que s'ha d'anar a executar la rutina de servei de la interrupció.
- Mentre dura el servei de la interrupció *ackISR* es manté activat per evitar el servei de cap altra interrupció.
- La instrucció RETI és la instrucció de retorn de rutina de servei d'interrupció. Apart de les mateixes funcions de RET també desactiva *ackISR*.



a) **b)**

Figura 7.2 El mòdul d'interrupcions. a) Quan es produeix una petició d'interrupció i està habilitat el flag general d'interrupció en la fase 1 d'execució de la interrupció es detecta i s'entra en la rutina de servei d'interrupció. b) La detecció de la instrucció RETI fa sortir el programa de la rutina de servei de la interrupció.

7.4 La *ISR* en EduP12

Les interrupcions acceptades per la CPU són peticions de servei per part de perifèrics acoblats a la CPU. El mòdul d'interrupció queda acoblat en EduP12 i la unitat de control és l'encarregada de controlar la petició de servei i el servei de les interrupcions.

7.4.1 Nomenclatura.

Per tal d'explicitar millor l'execució de les instruccions en EduP12, la instrucció es posa sota el següent format:

(adreça de memòria, "codi màquina"), -- Codi hex instrucció – Instrucció en assemblador

D'aquesta manera, si s'agafa la primera instrucció de la figura 7.3, es té que:

- L'adreça de memòria de programa en la que es carrega la instrucció és la 0.
- El codi màquina de la instrucció ve donat per 0100000011110110
- Que correspon a la codificació en hexadecimal 0x401F
- I que llur instrucció assemblador és: RJMP +31. Es pot observar que +31 no és res més que la codificació del salt relatiu que la instrucció ha de fer a la posició de memòria 32 que és on comença la part de programa *main*.
- Apart, el darrera sovint s'hi posen comentaris per fer més llegible el codi.
- Com a nota final, indicar que tot caràcter posterior a -- en una línia correspon a comentaris (tal com s'entén en VHDL).

Aquest format que s'empra per representar d'ara endavant per a les instruccions ve determinat per la forma com s'introdueix el programa dintre un fitxer de codi VHDL quan es sintetitza la màquina sota FPGA. Es comenta en més detall en el capítol 10.

7.4.2 La ISR.

L'ús d'interrupcions en EduP12 imposa establir un mètode de treball en la construcció del programa en assemblador en el que la màquina pugui detectar fàcilment on es troben les rutines de servei d'interrupció quan aquestes es produeixen.

Per altra part, es fa difícil establir posicions concretes de les rutines de servei d'interrupció per què, a priori, no es pot conèixer quina mida tindrà cadascuna d'aquestes subrutines.

Per aquest motiu, la pròpia construcció d'EduP12 i la pròpia acceptació d'interrupcions en el mòdul d'interrupció (figura 7.1) fixa una adreça per a cada interrupció en la que s'espera s'hi introdueixi l'adreça real en la que es troba l'inici de la rutina de servei de la interrupció. Aquestes adreces corresponen a les adreces més baixes de la memòria, començant per què en l'adreça 0 s'espera que hi hagi el salt a la rutina principal del programa (adreça d'inici del programa). Així, per exemple, si es sintetitza un mòdul d'interrupció que admet fins a 16 rutines de servei d'interrupció (tal com està programat actualment EduP12), en carregar un programa que empri interrupcions en EduP12 s'han de reservar les 16 primeres adreces (de la 0 fins a la 15) per a possibles interrupcions que es puguin introduir, on l'adreça 0 correspon al salt al programa principal.

Les especificacions amb les que ve inicialment EduP12 estableix les adreces de rutina de servei d'interrupcions especificades en l'apèndix A3, que estan d'acord amb les adreces d'interrupció inicialment programades en la màquina. És a dir, tal com es serveix EduP12 s'han introduït inicialment les següents interrupcions (taula 7.1):

# d'interrupció	Adreça de programa d'ISR	Comentari
0	Programa principal	Salt a l'inici del programa
1	iEXT0DOWN	Interrupció externa 0 per flanc de baixada
2	iEXT0UP	Interrupció externa 0 per flanc de pujada
3	iEXT1DOWN	Interrupció externa 1 per flanc de baixada
4	iEXT1UP	Interrupció externa 1 per flanc de pujada
5	iEXT2DOWN	Interrupció externa 2 per flanc de baixada
6	iEXT2UP	Interrupció externa 2 per flanc de pujada
7	iEXT3DOWN	Interrupció externa 3 per flanc de baixada
8	iEXT3UP	Interrupció externa 3 per flanc de pujada
9	iTIM0OVR	Interrupció per excés del timer0
10	iTIM0CMPA	Interrupció del comparador A del timer0
11	iTIM0CMPB	Interrupció del comparador B del timer0
12	-	No usada
13	-	No usada
14	iTxD	Interrupció de fi de transmissió
15	iRxD	Interrupció de fi de recepció

Taula 7.1. Adreces de rutina de servei d'interrupció especificades en EduP12

D'acord amb aquestes especificacions, es recomana que tot programa principal comenci a partir de la posició de memòria 17.

7.4.3 Exemple de construcció d'un programa amb interrupcions

La figura 7.3 mostra la construcció típica d'un programa que empra interrupcions en EduP12. En la posició 0 del programa s'hi posa un salt a la posició d'inici del programa principal. En les posicions següents de programa s'hi posen les adreces on s'inicien les rutines de servei de les diferents interrupcions. A excepció d'aquestes restriccions, la resta de programa és lliure de posar-se en les posicions de memòria que es cregui més convenient.

En el cas de l'exemple de la figura 3 es pot observar que el programa principal comença en la posició de memòria 32. També s'ha previst el servei d'interrupció provocat per un *timer*. L'adreça reservada per posar la rutina de servei d'interrupció del *timer* és la 9. En aquesta posició s'hi ha de posar l'adreça d'inici de la rutina de servei per excés (*ovf*) del *timer*. Es pot observar que aquesta adreça és la 256.

```
--Adreces de rutines de servei d'interrupció
(   0, "0100000000011111"),     -- 0x401F - RJMP main (+31)
                ......................
(   9, "0100000011110110"),     -- 0x410A - RJMP isrTim0Ovf (+246)  → a 256
                ......................
--Programa principal
(  32, "0111000111010000"),     -- 0x71DA - main: IN X, PORTA
                ......................
(  39, "0000000100000111"),     -- 0x0107 - SEI (BSET 0x7)
(  40, "0000000000000000"),     -- 0x0000 - test: NOP
(  41, "0100111111111110"),     -- 0x4FF5 - RJMP test (-2)
(  42, "0000000000000000"),     -- 0x0000 - NOP
--Rutina d'interrupció isrTim0Ovf
(  256, "0110100000000111"),    -- 0x6807 - SAVE
```

```
( 257, "0011000000001000"),   -- 0x3008 - INC R0
( 258, "0111100000000001"),   -- 0x710B - OUT PORTB, R0
( 259, "0110100000000110"),   -- 0x6806 - RESTORE
( 260, "0110010100000101"),   -- 0x6505 - continuar: RETI
( 261, "0000000000000000"),   -- 0x0000 - NOP
```

Figura 7.3. Programa exemple d'ús d'interrupcions en EduP12.

En el programa de la figura 7.3 es pot observar que el programa principal no fa res. És senzillament una rutina d'espera. Aquest fet no és rar en programes molt dedicats a interrupcions. El programa queda esperant una interrupció entre les instruccions 40 i 41. En produir-se una interrupció per excés del timer la unitat de control passa a executar la instrucció 9 que conté la referència a la rutina de servei. I la instrucció 9 salta a executar la rutina de servei de la interrupció que es troba en la posició 256.

Es pot observar l'ús de les instruccions SAVE i RESTORE en la rutina de servei de la interrupció. S'usen per guardar l'estat del processador durant l'execució de la rutina.

7.4 Resum del capítol

El capítol ha introduït el concepte d'interrupció i ha detallat el seu ús en Edup12. S'ha vist també com es reserven les posicions de salt a rutina de servei d'interrupció quan aquestes s'usen i se'n ha detallat el seu ús.

Encara que no s'emprin interrupcions és bona pràctica posar a la posició 0 de memòria un salt d'inici del programa principal que es trobi a partir de la posició 32 de memòria per què això deixa lliures les posicions de memòria més baixes per si en un futur es volen introduir l'ús d'interrupcions en el programa.

Cal recordar que quan s'usen interrupcions convé tenir molt present que en entrar a la rutina de servei d'interrupcions cal guardar l'estat i que cal recuperar-lo en sortir. Això es deu a què tota instrucció aritmètica actualitza el registre d'estat. Donat que el salt a la rutina de servei d'interrupció es pot produir en qualsevol moment, no observar aquest fet pot modificar el comportament del programa principal si el registre d'estat s'actualitza en la rutina de servei d'interrupció.

EduP12 es subministra amb un conjunt de perifèrics (capítol 8) que admeten interrupcions. Lo bonic d'aquest processador és que queda obert per modificar aquesta personalització amb la possibilitat de modificar les interrupcions i estendre-les si així es desitja. El capítol 10 dóna les bases per a fer-ho.

7.5 Plantejament d'exercicis

El plantejament d'exercicis en aquest capítol va una mica més enllà dels simples exercicis de classe, donat que l'ús d'interrupcions implica l'ús de components d'entrada/sortida que es veuen en el proper capítol.

Tot i així, es poden enunciar alguns exercicis elegants per veure com EduP12 treballa amb interrupcions.

Exercici d'entrada/sortida

Un simple exercici és realitzar un programa que enviï als 8 leds de què disposa la placa Spartan3 de Digilent l'estat en el que es troben els seus 8 interruptors. La placa Spartan-3 de Digilent és una placa simple educativa basada en una FPGA de Xilinx en la que es pot

descarregar EduP12. Es recomana consultar les seves prestacions ja sigui mirant l'apartat 10.5 del llibre o, millor encara, el *datasheet* de la placa (recomanat en la bibliografia).

Aquest exercici es presenta en la figura 8.7 del llibre en la forma d'un exercici molt simple i fàcil de fer. El programa està en un bucle d'espera sense fer res a l'espera que en el polsador 1 de la placa es produeixi un flanc de baixada. Aquest canvi és detectat per la rutina de servei d'interrupció habilitada sobre aquest botó. En executar-se la rutina de servei corresponent, el programa agafa la dada dels 8 interruptors que conté la placa i la mostra en els 8 leds.

Rutina d'espera amb el timer

Fins ara s'ha vist que una rutina d'espera es pot fer programant un bucle d'espera en el que el programa es queda enganxat esperant sortir del bucle. El problema és que en aquest cas la CPU queda enganxada i no pot executar cap més altra acció fins que el programa en surt.

Una forma de crear temps d'espera, mentre s'alliberi la CPU per fer altres tasques, és programant el timer amb el temps desitjat i fent que, cada cop que es produeixi excés del comptador, es produeixi la interrupció per excés del comptador. Aleshores, en la rutina de tractament de la interrupció del timer per excés es fan les accions corresponents.

Generació de PWM

Un senyal periòdic amb modulació de l'amplitud de pols (PWM) és fàcil generar-lo amb el timer que incorpora EduP12. Programant la interrupció iTIM0CMPA (o iTIM0CMPB) es fàcil generar aquest pols de manera que la CPU pot treballar desentenent-se de la seva generació.

7.5 Plantejament d’exercicis

Capítol 8

ENTRADA/SORTIDA EN EDUP12

8.1 Introducció.

EduP12 configura un processador programable que per les seves característiques d'estar descrit com a nucli programat (*soft-core* o *Intellectual Property* – IP) té una llibertat gran de configuració de la seva entrada/sortida, permetent la inclusió de múltiples i diferents perifèrics.

Tot i això, i amb l'objectiu de poder facilitar l'aprenentatge de fonaments de computadors amb eines de simulació predeterminats, s'ha personalitzat un conjunt de components d'entrada/sortida que s'acoblen a la CPU introduïda en els capítols anteriors.

Aquesta personalització inicial contempla els següents components:

- Port d'entrada de 12 bits.
- Port de sortida de 12 bits.
- Port d'entrada de 4 bits amb interrupció per flanc de baixada i/o pujada.
- Un port UART d'emissió i recepció amb interrupció.
- Comptador (*timer*) de 12 bits que conté les següents característiques:
    - Comptador amb accés (de lectura i escriptura) a l'estat.
    - Prescaler de 12 bits
    - Dos ports, anomenats A i B, de comparació de 12 bits.
    - Interrupcions per overflow i per comparació.
- Port de sortida a 4 7-segments, muntats sobre bus de dades comú i bus de selecció de set-segment.
- També s'incorpora un port ioPort d'entrada/sortida, tot i que no està entrat explícitament en el processador per donar més simplicitat a aquesta personalització inicial.

Tot seguit es passa a especificar el connexionat genèric de ports amb el processador i es detallen els ports més importants emprats en aquesta implementació del processador.

8.1 Connexió de perifèrics en la CPU

Tots els components s'acoblen en el processador de forma registrada. El conjunt de tots els registres es veu des de la CPU com una memòria d'entrada/sortida.

Tots els registres de la memòria d'entrada/sortida tenen, doncs, una adreça predeterminada. Donada l'amplada de 6 bits del bus d'adreces per entrada/sortida, la CPU admet fins a 64 registres d'entrada/sortida.

La memòria, per defecte, sobreentén que les dades d'entrada (asíncrones respecte al processador) s'actualitzen en flanc de baixada, mentre que les de sortida (que provenen del processador) ho fan en flanc de pujada. Això permet evitar glitxs en el sincronisme entre processador i senyals externs.

En l'especificació del processador es realitza la connexió entre perifèrics i CPU. Bàsicament:

- Tot registre d'entrada/sortida té una única adreça.
- Tota lectura d'un port d'entrada realitza l'operació de transport de la dada del port a un registre a través del bus de dades. Per tant, tots els ports d'entrada són multiplexats per a poder ser llegides. Se'n encarrega la instrucció IN.
- Tota escriptura a un port de sortida implica que el port de sortida llegeix, a través del bus de dades, la dada d'un registre intern de la CPU. Se'n encarrega la instrucció OUT.
- El mòdul genèric de ports se'n encarrega del transport de dades entre CPU i els perifèrics.

L'entrada/sortida es veu, en EduP12, com una estructura jeràrquica composta d'un port, genèric, i els mòduls d'entrada/sortida inclosos en el port. El port és el mòdul arrel de l'estructura que es responsabilitza de la comunicació amb la CPU, comuna per a tots els perifèrics. Els mòduls d'entrada/sortida, que poden ser diversos i variables en número, s'inclouen en el port. Als mòduls d'entrada/sortida se'ls anomenarà perifèrics.

La figura 8.1 mostra l'esquema del cablejat establert entre el port i la CPU.

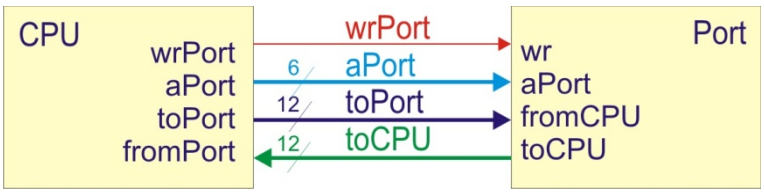


Figura 8.1. Senyals de comunicació entre CPU i el port d'entrada/sortida

Tota la comunicació entre CPU i port d'entrada i/o sortida passa pels senyals següents:

- wrPort. És el senyal que indica al port que es realitza una escriptura al port.
- aPort. És el bus d'adreces. Estableix el port de treball.
- dPort. És el bus de dades de port a CPU. En instrucció d'escriptura (OUT) s'escriu el registre del port adreçat per aPort. L'escriptura es realitza en el cicle que s'activa wrPort.
- qPort. És el bus d'entrada de dades de port a CPU. En instrucció de lectura (IN) es llegeix el registre del port adreçat per aPort.

Dintre el port s'hi connecten els perifèrics d'entrada i/o sortida. La figura 8.2 mostra un esquema genèric del cablejat intern. L'estructura s'entén quan s'analitza tenint present els tres tipus bàsics de perifèrics que poden conformar l'entrada/sortida:

- Perifèrics d'entrada
- Perifèrics de sortida
- Perifèrics d'entrada/sortida

Tots els altres perifèrics seguiran el mateix protocol per connectar al port.

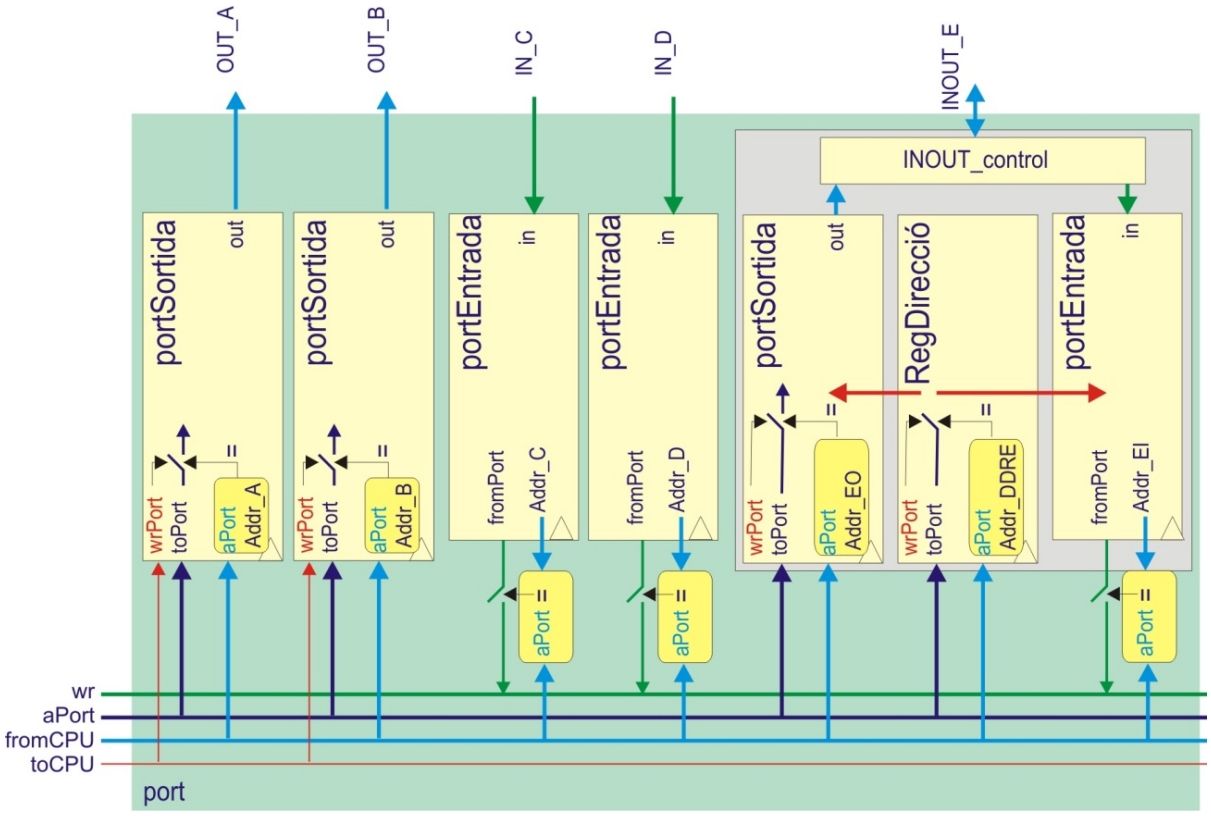


Figura 8.2. Estructura interna al port. Es mostra la connexió entre perifèrics d'entrada, de sortida i d'entrada/sortida que s'ajunten en el port. Qualsevol extensió amb altres perifèrics repeteix la mateixa estructura.

Per tant, el connexionat que s'estableix entre el processador i tot port d'entrada/sortida ha de contenir els senyals i seguir el protocol especificat en la figura 8.1. Qualsevol perifèric que ho compleixi es pot acoblar al processador. Qualsevol perifèric que no ho compleixi també es pot acoblar al processador si es crea una capa superior al perifèric que s'adapti al protocol exigit pel processador. A aquesta capa d'adaptació se l'anomena *wrapper*.

Moltes implementacions de processadors consideren un únic bus d'entrada/sortida. En aquesta implementació es consideren els busos de dades d'entrada i de sortida separats. Aquest fet evita senyals d'alta impedància interns, que són sovint fonts de soroll.

8.2 Perifèrics de sortida en el port

Donat que EduP12 està dissenyat pensant en què tots els senyals interns són d'entrada o de sortida, i que només es deixen senyals d'entrada/sortida exclusivament per connexions de busos amb

l'exterior, els perifèrics de sortida configuren la forma més simple de connexió d'un perifèric en EduP12.

La figura 8.3 mostra la connexió d'un perifèric de sortida en EduP12. Es realitza, senzillament, una extensió cap al perifèric del cablejat establert en la figura 8.1. Cada perifèric (de fet el registre que configura el perifèric) actualitza la seva dada (s'escriu) sempre que l'adreça del port coincideixi amb l'adreça del perifèric.

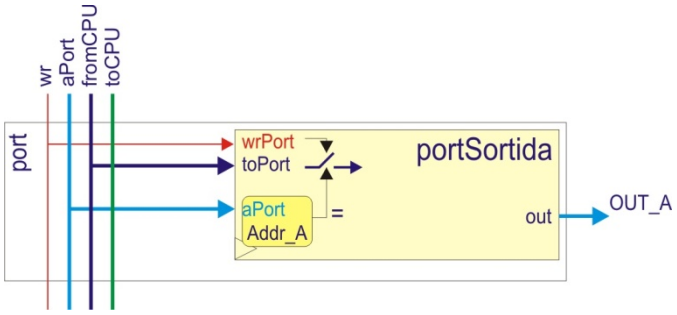


Figura 8.3. Connexió amb un perifèric de sortida.

En el cas de la figura 8.2, els ports A i B serien ports de sortida.

8.3 Perifèrics d'entrada

La connexió de ports d'entrada al sistema també és simple, tal com mostra la figura 8.4. La dada externa IN_A és mostrejada de forma contínua a la velocitat del port i guardada a cada cicle en el registre d'entrada. En el moment en què la CPU llegeix el port agafa el valor emmagatzemat i el transporta a un dels registres interns.

Donat que el processador evita senyals bidireccionals interns el port munta una estructura multiplexada per transportar el senya que es llegeix cap als registres. Aquesta estructura és interna al codi software i transparent a l'usuari a alt nivell, quan programa amb assemblador.

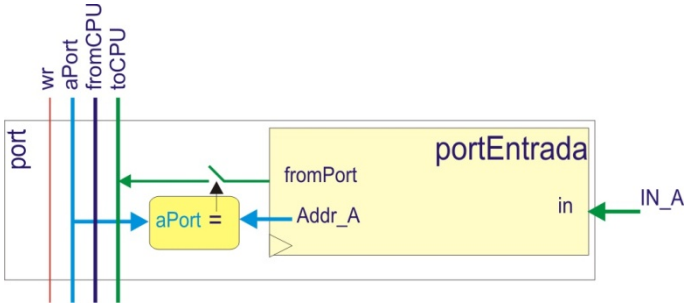


Figura 8.4. Port d'entrada

La figura 8.5 mostra un programa que empra un port d'entrada i un de sortida. En aquest cas es fa servir el port d'entrada per agafar les dades d'un interruptor de 8 bits i les envia cap al port de sortida que està connectat a 8 leds. El programa és simple i directe:

- S'agafa la dada pel port d'entrada amb una instrucció IN que especifica el port i el registre amb els que es treballa.
- I s'envia la dada del registre cap al port de sortida amb la comanda OUT.

- El programa queda en un bucle esperant repetir l'acció indefinidament.

La instrucció IN i la instrucció OUT comparteixen format. D'acord amb aquest format és fàcil veure que l'adreça del port d'entrada és la 000000 i la del port de sortida la 000001.

```
--IN/OUT: Senzillament porta a PORTB el que veu en PORTA
( 0, "0111000111010000"),      -- 0x71D0  - IN X, PORTA
( 1, "0111100111010001"),      -- 0x79D1 - OUT PORTB, X
( 2, "0100111111111101"),      -- 0x4FF9  - RJMP inp (-3)
( 3, "0000000000000000"),      -- 0x0000  - NOP
```

Figura 8.5. El programa envia cap al port de sortida la lectura contínua d'un port d'entrada

8.4 Perifèrics d'entrada/sortida

Els perifèrics d'entrada/sortida tenen una estructura més complexa ja que s'ha de controlar si el port actua com a entrada o com a sortida. Tot i que no està inclòs en la configuració per defecte de EduP12, s'analitza per què a vegades cal un port d'entrada/sortida.

El perifèric d'entrada/sortida mostrat en la figura 8.5 permet la programació com a entrada o sortida de cada element. Per poder fer-ho consta de tres registres:

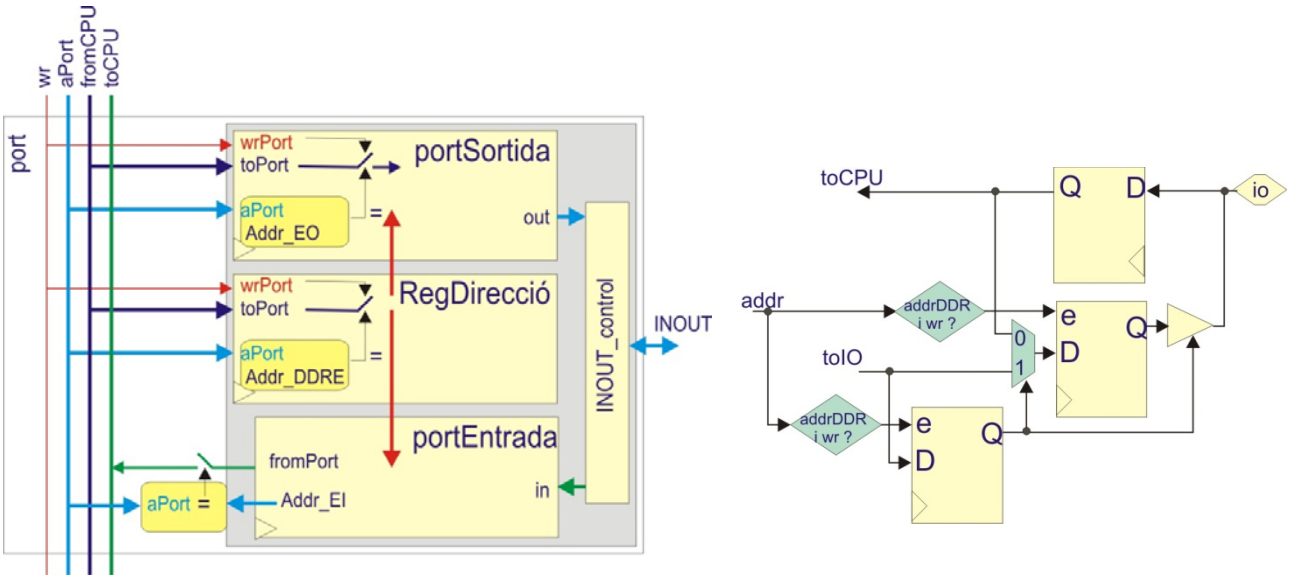


Figura 8.5. a) Connexió amb un perifèric de d'entrada/sortida. b) Connexió establerta en un pin d'entrada/sortida

- El registre d'entrada o DIN. Actua quan es fa servir el pin com a entrada. Si es fa servir el pin del port com a sortida agafa el valor del pin de sortida amb un cicle de retard. Des del punt de vista de la CPU és un registre de lectura.
- El registre de sortida o DOUT. Actua quan es fa servir el pin com a sortida. Si es fa servir el pin del port com a entrada agafa el valor del pin d'entrada amb un cicle de retard. Des del punt de vista de la CPU és un registre d'escriptura.

- El registre de control de direcció o DDR. És el pin de control de direcció de cada pin. És un registre d'escriptura.

La figura 3.5b mostra un esquema del cablejat d'un pin. Es pot observar la recirculació que s'estableix de la dada per garantir la unicitat del contingut entre els registres d'entrada i de sortida.

Com que aquest perifèric d'entrada/sortida conté tres registres, el perifèric necessita tres adreces. Com en el cas dels perifèrics de només entrada o només sortida, l'adreça és única i ve predeterminada.

8.5 Entrada amb interrupcions

A vegades interessa deslligar el funcionament del processador de rutines d'espera de dades que provenen d'entrades externes o de components propis interns per a la realització d'operacions molt específiques. A més, sovint passa que la transferència de dades que es realitza amb l'exterior pot arribar a ser molt més lenta que la velocitat de procés interna que es té, fet que sol implicar haver de crear bucles interns d'espera o *polling* per acoblar processador i perifèric. Aquest pot ser el cas d'una comunicació sèrie en la que es reben dades d'un perifèric que funciona a una velocitat de transferència (*baudrate*) baixa. També pot ser el cas d'un procés de dades d'un sensor llur sortida ve donada per modulació en amplada de pols i la qual s'ha de processar per conèixer-ne el valor. Quan això passa es lliga l'atenció del processador cap al perifèric a través d'interrupcions.

Per tal d'evitar cicles d'espera en la recepció de dades externes EduP12 incorpora el perifèric d'entrada *intEXT0* que és un port d'entrada amb interrupcions amb nombre de pins dedicats inicialment establert en 4. La figura 8.6 mostra la connexió que s'estableix amb el processador en el cas de *intEXT0*.

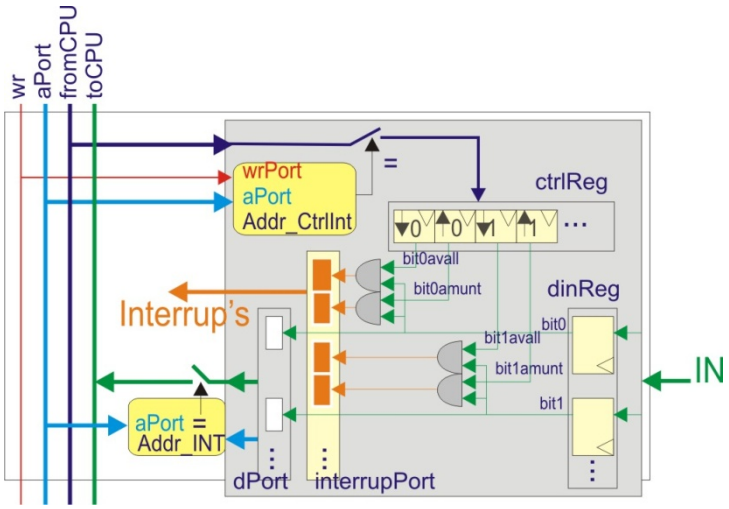


Figura 8.6. Connexió del processador amb el port d'entrada amb interrupcions PortDInt

L'estructura de *intEXT0* és molt simètrica:

- -Conté un registre de control que guarda la informació de les interrupcions que s'estableixen. Per a cada pin d'entrada el registre de control té dos bits que guarden informació sobre si s'habilita interrupció per flanc de pujada i/o flanc de baixada.
- Les entrades es guarden en un registre intern. El canvi en la sortida de cada bit del registre combinat amb la interrupció habilitada del bit activa la corresponent interrupció. Per tant, cada pin d'entrada pot activar fins a dues interrupcions (flanc de pujada i/o flanc de baixada).

Per evitar sensibilitat de la interrupció a glitxs de l'entrada el registre retarda fins a 4 cicles la dada d'entrada.

- El port de sortida *intEXT0* és un port normal de lectura del registre *dinReg* i s'acobla al bus que va cap al processador, igual que en els altres ports d'entrada/sortida vistos fins ara. El port *interrupt's* és un port que va directa cap al mòdul d'interrupció.

El programa de la figura 8.7 mostra un ús simple del port d'interrupcions externes.

```
--TREBALLANT AMB INTERRUPCIONS
--Senzillament treu pel PORTB el que veu en PORTA quan es deixa de prémer botó 1
--S'han suposat interrupcions externa en polsador 1 per flanc de baixada
--Adreça del programa principal
(  0, "0100000000011111"),    -- 0x401F - RJMP main (+31) → Està en posició 32
--Adreça de la rutina de servei de deixar de prémer botó 1
(  3, "0100000001111100"),    -- 0x407E - RJMP isrExtDw1 (+124)
(  4, "0000000000000000"),    -- 0x0000 - NOP
--Rutina principal
-- Inicialització interrupció externa del botó 1 per flanc negatiu
( 32, "0011000100000000"),    -- 0x3100 - main: LDI R16,       --Activar int. polsador 1 de baixada
( 33, "0000000000000100"),    -- 0x0x004
( 34, "0111100100000011"),    -- 0x7908 - OUT EXT0ctrl, R16
( 35, "0000000100000111"),    -- 0x0107 - SEI (BSET 0x7)
( 36, "0000000000000000"),    -- 0x0000 - bucle: NOP
( 37, "0100111111111110"),    -- 0x4FFE - RJMP bucle (-2)
( 38, "0000000100001111"),    -- 0x010F - CLI (BCLR 0x7)
( 39, "0000000000000000"),    -- 0x0000 - NOP
--Rutina d'interrupció
( 128, "0111000111011010"),   -- 0x71DA - isrExtDw1: IN X, PORTA
( 129, "0111100111010001"),   -- 0x79DB - OUT PORTB, X
( 130, "0110010100000101"),   -- 0x6505 - RETI
( 131, "0000000000000000"),   -- 0x0000 – NOP
```

Figura 8.7. Programa que mostra com es treballa amb el port extern amb interrupcions.

Nota al port *EXT0*:

El port EXT0 està predefinit com a un port d'entrada de 4 bits. Nogensmenys, donat que en la implementació en la placa Spartan3 es connecta a tres polsadors i a una entrada del port d'expansió, es pot fer servir com a un port d'entrada per a aquests dispositius i, aleshores, agafa el nom de PORTE, convertint-se (quan no s'empren interrupcions) en un port d'entrada de 4 bits.

Temporitzadors o *timers*.

Els timers són comptadors que solen venir inclosos en molts microcontroladors com a perifèrics amb diverses funcionalitats molt útils quan el processador treballa analitzant i actuant amb l'entorn. En concret:

- La funcionalitat del timer és paral·lela a la del processador. Funciona de manera autònoma. El processador pot actuar sobre el seu funcionament alterant-lo com també fer-lo servir com a recurs per actuar d'acord amb el seu estat.

Temporitzadors o timers.

- Els timers es configuren igual que els ports d'entrada/sortida. Tots els registres interns visibles han de poder ser escrits i/o llegits, depenent de quina sigui la seva funció.
- Poden implementar funcions tan bàsiques com la de només comptar, o tan complexes com la de generar senyals amb modulació d'amplitud i/o fase.

En el processador EduP12, donada la seva característica de codi obert, la llibertat per implementar timers és total, podent programar qualsevol funcionalitat addicional a la de simplement comptar. En qualsevol cas sempre han de complir la condició que per connectar al processador s'han d'acoblar al bus d'entrada/sortida emprat pels ports d'entrada/sortida.

La figura 8.8 mostra l'esquema d'un timer simple incorporat en el processador.

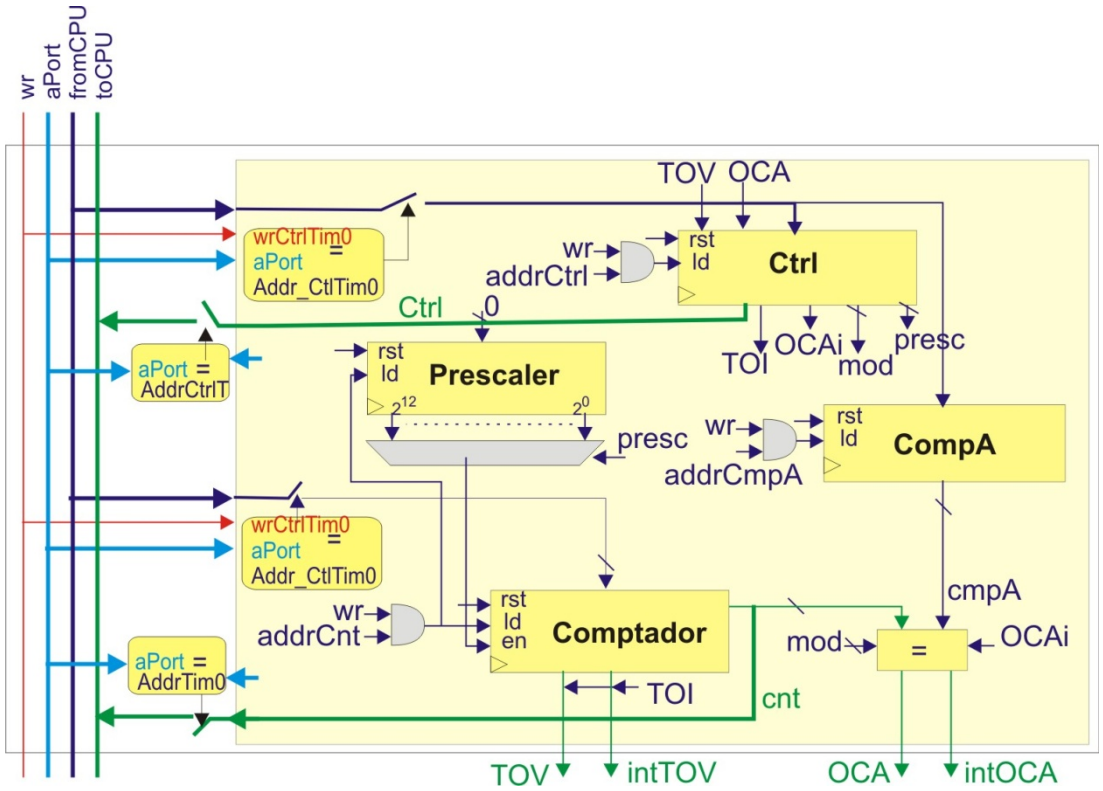


Figura 8.8. Timer amb sortida de comparació. Per clarificar el dibuix només s'ha posat un comparador

El timer consta dels següents components:

- Comptador. És un mòdul de 12 bits encarregat de portar el compte. Com tot registre perifèric té una adreça d'accés que permet llegir el seu estat i modificar el seu estat. El seu funcionament és simple:
  - En estar activat el prescaler el comptador compta amunt.
  - En passar del darrer estat a 0 s'activa el senyal de sortida del comptador TOV, que també és sortida externa del processador.
  - TOV correspon a un bit del registre de control del timer.
  - TOV roman a 1 fins que s'escriu el bit del registre de control.
  - Quan el senyal d'interrupció TOI està activat s'encarrega directament d'esborrar el bit TOV fent que la seva duració sigui d'1 cicle de rellotge base.

- Registre prescaler. És un comptador intern al comptador que genera la freqüència (programable) de treball del comptador. El seu funcionament és simple:
    - Es programa amb 4 bits (amb valors que van de 0 a 12) proporcionant una divisió de la freqüència de treball del comptador en potències de dos que va de 2^0 a 2^{12}.
    - Un valor superior a 12 atura el prescaler i el comptador.
    - S'inicialitza en carregar el comptador.
- Conté dos registres comparador que funcionen de la mateixa manera i amb el mateix mode de funcionament. El funcionament dels comparadors és simple:
    - Cada registre té una adreça pròpia per poder carregar el valor de comparació.
    - Cada comparador proporciona una sortida externa OCA/OCB que actua d'acord amb el mode de treball especificat en dos bits del registre de control, i que s'especifica en la taula 8.1.
    - Si s'activa el corresponent bit d'interrupció, quan el comptador passar pel valor del comparador es genera una interrupció.

Mode	Funció
OFF: 00	Comparadors desactivats.
SET0: 01	OC a 1 quan el valor comptador és més petit del valor comparador.
SET1: 10	OC a 1 quan el valor comptador és igual o més gran del valor comparador.
INV: 11	S'inverteix el valor d'OC quan el comptador passa pel valor del comparador

Taula 8.1. Modes de funcionament dels comparadors

- Finalment conté el registre de control que pot ser llegit i escrit. La funció de cada bit és la següent:

Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bits 5 a 4	Bits 3 a 0
TOV	OCA	OCB	TOI	OCAI	OCBI	Mode	Prescaler

    - Bit 11. Indica si el comptador ha sobreeixit. S'esborra escrivint un 0 en aquest bit. Si es troba activat el bit d'interrupció per sobreeiximent del comptador TOI aleshores només es genera un pols d'un període de rellotge (s'esborra automàticament).
    - Bit 10 i 9. Sortides dels comparador A i B, respectivament. Valen 1 quan es compleix la condició de comparació. El seu funcionament està d'acord amb els modes de funcionament especificats en la taula 8.1.
    - Bits 8, 7 i 6. Bits d'establiment d'interrupció per sobreeiximent del comptador i pas pels comparadors A i B. Un 1 en aquests bits quan les interrupcions estan habilitades genera un pols d'interrupció quan el comptador passa pel valor del comparador.
    - Bits 5 a 4. Estableixen el mode de funcionament dels comparadors.
    - Bits 3 a 0. Estableixen el prescaler. Més gran de 12 no funciona ni prescaler ni comptador. Igual o més petit de 12 proporcionen 12 freqüències de divisió del rellotge del comptador corresponents a les potències de 2 que van des de 2^{12} a 2^0.

El programa de la figura 8.9 mostra un exemple de programació del timer. El programa treu per les sortides de comparació dos pols PWM a una freqüència de 20KHz (treballant amb un rellotge base de 20MHz) pel control de dos motors DC.

```
-- Control de motors DC per PWM emprant Timer0
--            Freq=20000Hz -> prescaler=0, cmpA a 1/3 i cmpB a 2/3 de període
(   0, "0100000000011111"),     -- 0x401F - RJMP main (+31)
(   9, "0100000100100110"),     -- 0x4126 - RJMP isrTOV0i (+294)  --A 304
(  17, "0000000000000000"),-    -- 0x0000 - NOP
--
(  32, "0011000100000000"),     -- 0x3100 – main: LDI R16,--CMPA timer a 1/3
(  33, "0000110101100110"),     -- 0xD66                --2^12-666=3430
(  34, "0111101100000010"),     -- 0x7B02 - OUT TIMER0CMPA, R16
(  35, "0011000100000000"),     -- 0x3100 - LDI R16,  --CMPB timer a 2/3
(  36, "0000111010110011"),     -- 0xEB3                --2^12-333
(  37, "0111101100000011"),     -- 0x7B0C - OUT TIMER0CMPB, R16
(  38, "0011000100000000"),     -- 0x3100 - LDI R16,  --Control timer:
(  39, "0000000100010000"),     -- 0x181    --TOI, presc=0, modeCMP=10
(  40, "0111101100000001"),     -- 0x7B0C - OUT TIMER0CTRL, R16
(  41, "0000000100000111"),     -- 0x0107 - SEI (BSET 0x7)
(  42, "0000000000000000"),     -- 0x0000 - test: NOP
(  43, "0100111111111110"),     -- 0x4FF6 - RJMP test (-2)
(  44, "0000000000000000"),     -- 0x0000 - NOP
--Rutina d'interrupció isrRxD (ha arribat una dada)
(  304, "0011000111010000"),    -- 0x77DD – isrTOV0i: LDI X,--Inicialitzar timer0
(  305, "0000110000100111"),    -- 0xC27    --2^12-1000+15 (t_accés interrupt)=3111
(  306, "0111101111010000"),    -- 0x7B0C - OUT TIMER0CNT, X
(  307, "0110010100000101"),    -- 0x6505 - continuar: RETI
(  308, "0000000000000000"),    -- 0x0000 - NOP
```

Figura 8.9. Programa de control de motors per PWM

El funcionament del programa és simple:

- El programa inicialitza el timer: control de comparadors en mode SET0 i habilitació d'interrupció del bit TOV.
- El timer ha de treballar a una freqüència de 20KHz. S'aconsegueix amb un compte de 1000 del comptador. Per fer-ho s'activa el bit TOI que habilita la interrupció del bit TOV. Cada cop que s'activa la interrupció la rutina de servei d' interrupció de TOV posa el comptador al valor 3111 que permet obtenir una freqüència del comptador de 20000Hz.
- El programa principal només ha d'esperar en un bucle ja que els comparadors actuen autònomament i la interrupció s'encarrega del comptador.

8.7 La UART.

8.7.1 Connexió de la UART amb EduP12

EduP12 incorpora també un mòdul de comunicació UART (figura 8.10). És un mòdul simple que permet comunicar el (transmissió i recepció) processador emprant un protocol RS232 cap a l'exterior.

El mòdul, anomenat UartLite, està preparat per fer una comunicació sèrie simple de transmissió i recepció amb un protocol de 8 bits, sense paritat, i un únic bit de stop. Aquest protocol sol ser el més emprat en aquest tipus de comunicació. Utilitza dues línies de comunicació (la TxD –transmissió- i la

RxD –recepció-) sense mode de *handshake*. La figura 8.x mostra la connexió que s'estableix entre el mòdul i la CPU.

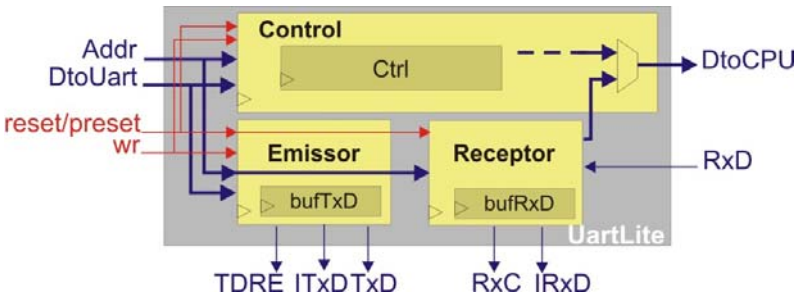


Figura 8.10. Connexió USRAT amb CPU.

La connexió de la comunicació entre la UART i la CPU consta dels següent senyals:

- Els senyals de reset/preset i de rellotge dels mòduls. Tant l'emissor com el receptor s'inicialitzen a 1-lògic d'acord amb el protocol de transmissió RS232.
- La transmissió sèrie es realitza a través dels senyals RxD i TxD. RxD és l'entrada per la que arriben els bits sèrie de la comunicació. El senyal TxD és el bit sèrie que s'emet cap a l'exterior en la comunicació sèrie.
- Els busos DtoUart i DtoCPU. Són els busos de comunicació de dades entre el mòdul UartLite i la CPU. DtoUart són les dades (12 bits) que arriben de la CPU cap al mòdul de comunicació i poden ser de control (cap el mòdul de control) o el byte de transmissió, que es carrega en el buffer de sortida. DtoCPU és la dada que va cap a la CPU que pot ser la lectura del registre de control o el byte que s'ha rebut en la comunicació.
- Els senyals ITxD i IRxD són els senyals d'interrupció que indiquen, respectivament, que el mòdul de transmissió ha enviat una dada i està llest per enviar una altra dada i que el mòdul de recepció acaba de rebre un byte i que està en el buffer per a ser llegida.
- Finalment el senyal TDRE informa de la disponibilitat del buffer per a poder carregar una nova dada en el buffer de sortida per a ser enviada.

La comunicació entre la CPU i el mòdul UartLite es realitza a través de 3 registres d'entrada/sortida:

- El registre de control, que s'explica en l'apartat següent.
- El buffer de transmissió, que és on es carrega la dada que s'ha de transferir al registre encarregat de la transmissió.
- El buffer d'entrada que és on es carrega una dada vàlida que hagi arribat.

8.7.2 El registre de control

El control de UartLite es realitza a través del buffer de control consta dels següents 12 bits:

11	10	9	8	7 a 0
TDRE	RxC	intTxD	intRxD	Prescaler

- El bit 11 correspon al senyal de *registre de transmissió* buit o TDRE. La consulta (llegint el registre de control d'UartLite) permet conèixer quan el buffer del mòdul de transmissió està buit i es pot tornar a enviar-hi un nou byte per a ser transmès. Si la unitat de transmissió està buida, s'iniciarà de seguida la nova transmissió. Si no, s'esperarà a acabar la transmissió per

transmetre el nou byte. Emprant aquest senyal es poden encadenar transmissions seguides de bytes.

- El bit 10 del registre de control correspon al senyal de *recepció completa* RxC. La forma de funcionar d'aquest bit és la següent:
    - Quan no es treballa amb interrupció de recepció, RxC s'activa en arribar un nou byte, el que permet informar al programa, consultant aquest bit, que es troba disponible un nou byte en el buffer d'entrada. El bit s'ha de posar a 0 manualment escrivint un 1 en aquest bit del registre de control. L'operació de control d'aquest bit és, per tant, RxD⊕ctrl(10).
    - Si es treballa amb interrupció per recepció, en activar el bit d'interrupció de recepció es desactiva automàticament aquest bit
- Els bits 9 i 8 són intTxD i intRxD, respectivament. Són els bits d'interrupció. Sempre que es troba habilitat el bit general d'interrupcions aquests bits s'activen en haver acabat una transmissió o haver arribat un nou byte, respectivament. La seva duració és exactament d'un cicle de rellotge base, suficient per habilitar el bit de petició de servei de la interrupció.
- Els 8 bits restants corresponen al prescaler del mòdul de transmissió. La missió del prescaler es reduir la freqüència de funcionament del rellotge base dels mòduls de transmissió i recepció a la velocitat de transmissió (*baudrate*) necessari. La velocitat de transmissió es troba dividint la freqüència base de treball del processador per 16 i restant-li 1, d'acord amb la fórmula:

$$Prescaler = \ arrodonir\left(\frac{fck}{16 * baudrate}\right) - 1$$

L'error que s'introdueix ve donat per:

$$Error = (\frac{Prescaler}{\frac{fck}{16 * baudrate}} - 1) * 100$$

La taula 8.2 dóna el precàlcul del prescaler per a determinades freqüències de funcionament del processador i determinades velocitats de transmissió. S'han ombrejat les cel·les llurs freqüències de transmissió es poden aconseguir treballant amb un prescaler de 8 bits.

	Prescaler Freqüència (MHz)				**Error** (%)			
	25	24	20	18	25	24	20	18
300	5207	4999	4166	3749	-0,01	0,00	0,01	0,00
1200	1301	1249	1041	937	-0,01	0,00	0,03	0,05
2400	650	624	520	468	-0,01	0,00	0,03	0,05
4800	325	312	259	233	0,15	0,16	-0,16	-0,16
9600	162	155	129	116	0,15	-0,16	-0,16	-0,16
19200	80	77	64	58	-0,47	-0,16	-0,16	0,69
38400	40	38	32	28	0,76	-0,16	1,38	-1,01
57600	26	25	21	19	-0,47	-0,16	1,38	2,40
115200	13	12	10	9	3,22	-0,16	1,38	2,40

Taula 8.2. Prescaler segons freqüència del rellotge i velocitat de transmissió

8.7.3 Funcionament de la UART

Mòdul de transmissió

El mòdul de transmissió es basa en una màquina d'estats finits i té el següent funcionament:

- El mòdul està en estat d'espera fins que es carrega una dada en el buffer de sortida. Com tot perifèric connectat a EduP12 la dada es transfereix pel bus de dades de la memòria de sortida i s'adreça amb el bus d'adreces.
- Si el senyal TDRE està a 1-lògic, indicant que no s'està transmetent cap dada, es carrega la dada en el registre de sortida i s'inicia d'immediat la transmissió.
- Tant bon punt s'inicia la nova transmissió es posa a 1-lògic TDRE indicant que es pot acceptar una nova dada per a ser transmesa.
- Si es vol transmetre la dada i TDRE està a 0-lògic, cal fer un bucle d'espera fins que s'allibera el buffer.
- Si el bit d'interrupció intTxD del registre de control de la UART està activat aleshores s'envia un pols d'interrupció en acabar la transmissió indicant que es pot enviar una nova dada. Compta que per a generar una interrupció és necessari que també estigui activat el bit d'interrupció general (bit I del registre d'estat).

La figura 8.11 mostra un esquema de funcionament de la transmissió en el cas de transmissió no emprant interrupcions. Quan s'ha d'emetre una dada el programa s'espera fins que el bit TDRE que es troba en el registre de control està lliure.

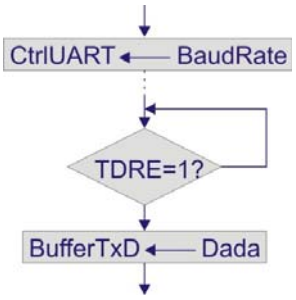


Figura 8.11. Funcionament del mòdul de transmissió.

El programa de la figura 8.12 mostra la codificació seguida en l'adquisició de senyal del compàs magnètic CMPS03.

```
-- Treballant amb compàs magnètic CMPS03
-- Rutines de servei d'interrupció
( 0, "0100000000011111"),          -- 0x401F - RJMP main (+31)
( 1, "0100000011111110"),          -- 0x40F8 - RJMP isrExt4down (+254)  → a 256
( 2, "0100000100000101"),          -- 0x4105 - RJMP isrExt4up (+261)    → a 264
( 3, "0000000000000000"),          -- 0x0000 - NOP
-- Inicialització de perifèrics
-- Interrupció externa bit0 flancs negatiu i positiu:
( 32, "0011000100000000"),         -- 0x3100 - main: LDI R16,
( 33, "0000000000000011"),         -- 0x003
( 34, "0111100100000011"),         -- 0x7903 - OUT intEXT0, R16
-- Divisor presc timer0 a 2^12
( 35, "0011000100000000"),         -- 0x3100 - LDI R16,
( 36, "0000000000001100"),         -- 0x00C
```

```
( 37, "0111101100000001"),              -- 0x7B01 - OUT TIMER0CTRL0, R16
--Control usart: no interrupció, presc=10 (115200bauds-20MHz)
( 38, "0011000100000000"),              -- 0x3100 - LDI R16,
( 39, "0000000010000001"),              -- 0x081
( 39, "0000000000001010"),              -- 0x00A,
( 40, "0111110100000000"),              -- 0x7D00 - OUT CTRLUSARTLITE, R16
-- Habilitació d'interrupcions
(  41, "0000000100000111"),             -- 0107 - SEI (BSET 0x7)
-- Programa principal
( 42, "0000000000000000"),              -- 0x0000 - test: NOP
( 43, "0100111111111110"),              -- 0x4FF5 - RJMP test (-2)
( 44, "0000000000000000"),              -- 0x0000 - NOP
--Rutina d'interrupció isrExt4down
--En flanc de baixada del CMPS03 s'agafa valor timer i s'envia a leds i a hyperterminal
( 256, "0111010000000000"),    -- 0x7400 - IN R0, CNT0
( 257, "0111100000001011"),    -- 0x780B - OUT PORTB, R0
( 258, "0111110000000010"),    -- 0x7C02 - OUT TXDUSARTLITE, R0
( 259, "0110010100000101"),    -- 0x6505 - continuar: RETI
( 260, "0000000000000000"),    -- 0x0000 - NOP
--Rutina d'interrupció isrExt4up
--En flanc de pujada del CMPS03 es posa el comptadors a 0
( 264, "0011000000000000"),    -- 0x3000 - isrExt4down: LDI R0,
( 265, "0000000000000000"),    -- 0x000
( 266, "0111101000000000"),    -- 0x7A00 - OUT CNT0, R0
( 267, "0110010100000101"),    -- 0x6505 - RETI
( 268, "0000000000000000"),    -- 0x0000 - NOP
```

Figura 8.12. Programa de captura de senyal de compàs magnètic (operant en mode PWM) i que es mostra en l'hyperterminal. En cursiva les instruccions referents a la preparació de la UART i la transmissió de dades.

El funcionament, simple, mostra el funcionament del mode de transmissió de dades, del sensor cap a l'ordinador, emprant comunicació per RS232:

- Tot primer el programa especifica les rutines de tractament de les dues interrupcions que s'utilitzen pel sensor. Detecten els flancs positius i negatius del senyal de sortida del compàs magnètic. El flanc positiu indica al timer que s'ha d'inicialitzar. El negatiu agafa la mesura i l'envia als leds i el transmet cap a l'ordinador.
- Les primeres instruccions del programa principal inicialitzen els perifèrics.
- El programa principal és un bucle que no fa res. Tot el tractament es fa en les rutines de servei d'interrupció.
- Finalment es posen les rutines de servei d'interrupció.

En aquest exemple s'ha agafat la captura de dades del sensor emprant el bit 0 d'entrada/sortida per interrupció. Les posicions de memòria que contenen la posició de les rutines de servei es troben en les posicions 1 i 2 de memòria pel flanc negatiu i positiu, respectivament.

Mòdul de recepció

El funcionament del mòdul de recepció es similar al de transmissió basant-se en el següent esquema de funcionament:

- El mòdul de recepció està en estat d'espera fins que el bit d'entrada RxD es posa a 0, canvi que indica que està arribant una dada per la línia de recepció.
- El circuit de recepció sincronitza la recepció dels bits d'entrada enmig del cicle de recepció de cada bit. Amb aquesta mesura s'espera minimitzar la pèrdua de bits.
- En haver rebut els 10 bits (corresponents als 8 bits de la dada més els bits de capçalera i final) es comprova si els bits de capçalera i final són 0 i 1 respectivament. De ser així es posa el senyal de *recepció completa* RxC a 1-lògic indicant que s'ha rebut una dada. Si no està activada la interrupció per recepció, RxC roman a 1-lògic fins que es reinicialitza. El bit RxC és un bit del registre de control.
- Si el bit d'interrupció intRxD està activat s'envia una petició d'interrupció al mòdul d'interrupcions i es desactiva, de forma automàtica, RxC. Compta que per a generar una interrupció és necessari que també estigui activat el bit d'interrupció general (bit I del registre d'estat).
- El programa de la figura 8.13 mostra la codificació seguida en l'adquisició de senyal del compàs magnètic CMPS03.

```
-- Test de recepció molt simple (funcionant)
--          Rebuda de dades d'hyperterminal per interrupció
--          Mostra la dada rebuda en leds

(  0, "0100000000011111"),     -- 0x401F - RJMP main (+31)
( 15, "0100000100010000"),     -- 0x4110 - RJMP isrRxD (+272)
( 17, "0000000000000000"),     -- 0x0000 - NOP
--
( 32, "0011000100000000"),     -- 0x3100 - LDI R16, → Control usart:
( 33, "0000000110000001"),     -- 0x0x181           → interrupció,9600bauds-20MHz
( 34, "0111110100000000"),     -- 0x7D0C - OUT CTRLUSARTLITE, R16
( 35, "0000000100000111"),     -- 0x0107 - SEI (BSET 0x7)
( 36, "0000000000000000"),     -- 0x0000 - test: NOP
( 37, "0100111111111110"),     -- 0x4FF6 - RJMP test (-2)
( 38, "0000000000000000"),     -- 0x0000 - NOP
--Rutina d'interrupció isrRxD (ha arribat una dada)
( 288, "0111010111010001"),    -- 0x75D1 - isrRxD: IN X, RXDUSARTLITE
( 289, "0111100111010001"),    -- 0x79DB - OUT PORTB, X
( 290, "0110010100000101"),    -- 0x6505 - continuar: RETI
( 291, "0000000000000000"),    -- 0x0000 – NOP
```

Figura 8.13. Programa de recepció de dades des de PC que s'envien cap al port de sortida (leds).

8.8 Controlador de 7-segments

Els 7-segments són uns dispositius optoelectrònics que integren 7 leds disposats en forma de 8 en un mateix suport. El control dels leds permet mostrar pel dispositiu caràcters hexadecimals.

Per mostrar un caràcter hexadecimal en el 7-segments cal una descodificació prèvia. La codificació dels caràcters hexadecimals empra el codi binari ascendent. Donat que hi ha 16 caràcters hexadecimals, amb 4 dígits binaris n'hi ha prou per a representar-los tots. La codificació passa per representar cada codi hexadecimal en els leds del 7-segment (figura 8.14; veure també exercici 1 de

l'apartat 3.8), anomenats amb les lletres *a* a la *g*. Sovint els 7-segments disposen d'un led addicional que representa al punt decimal, i que se sol anomenar led *h*.

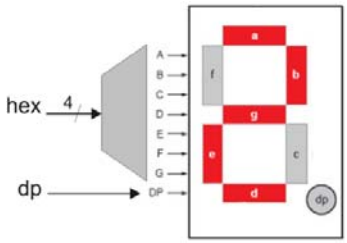


Figura 8.14. Codificació hexadecimal a 7-segments.

Per tant, el control d'un 7-segments implica convertir el codi hexadecimal a 4 bits (es pot fer per programa) als 7 bits d'entrada en el 7-segments (se li pot afegir el bit de punt decimal). Si es tenen 4 7-segments, implica tenir una sortida de 28 pins (corresponents a cadascuna de les 8 entrades per 7-segment), fet que ho fa prohibitiu.

La IP integrada en el processador porta un petit controlador que distribueix l'encesa de cada 7-segments a una freqüència de 200 Hz fet que permet, controlant el 7-segments que s'encén a cada unitat de temps, compartir el mateix bus de dades per a tots els 7-segments (figura 8.14). El control de cada 7-segments es fa a partir del senyal de control d'encesa del 7-segments (controlant el senyal de capacitació AN0-3). Per altra part, la IP conté un registre per cada 7-segments que manté la dada a visualitzar (i que és refrescada contínuament).

Per a poder accedir a cada 7-segments, s'ha assignat una adreça d'entrada/sortida per 7-segments. Així una instrucció de sortida envia la dada continguda en un registre en el 7-segments especificat.

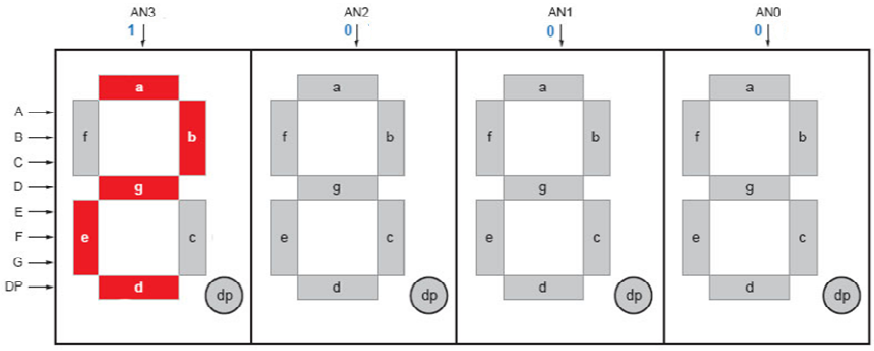


Figura 8.14. Connexió dels 4 7-segments.

La figura 8.15 mostra un tros de programa que inicialitza els 7-segments 0 a 3 als valors 0, 1, 2 i 3 respectivament.

```
-- Inicialització 7-segments
(  0, "0100000000011111"),   -- 401F - RJMP main (+31)→ Crida programa principal
-- Inicialitzar set-segments
( 32, "0011000111010000"),   -- 31D0 - LDI X,        → El registre índex indica inici codificació
( 33, "0000001111111111"),   -- 1023
( 34, "1100001100001101"),   -- C30D - LPM +R16, X→ Incrementar i carregar valor en R16 que...
( 35, "0111100100000100"),   -- 7B00 - OUT SS0, R16→...s'envia a 7-segments 0, ...
( 36, "1100001100001101"),   -- C30D - LPM +R16, X
( 37, "0111100100000101"),   -- 7B01 - OUT SS1, R16
```

```
(  38, "1100001100001101"),    -- C30D - LPM +R16, X
(  39, "0111100100000110"),    -- 7B02 - OUT SS2, R16
(  40, "1100001100001101"),    -- C30D - LPM +R16, X
(  41, "0111100100000111"),    -- 7B03 - OUT SS3, R16
...
--A partir de la posició 1024 hi ha la codificació dels codis hexadecimals a 7-segments
(  1024, "0000000000111111"),    -- 0
(  1025, "0000000000000110"),    -- 1
(  1026, "0000000001011011"),    -- 2
(  1027, "0000000001001111"),    -- 3
(  1028, "0000000001100110"),    -- 4
(  1029, "0000000001101101"),    -- 5
(  1030, "0000000001111101"),    -- 6
(  1031, "0000000000000111"),    -- 7
(  1032, "0000000001111111"),    -- 8
(  1033, "0000000001100111"),    -- 9
(  1034, "0000000001011111"),    -- a
(  1035, "0000000001111100"),    -- b
(  1036, "0000000001011000"),    -- c
(  1037, "0000000001011110"),    -- d
(  1038, "0000000001111011"),    -- e
(  1039, "0000000001110001"),    -- f
(  1040, "0000000010000000"),    -- pd
```

Figura 8.15. Exemple de programa d'inicialització de 7-segments.

8.9 Resum del capítol

Els perifèrics són els dispositius annexes al processador que permeten la seva comunicació amb l'exterior. En aquest capítol s'ha mostrat el comportament dels perifèrics bàsics que s'inclouen en la versió simple del processador.

És important entendre com s'estableix la connexió entre el processador i els perifèrics, mostrada en tots els perifèrics introduïts. Qualsevol perifèric es pot connectar al processador si estableix una adreça única que el permet identificar. Tota entrada/sortida amb el perifèric es fa a través de reegistres. En perifèrics grans es pot crear una interfície entre processador i perifèric que s'encarrega de la comunicació entre ambdós. Aquesta interfície rep el nom de *wrapper*. Aquest aspecte es tracta amb més detall en el capítol 10.

8.9 Resum del capítol

Capítol 9

ASSEMBLANT I SIMULANT AMB EDUP12

En aquest capítol s'introdueix un assemblador bàsic que permet escriure de forma més fàcil els programes que corren en EduP12.

L'assemblador que es presenta és simple i, tot i que realitza un petit tractament d'errors de sintaxi, en cap cas evita que la introducció d'errors de programació per part de l'usuari sigui corregit. Per això el capítol introdueix un conjunt de normes de *bona pràctica* en l'escriptura d'un programa en assemblador per evitar la pèrdua de temps degut a errades de sintaxi i de programació.

9.1 L'assemblador

L'assemblador és un petit llenguatge de baix nivell, molt proper a la màquina, que simplifica enormement la tasca de programació del processador. La seva missió principal és la de permetre al programador escriure el codi de la màquina emprant un conjunt de mnemotècnics que representen al codi màquina de la instrucció.

UEn programa escrit en llenguatge assemblador, un cop assemblat, es converteix en codi màquina. El llenguatge assemblador és, per tant, específic del processador. Un programa escrit en assemblador no es compila, sinó que s'assembla.

El programa que realitza l'assemblatge també s'anomena assemblador. El programa escrit en assemblador és el codi font, i el programa assemblat és el codi objecte. El codi objecte és el que conté el codi màquina

9.2 Assemblador asmEduP12.

L'assemblador del processador EduP12 consta de directives i de mnemotècnics. Les directives són comandes que simplifiquen la comprensió del programa donant noms específics a les dades i als registres que empra el programa. Els mnemotècnics són els noms donats a les instruccions, que ja s'han introduït en capítols anteriors.

9.2.1 Directives.

Les directives emprades en el processador són:

.ORG <pos>

Indica la posició on es posa l'origen de les instruccions. <pos> és una constant numèrica (decimal o hexadecimal) indica la posició de memòria on es comencen a posar les instruccions que succeeixen a la directiva.

.CON <nom> = <constant>

Dóna nom a valors constants que s'empren en el programa. <nom> és el nom que rep la constant. <constant> és una constant numèrica (decimal o hexadecimal.

.DEF <nom> = <Rx>

Dóna nom als registres que s'utilitzen en el programa. <nom> és el nom que rep el registre. <Rx> és el registre que s'anomena. Tot i que s'assigni un nom a un registre, no és requisit haver d'emprar el nom donat per adreçar al registre.

.DW <valor 1> , <valor 2> , ... , <valor n>

Permet guardar un conjunt de valors en memòria de programa. És útil, per exemple, per guardar taules de valors que han de ser utilitzats pel programa.

9.2.2 Instruccions

Els mnemotècnics de les instruccions que s'empren en el llenguatge assemblador ja s'han introduït en capítols anteriors i es troben especificades en l'apèndix A1. La primera columna de la taula mostra el format que cal seguir en cada instrucció.

9.3 Guia de bones pràctiques

Per a programar un processador en llenguatge assemblador es recomanable seguir els següents consells:

- Primer de tot cal fer un diagrama de flux del procés.
- Si es fa un diagrama de flux d'alt nivell, basat en instruccions de llenguatge de programació d'alt nivell, és útil convertir-lo a un diagrama de flux basat en instruccions del tipus RTL (llenguatge de transferència de registres).
- El diagrama de flux s'ha de basar en el format de les instruccions assemblador amb les que es treballa. Els típics bucles i salts condicionals de llenguatges de programació d'alt nivell aquí convé especificar-los en funció de les instrucció de salt de què es disposa, basades en el registre d'estat.

En el que respecte al format que s'ha de seguir en l'escriptura del programa en assemblador convé seguir les següents normes:

- Format general:
    - Totes les instruccions acaben en *;*. Quan no hi ha *;* en una línia es considera comentari i l'assemblador no la tracta.
    - Quan una instrucció s'identifica per etiqueta, aquesta ocupa el primer camp i acaba amb *:*.
- Respecte a directives:
    - És útil redefinir constants i registres a valors pràctics del programa

  - Però s'han de designar als registres x, y i z com a tals sempre que s'usin com a índex.
  - La directiva .ORG és útil sempre que es vulgui posar un determinat programa en una posició determinada de la memòria. Cal anar en compta, però, en no reescriure una part del programa per un ús excessiu d'aquesta directiva.

- En números amb signe, no hi ha d'haver espai entre signe i número
- Quan s'empra pre-increment o pre-decrement en registre índex, el signe ha d'anar enganxat al registre i/o a la constant. El registre índex ha de ser X, Y o Z. Exemples:

```
LD R16, +X;
STD Z+40, R0;
```

- El format general és :

```
.ORG 0;              - Tot programa comença en la posició 0
RJMP main;           - Salt al programa principal
... RJMP isr1;       - Salts a rutines de servei d'interrupció (quan s'empren interrupcions)
...
main: ...    - Rutina principal del programa
....
Isr1: ...            - Inicia primera rutina de servei d'interrupció
...
RETI
...
```

9.4 Exemples.

Exemple 1.

En aquest primer exemple es mostra en els leds l'estat dels interruptors de la placa Spartan3. Quan es pitja el botó 0 s'agafa la dada de l'interruptor i s'envia als leds. Mentre no es polsi, no s'actualitzen els leds.

El programa presenta una nova versió del programa ja introduït en apartats anteriors, però es fa pera què e puguin comparar la versió que no empra interrupcions amb la que sí n'empra.

El plantejament del programa és el següent:

- S'empren els següents elements de la placa:
  - Els interruptors. Es troben connectats en el PORTA d'entrada.
  - Els leds. Es troben connectats al port de sortida PORTB.
  - S'empra el botó 0. Es troba connectat en el pin 1 del port d'interrupcions. Com que no s'empren interrupcions, s'empra el port com a entrada sota el nom PORTE.
- El funcionament del programa és simple. S'està en un bucle d'espera fins que es polsi el botó. Aleshores s'agafa l'estat dels interruptors i s'envia als leds.
- En el diagrama de flux (figura 9.1) s'observa que per conèixer si s'ha polsat el botó 0 (connectat al pin 1 del PORTA) es fa la AND amb 0x002. Com que el pin està en repòs a 0-lògic, mentre es compleixi la condició implica que no s'ha polsat el botó. En polsar el botó es llegeix el nou valor en el port d'interruptors (PORTA) i s'envia als leds (PORTB).
- En el programa s'ha previst una rutina d'espera d'uns 2 ms (treballant a 20MHz) per limitar l'efecte rebot del botó.

9.4 Exemples.

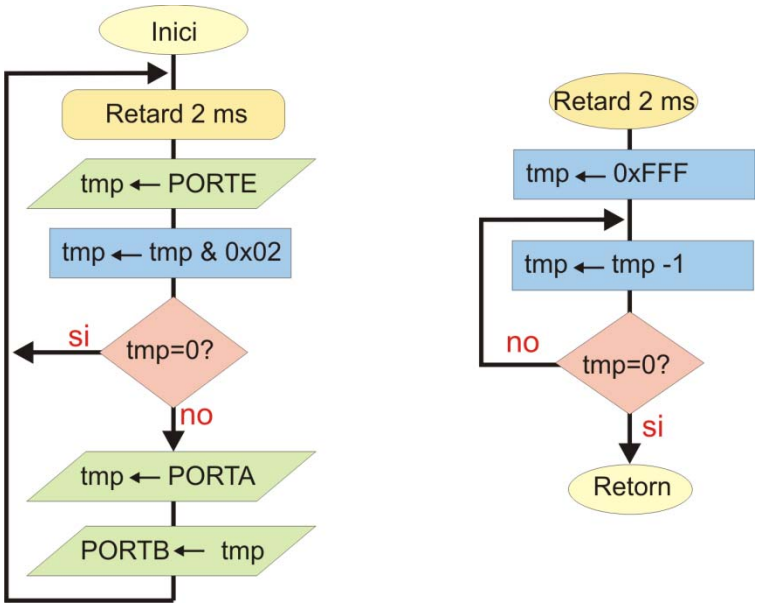


Figura 9.1. Diagrama de flux de l'exemple 1.

La figura 9.2 mostra el codi assemblador del programa.

-- INTERRUPTORS EN LEDS	
.DEF tmp = R16;	
inici: RCALL espera;	(0, "0101000000000111"), -- 0x5007 - rcall 7
IN tmp, PORTE;	(1, "0111000100000010"), -- 0x7102 - in r16 000010
ANDI tmp, 0x002;	(2, "0011000100000010"), -- 0x3102 - andi r16 0x002
	(3, "0000000000000010"), -- 0x2
BRZE inici;	(4, "1110111111011001"), -- 0xEFD9 - brze -5
IN tmp, PORTA;	(5, "0111000100000000"), -- 0x7100 - in r16 000000
OUT PORTB, tmp;	(6, "0111100100000001"), -- 0x7901 - out 000001 r16
RJMP inici;	(7, "0100111111111000"), -- 0x4FF8 - rjmp -8
--Bucle d'espera	
espera: LDI tmp, 0xFFF;	(8, "0011000100000000"), -- 0x3100 - ldi r16 0xfff
	(9, "0000111111111111"), -- 0xFFF
bucle: DEC tmp;	(10, "0011000100001001"), -- 0x3109 - dec r16
BRNZ bucle;	(11, "1111111111110001"), -- 0xFFF1 - brnz -2
RET;	(12, "0110000000000100"), -- 0x6004 - ret
a) Codi assemblador	**b) Codi màquina**

Figura 9.2. Codi assemblador i codi màquina de l'exemple 1.

Introducció d'interrupcions en l'exemple 1.

En un processador amb interrupcions, el plantejament fet d'aquest exercici és ineficient, ja que el programa sempre queda esperant que es polsi el botó sense poder fer cap altra acció.

La interrupció permet millorar el funcionament del processador en tant que l'allibera de la tasca d'espera. Programant una interrupció per flanc (de pujada, per exemple) en el polsador 1, el processador pot estar realitzant altres tasques mentre no es el polsador. En polsar el polsador (produeix un flanc de pujada) s'entra en la rutina de servei de la interrupció i aquesta és la responsable de realitzar la corresponent actuació.

El plantejament del problema ara és molt més simple:

- El programa principal inicialitza el processador per interrupció per flanc de pujada en el polsador 1. Donat que el flanc de pujada del polsador 1 habilita la interrupció externa 4 s'ha d'inicialitzar el registre de control d'interrupcions externes a 0x008.
- El programa principal és el responsable del procés del sistema. Pot estar fent altres accions mentre no es el botó. El fet que es deixi en un bucle infinit es deu al fet que aquest programa és molt simple i no se li requereixen més accions al processador.
- En activar el polsador el processador salta a tractar la rutina de servei de la interrupció, que agafa la dada dels interruptors (PORTA) i l'envia als leds (PORTB). La interrupció elimina la rutina d'espera.

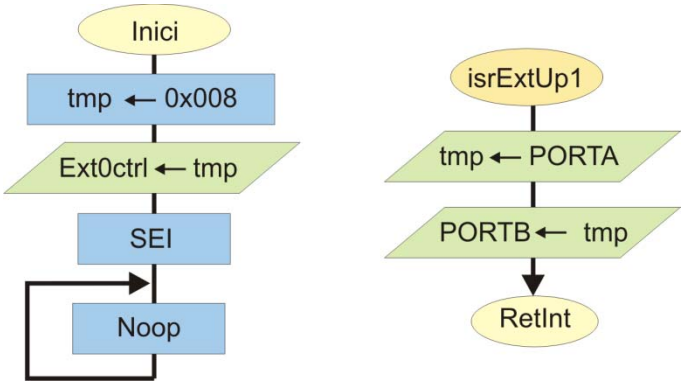


Figura 9.3. Diagrama de flux de l'exemple 1 amb interrupció.

La figura 9.4 mostra el codi assemblador del programa.

-- INTERRUPTOR EN LEDS	
--TREBALLANT AMB INTERRUPCIONS	
.DEF tmp = R16;	
.org 0;	
RJMP inici;	(0, "0100000000011111"), -- 0x401F - rjmp 31
.org 4;	
RJMP isrExtUp1;	(4, "0100000000101101"), -- 0x402D - rjmp 45
NOP;	(5, "0000000000000000"), -- 0x0 - nop
.org 32;	
inici: LDI tmp, 0x008;	(32, "0011000100000000"), -- 0x3100 - ldi r16 0x008
OUT EXT0ctrl, tmp;	(33, "0000000000001000"), -- 0x8
SEI;	(34, "0111100100000011"), -- 0x7903 - out 000011 r16
bucle: NOP;	(35, "0000000100000111"), -- 0x107 - sei
RJMP bucle;	(36, "0000000000000000"), -- 0x0 - nop
NOP;	(37, "0100111111111110"), -- 0x4FFE - rjmp -2
--Rutina de servei d'interrupció	(38, "0000000000000000"), -- 0x0 - nop
.org 50;	
isrExtUp1: IN tmp, PORTA;	(50, "0111000100000000"), -- 0x7100 - in r16 000000
OUT PORTB, tmp;	(51, "0111100100000001"), -- 0x7901 - out 000001 r16
RETI;	(52, "0110000000000101"), -- 0x6005 - reti
NOP;	(53, "0000000000000000"), -- 0x0 - nop
a) Codi assemblador	**b) Codi màquina**

Figura 9.4. Codi assemblador i codi màquina de l'exemple 1 amb interrupcions.

9.4 Exemples.

Exemple 2. Treballant amb els 7-segments.

Realitzar un programa que faci rotar un 1 entre 0's en els 7-segments.

El plantejament fet en l'exercici és el següent:

- S'inicialitzen dos registres amb els valors que mostren en els 7-segments un zero i un 1.
- En un bucle es fa rotar l'1 entre zeros en els 7-segments.
- S'introdueix una rutina d'espera de 200ms per veure rotar el 1.

La figura 9.5 mostra el diagrama de flux, mentre que en la figura 9.6 hi ha els codi assemblador i màquina.

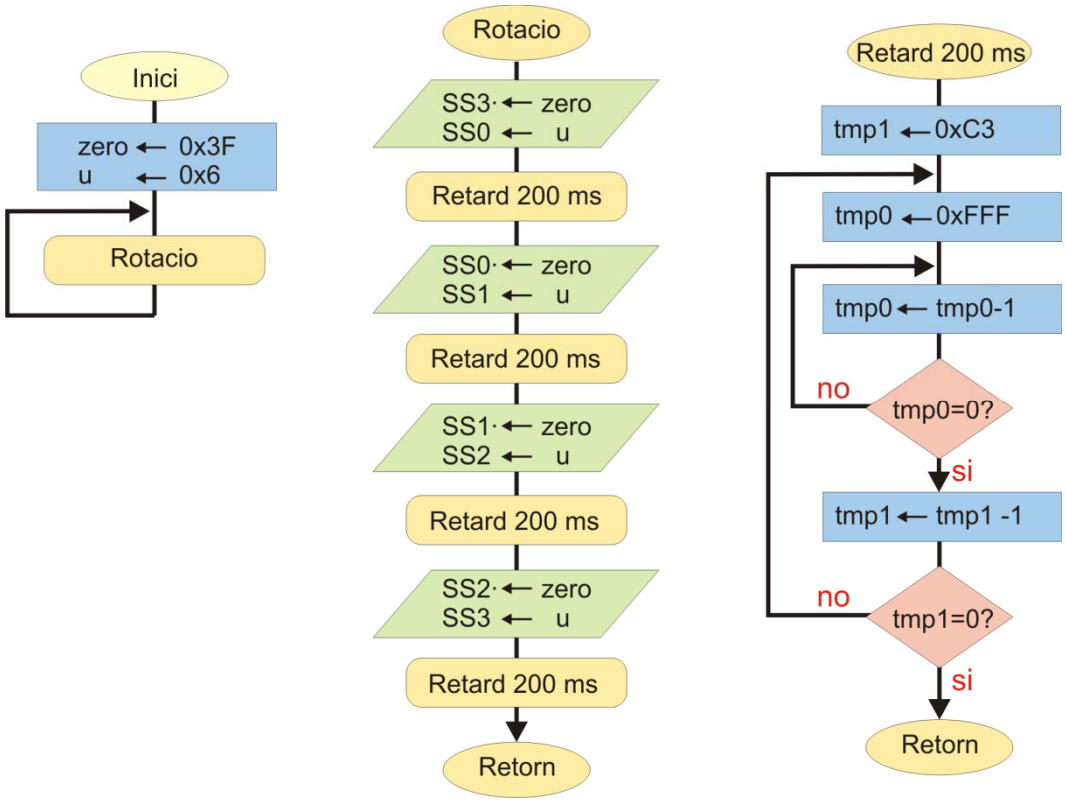


Figura 9.5. Diagrama de flux de l'exemple 2.

-- UN 1 ENTRE 0'S .DEF zero = R0; .DEF u = R1; .DEF tmp0 = R16; .DEF tmp1 = R17; .ORG 0; --Inici programa RJMP main;	 (0, "0100000000011111"), -- 0x401F - rjmp 31
.ORG 32; --Programa principal main: LDI zero, 0x3F; LDI u, 0x6;	 (32, "0011000000000000"), -- 0x3000 - ldi r0 0x3f (33, "0000000000111111"), -- 0x3F (34, "0011000000010000"), -- 0x3010 - ldi r1 0x6

	(35, "0000000000000110"), -- 0x6
torni: RCALL ss;	(36, "0101000000001101"), -- 0x500D - rcall 13
RJMP torni;	(37, "0100111111111110"), -- 0x4FFE - rjmp -2
.ORG 50; -- Rutina de rotació	
ss: OUT SS3, zero;	(50, "0111100000000111"), -- 0x7807 - out 000111 r0
OUT SS0, u;	(51, "0111100000010100"), -- 0x7814 - out 000100 r1
RCALL ESPERA200;	(52, "0101000000101111"), -- 0x502F - rcall 47
OUT SS0, zero;	(53, "0111100000000100"), -- 0x7804 - out 000100 r0
OUT SS1, u;	(54, "0111100000010101"), -- 0x7815 - out 000101 r1
RCALL ESPERA200;	(55, "0101000000101100"), -- 0x502C - rcall 44
OUT SS1, zero;	(56, "0111100000000101"), -- 0x7805 - out 000101 r0
OUT SS2, u;	(57, "0111100000010110"), -- 0x7816 - out 000110 r1
RCALL ESPERA200;	(58, "0101000000101001"), -- 0x5029 - rcall 41
OUT SS2, zero;	(59, "0111100000000110"), -- 0x7806 - out 000110 r0
OUT SS3, u;	(60, "0111100000010111"), -- 0x7817 - out 000111 r1
RCALL ESPERA200;	(61, "0101000000100110"), -- 0x5026 - rcall 38
RET;	(62, "0110000000000100"), -- 0x6004 - ret
NOP;	(63, "0000000000000000"), -- 0x0 - nop
.ORG 100; --Espera 200 ms	
espera200: LDI tmp1, 0xD0;	(100, "0011000100010000"), -- 0x3110 - ldi r17 0xc3
	(101, "0000000011000011"), -- 0xC3
bucle2: LDI tmp0, 0xfff;	(102, "0011000100000000"), -- 0x3100 - ldi r16 0xfff
	(103, "0000111111111111"), -- 0xFFF
bucle1: DEC tmp0;	(104, "0011000100001001"), -- 0x3109 - dec r16
BRNZ bucle1;	(105, "1111111111110001"), -- 0xFFF1 - brnz -2
DEC tmp1;	(106, "0011000100011001"), -- 0x3119 - dec r17
BRNZ bucle2;	(107, "1111111111010001"), -- 0xFFD1 - brnz -6
RET;	(108, "0110000000000100"), -- 0x6004 - ret
NOP;	(109, "0000000000000000"), -- 0x0 - nop

Figura 9.6. Codi assemblador i codi màquina de l'exemple 2.

Exemple 3. Comunicació amb ordinador emprant el port sèrie

Quan es produeix una interrupció externa sobre polsador 1 en flanc de baixada, el programa envia la dada posada en els interruptors (PortA) cap a l'ordinador.

Per altra part, tota dada rebuda de l'ordinador en el port RxD s'envia als leds (PortB).

El plantejament fet en l'exercici és el següent:

- Prèvia: En la comunicació amb l'hyperterminal de l'ordinador cal recordar que cal enviar caràcters ASCII per un fàcil reconeixement dels mateixos. Per tant, convé enviar els 8 bits com a caràcter ASCII (dos codis hexadecimals). Per exemple, el codi 0x30 representa el zero i vindrà donat pel codi b00110000.
- Els 7-segments s'inicialitzen amb un punt decimal.
- La transmissió s'inicia quan es polsa el botó 1. La rutina de servei d'interrupció agafa la dada dels interruptors i l'envia als leds, als 7-segments i al buffer de sortida de transmissió.

9.4 Exemples.

- La recepció es fa per interrupció. Cada cop que es rep una dada per RxD es genera interrupció de recepció de dada. En la rutina de servei de la interrupció s'envia la dada al PORTB (leds) i als 7-segments.
- A tot això el programa principal inicialitza ports i registres i queda en espera d'events.

La figura 9.7 mostra el diagrama de flux, mentre que en la figura 9.8 hi ha els codi assemblador i el codi màquina.

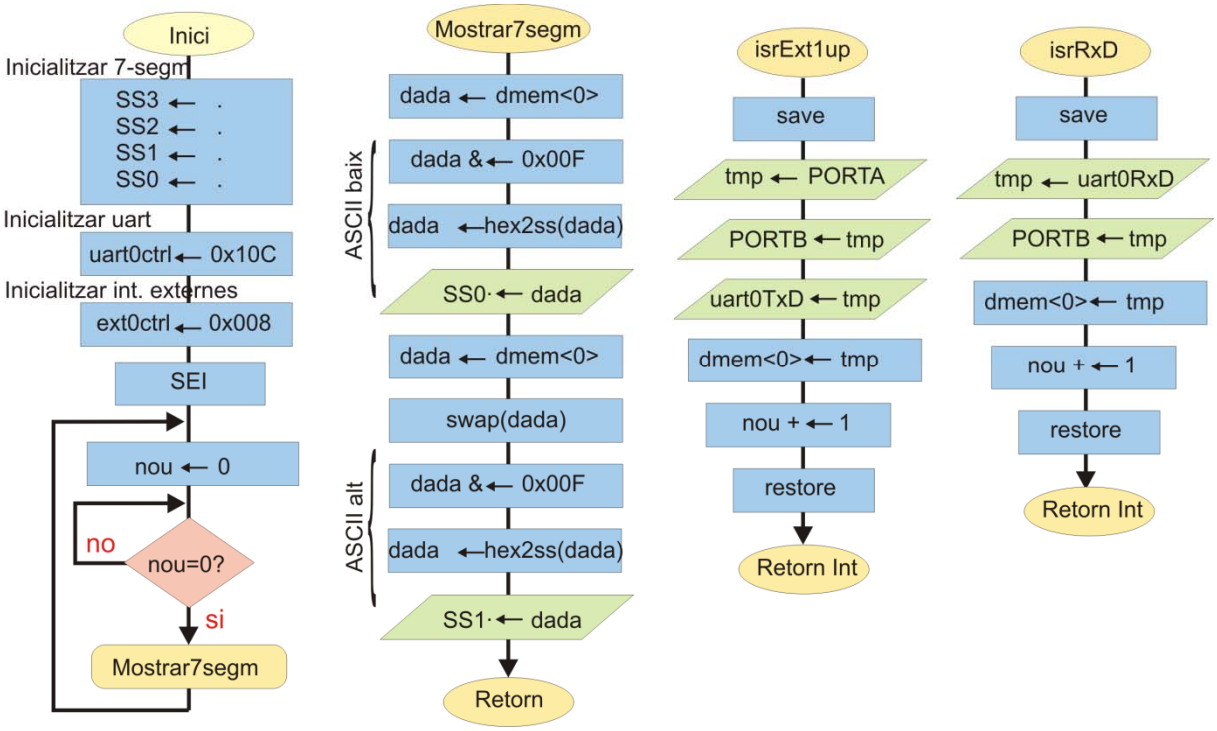


Figura 9.7. Diagrama de flux de l'exemple 3.

-- COMUNICACIONS	
.CON posDada = 0; --Posició de memòria	
.DEF nou = R0;	
.DEF dada = R1;	
.DEF tmp = R16;	
.ORG 0;	
RJMP inici;	
.ORG 4;	(0, "0100000000011111"), -- 0x401F - rjmp 31
RJMP isrExt1up;	--
.ORG 15;	(4, "0100001111100011"), -- 0x43E3 - rjmp 995
RJMP isrRxD;	--
	(15, "0100010000111100"), -- 0x443C - rjmp 1084
.ORG 32; --Programa principal	--
inici: LDI X, hex2ss16;	(32, "0011000111010000"), -- 0x31D0 - ldi r29 1616
	(33, "0000011001010000"), -- 0x650
LPM tmp, X;	(34, "1100001100001100"), -- 0xC30C - lpm r16 r29
OUT SS3, tmp;	(35, "0111100100000111"), -- 0x7907 - out 000111 r16

9.4 Exemples.

OUT SS2, tmp;	(36, "0111100100000110"), -- 0x7906 - out 000110 r16
OUT SS1, tmp;	(37, "0111100100000101"), -- 0x7905 - out 000101 r16
OUT SS0, tmp;	(38, "0111100100000100"), -- 0x7904 - out 000100 r16
LDI tmp, 0x10C; --115200baud	(39, "0011000100000000"), -- 0x3100 - ldi r16 0x10c
	(40, "0000000100001100"), -- 0x10C
OUT UART0CTRL, tmp;	(41, "0111110100000000"), -- 0x7D00 - out 100000 r16
LDI tmp, 0x008; --Ext1up	(42, "0011000100000000"), -- 0x3100 - ldi r16 0x008
	(43, "0000000000001000"), -- 0x8
OUT EXT0CTRL, tmp;	(44, "0111100100000011"), -- 0x7903 - out 000011 r16
SEI;	(45, "0000000100000111"), -- 0x107 - sei
bucle: CLR nou;	(46, "0010100000000000"), -- 0x2800 - clr r0
espera: CPI nou, 0;	(47, "0011000000000110"), -- 0x3006 - cpi r0 0
	(48, "0000000000000000"), -- 0x0
BRZE espera;	(49, "1110111111101001"), -- 0xEFE9 - brze -3
RCALL mostrarEn7S;	(50, "0101000000110001"), -- 0x5031 - rcall 49
RJMP bucle;	(51, "0100111111111010"), -- 0x4FFA - rjmp -6
NOP;	(52, "0000000000000000"), -- 0x0 - nop
.ORG 100; --Rutina tractament 7-segm	
mostrarEn7S: LDS dada, posDada;	(100, "1100000000010100"), -- 0xC014 - lds r1 0
	(101, "0000000000000000"), -- 0x0
ANDI dada, 0x00F;	(102, "0011000000010010"), -- 0x3012 - andi r1 0x00f
	(103, "0000000000001111"), -- 0xF
LDI X, hex2ss0;	(104, "0011000111010000"), -- 0x31D0 - ldi r29 1600
	(105, "0000011001000000"), -- 0x640
ADD X, dada;	(106, "0000100111010001"), -- 0x9D1 - add r29 r1
LPM dada, X;	(107, "1100001000011100"), -- 0xC21C - lpm r1 r29
OUT SS0, dada;	(108, "0111100000010100"), -- 0x7814 - out 000100 r1
LDS dada, posDada;	(109, "1100000000010100"), -- 0xC014 - lds r1 0
	(110, "0000000000000000"), -- 0x0
SWAP dada;	(111, "0011001000010000"), -- 0x3210 - swap r1
ANDI dada, 0x00F;	(112, "0011000000010010"), -- 0x3012 - andi r1 0x00f
	(113, "0000000000001111"), -- 0xF
LDI X, hex2ss0;	(114, "0011000111010000"), -- 0x31D0 - ldi r29 1600
	(115, "0000011001000000"), -- 0x640
ADD X, dada;	(116, "0000100111010001"), -- 0x9D1 - add r29 r1
LPM dada, X;	(117, "1100001000011100"), -- 0xC21C - lpm r1 r29
OUT SS1, dada;	(118, "0111100000010101"), -- 0x7815 - out 000101 r1
RET;	(119, "0110000000000100"), -- 0x6004 - ret
NOP;	(120, "0000000000000000"), -- 0x0 - nop
.ORG 1000; --Interrupció TxD	
isrExt1up: SAVE;	(1000, "0110100000000111"), -- 0x6807 - save
IN tmp, PORTA;	(1001, "0111000100000000"), -- 0x7100 - in r16 000000
OUT PORTB, tmp;	(1002, "0111100100000001"), -- 0x7901 - out 000001 r16
OUT UART0TXD, tmp;	(1003, "0111110100000010"), -- 0x7D02 - out 100010 r16
STS posDada, tmp;	(1004, "1101000100000100"), -- 0xD104 - sts 0 r16
	(1005, "0000000000000000"), -- 0x0
INC nou;	(1006, "0011000000001000"), -- 0x3008 - inc r0

RESTORE;	(1007, "0110100000000110"), -- 0x6806 - restore
RETI;	(1008, "0110000000000101"), -- 0x6005 - reti
NOP;	(1009, "0000000000000000"), -- 0x0 - nop
.ORG 1100;	
isrRxD: SAVE; --Interrupció RxD	(1100, "0110100000000111"), -- 0x6807 - save
IN tmp, UART0RXD;	(1101, "0111010100000001"), -- 0x7501 - in r16 100001
OUT PORTB, tmp;	(1102, "0111100100000001"), -- 0x7901 - out 000001 r16
STS posDada, tmp;	(1103, "1101000100000100"), -- 0xD104 - sts 0 r16
	(1104, "0000000000000000"), -- 0x0
INC nou;	(1105, "0011000000001000"), -- 0x3008 - inc r0
RESTORE;	(1106, "0110100000000110"), -- 0x6806 - restore
RETI;	(1107, "0110000000000101"), -- 0x6005 - reti
NOP;	(1108, "0000000000000000"), -- 0x0 - nop
.ORG 1600; --Codis ascii2ss	
hex2ss0: .dw 0x3F;	(1600, "0000000000111111"), -- 0x3F - .dw 0x3f
hex2ss1: .dw 0x6;	(1601, "0000000000000110"), -- 0x6 - .dw 0x6
hex2ss2: .dw 0x5B;	(1602, "0000000001011011"), -- 0x5B - .dw 0x5b
hex2ss3: .dw 0x4F;	(1603, "0000000001001111"), -- 0x4F - .dw 0x4f
hex2ss4: .dw 0x66;	(1604, "0000000001100110"), -- 0x66 - .dw 0x66
hex2ss5: .dw 0x6D;	(1605, "0000000001101101"), -- 0x6D - .dw 0x6d
hex2ss6: .dw 0x7D;	(1606, "0000000001111101"), -- 0x7D - .dw 0x7d
hex2ss7: .dw 0x7;	(1607, "0000000000000111"), -- 0x7 - .dw 0x7
hex2ss8: .dw 0x7F;	(1608, "0000000001111111"), -- 0x7F - .dw 0x7f
hex2ss9: .dw 0x67;	(1609, "0000000001100111"), -- 0x67 - .dw 0x67
hex2ss10: .dw 0x5F;	(1610, "0000000001011111"), -- 0x5F - .dw 0x5f
hex2ss11: .dw 0x7C;	(1611, "0000000001111100"), -- 0x7C - .dw 0x7c
hex2ss12: .dw 0x58;	(1612, "0000000001011000"), -- 0x58 - .dw 0x58
hex2ss13: .dw 0x5E;	(1613, "0000000001011110"), -- 0x5E - .dw 0x5e
hex2ss14: .dw 0x79;	(1614, "0000000001111001"), -- 0x79 - .dw 0x79
hex2ss15: .dw 0x71;	(1615, "0000000001110001"), -- 0x71 - .dw 0x71
hex2ss16: .dw 0x80;	(1616, "0000000010000000"), -- 0x80 - .dw 0x80

Figura 9.8. Codi assemblador i codi màquina de l'exemple 3.

9.4 Resum del capítol.

El capítol ha introduït l'assemblador com al llenguatge de més baix nivell que simplifica la tasca de programació del processador.

La missió de l'assemblador és la de traduir un programa escrit en les instruccions assemblador del processador a codi màquina. Tot i que el llenguatge és molt simple, convé seguir les normes i consells donades en l'apartat 9.3

Els exemples presentats en l'apartat 9.4 serveixen per veure programes complerts realitzats amb l'assemblador del processador. Convé agafar-los com a guia.

9.5 Exercicis.

1. Siguin els dos programes següents. Es demana:
   a. Realitzar el diagrama de flux.
   b. Dir similituds i diferències. Quin valor màxim en a es pot entrar en cada cas? Analitzar la profunditat de la pila que es necessita i indicar si en algun cas es poden produir problemes de saturació.

```
.DEF a = R30;
.DEF f = R31;
.ORG 0;
inici:  LDI f, 0;
        IN a, PORTA;
        RCALL rutina;
fi:     RJMP fi;
.ORG 10;
rutina: ADD f, a;
        OUT PORTB, f;
        DEC a;
        BRNZ rutina;
        RET;
```

```
.DEF a = R30;
.DEF f = R31;
.ORG 0;
inici:  IN a, PORTA;
        LDI f, 0;
        RCALL rutina;
fi:     RJMP fi;
.ORG 10;
rutina: PUSH a;
        DEC a;
        BRZE calc;
        RCALL rutina;
calc:   POP a;
        ADD f, a;
        OUT PORTB, f;
        RET;
```

2. Analitzar els següents programes. Fer un diagrama de flux. Entendre què fan. Pots establir alguna relació entre ells? Observar que els programes 1 i 2 fan servir la rutina Retard.

Programa 1

```
.DEF led = R1;
.ORG 0;
inici:  LDI X, hex0;
rotar:  LPM led, +X;
        OUT SS1, led;
        RCALL retard;
        CPI X, hex32;
        BRMI rotar;
        RJMP inici;
.ORG 2000;
hex0: .dw 0;
hex1: .dw 1;
hex2: .dw 2;
hex4: .dw 4;
hex8: .dw 8;
hex16: .dw 16;
hex32: .dw 32;
```

Programa 2

```
.DEF led = R2;
.ORG 0;
inici:  LDI led, 0x1;
rotar:  OUT SS0, led;
        LSL led;
        RCALL retard;
        CPI led, 0x020;
        BRNE rotar;
        RJMP inici;
```

Rutina Retard

```
.ORG 100;
retard: LDI R17, 240;
bucle2: LDI R16, 1665;
bucle1: DEC R16;
        BRNZ bucle1;
        DEC R17;
        BRNZ bucle2;
        RET;
        NOP;
```

Programa 3

```
.DEF r = R3;
.ORG 0;
        RJMP inici;
.ORG 9;
        RJMP ovfTIM0;
.ORG 32;
inici:  LDI r, 0x109;
        OUT TIMER0ctrl, r;
        LDI r, 0x001;
        SEI;
aqui:   RJMP aqui;
.ORG 1000;
ovfTIM0: OUT SS2, r;
        LSL r;
        CPI r, 0x100;
        BRNE tornar;
        LDI r, 0x1;
tornar: RETI;
```

9.5 Exercicis.

3. Modificar l'exemple 2 de l'apartat 9.4 de manera que els nombres que es rotin s'entrin en polsar el botó 1.
   Ajuda: substituir els números *zero* i *u* per *primer* i *segon*, respectivament. Inicialitzar-los. Cada cop que es polsi el botó (emprar interrupcions, a ser possible) el valor de *primer* passa a *segon* i el valor dels interruptors passa a *primer*.
4. Donat el següent programa, realitzar un diagrama de flux i comentar el seu funcionament.

```
.con n = 5;
.con start = 100;

.def nou = R16;
.def vell = R17;

inici:      ldi z, n;
    clr vell;
entrar:     in nou, PORTA;
    cp nou, vell;
    breq entrar;
    std z+start, nou;
    mov vell, nou;
    dec z;
    brpl entrar;
    ldi z, n;
sortir:     ldd vell, z+start;
    out PORTB, vell;
    dec z;
    brpl sortir;
aqui:       rjmp aqui;
```

5. Realitzar un sumador de nombres de 8 bits. El funcionament serà el següent:
   a. Quan es premi el botó 1 s'entra el nombre A (pel PortA connectat als interruptors).
   b. Quan es premi el botó 2 s'entra el nombre B (pel PortA connectat als interruptors).
   c. En prémer el botó 3 es fa la suma d'ambdós números.
6. Emprant les interrupcions externes dels botons 0 a 2 (recordar que el botó 3 és el reset), fer un programa que mostri pels 7-segments el botó i flanc que ha provocat la interrupció. Per exemple: en prémer botó 2 i flanc de pujada mostrar 2-UP; i en prémer botó 0 per flanc de baixada mostrar 0-dO, etc..
7. El control d'un servomotor es realitza a través d'un pols (que dura entre 1ms i 2 ms) en un període de 20ms. Quan el pols és d'1.5 ms, el motor queda aturat. Conforme s'augmenta el pols cap a 2 ms, el motor va girant més ràpid cap a un sentit. Si el pols disminueix cap a una duració d'1 ms, el motor va girant més ràpid en l'altre sentit. Es demana:
   a. Generar una ona que seleccionant un botó (0, 1 o 2) aturi el motor o el faci girar en un o altre sentit.
   b. Controlar ara el motor entrant la dada a través de la UART.

Capítol 10

EDUP12 COM A PROPIETAT INTEL·LECTUAL

Aquest capítol resumeix el tractament que el processador rep quan es tracta com a propietat intel·lectual. S' introdueixen normes de conducta per quan es vol utilitzar el processador en un disseny o per quan es volen introduir nous perifèrics en el mateix.

A través de diferents codis vhdl es mostra com s'estableix la connectivitat quan es vol expandir EduP12 amb més perifèrics.

10.1 Estructura interna del processador EduP12 i funcionament global.

A nivell estructural EduP12 és un processador personalitzable i autocontingut que funciona de manera autònoma un cop descarregat sobre una FPGA.

La figura 10.1 visualitza la interacció interna que s'estableix entre processador, memòria i perifèrics. El processador, com a nucli, conté la unitat de procés i la unitat de control. El programa es guarda en la memòria de programa i les dades de càlculs intermedis es guarden en la memòria de dades. Totes dues memòries es col·loquen en la memòria interna de la FPGA, motiu pel que es creen en el moment de síntesis de tot el processador. Els perifèrics es troben agrupats en el fitxer *ports.vhd*. Els busos d'entrada i sortida realitzen la connexió entre processador i perifèrics.

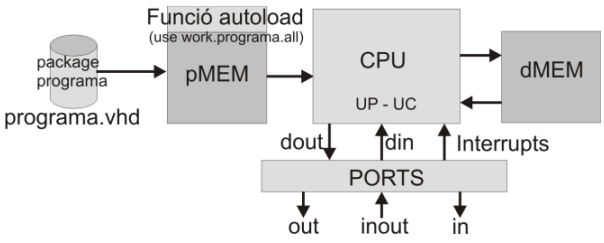


Figura 10.1. estructura interna de EduP12

El programa escrit en codi màquina es troba en un paquet (*package*) anomenat *programa.vhd*. Es tracta d'un fitxer *vhdl* que conté un paquet en el que es descriu el codi màquina que s'ha de carregar

a cada posició de memòria, i que correspon al programa que executarà el processador. Aquest paquet es crida des de la memòria de programa mitjançant una funció d'autocàrrega en el moment d'executar la síntesis.

El format del paquet és versàtil, de manera que només fa falta descriure a quina posició de memòria va cada codi màquina. El format és de tipus array. La figura 10.2 mostra un programa d'exemple en el que es realitza la suma de dos nombres immediats emmagatzemats en els registres R0 i R1.

```
library ieee;
use ieee.std_logic_1164.all;
use work.PK_eduP12.all;
package test_simple is
    type mostra is record
        dir: integer range 0 to 2**mp-1;
        valor: std_logic_vector(15 downto 0);
    end record;
    type contingut_mem is array (natural range <>) of mostra;
    constant test: contingut_mem:=(
        -- POSAR EL PROGRAMA A PARTIR D'AQUÍ
        -- FORMAT INSTRUCCIONS: (adreça, "codi màquina"), --Comentaris
        (  0, "0011000000000000"),          -- 3000 - LDI R0,
        (  1, "0000000000000001"),          -- 0x001
        (  2, "0011000100000000"),          -- 3100 - LDI R16,
        (  3, "0000000100000000"),          -- 0x100
        (  4, "0000100100000000"),          -- 0900 - ADD R16, R0
        (  5, "0100111111111111"),          -- 4FFF - RJMP inp (-1)
        (  6, "0000000000000000")           -- 0000 – NOP
        --NO POSAR RES A PARTIR D'AQUÍ
    );
end;
```

Figura 10.2. Exemple fitxer programa.vhd.

10.2 EduP12 i perifèrics

La figura 10.3 mostra la composició del processador emprant una jerarquia *top-down*, en la que es descriuen els components que el composen a partir del nivell més alt. El nivell superior està format (en la implementació en Xilinx) per un esquemàtic que agrupa a un PLL i EduP12. EduP12 és la descripció del processador en vhdl.

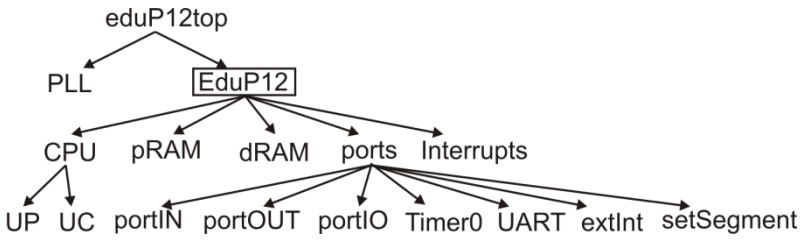


Figura 10.3. Estructura top-down del processador.

La figura 10.4 mostra aquest esquemàtic. Es pot observar que s'hi ha inclòs un pll (*phase-locked loop*). La seva tasca és la de generar un rellotge intern altament precís de manera que eviti els

desfases en els senyals de rellotge interns entre els diferents components. És altament recomanable la seva instrucció i sol ser un component de la llibreria del paquet amb el que es treballa.

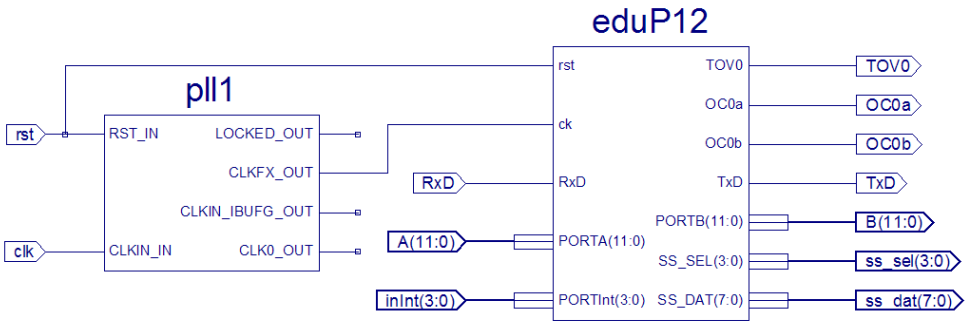


Figura 10.4. Configuració de EduP12

El símbol *eduP12* correspon a la descripció de més alt nivell en VHDL. La figura 10.5 mostra l'entitat corresponent al processador. Es pot observar que tot nou port a introduir s'ha de posar com a senyal en l'entrada/sortida del processador.

```
entity eduP12 is
        Port ( rst : in  STD_LOGIC;
            ck : in  STD_LOGIC;
            --Afegir ports quan calgui----------------------------------------------------------------------
            PORTA : in  STD_LOGIC_VECTOR (n-1 downto 0);  --Sortida PortA
            PORTB : out  STD_LOGIC_VECTOR (n-1 downto 0);--Sortida PortA
            PORTInt : in  STD_LOGIC_VECTOR (ints-1 downto 0);-- Port amb interrupcions
            SS_SEL : out  STD_LOGIC_VECTOR (3 downto 0);  --Sortida selecció a set-segment
            SS_DAT : out  STD_LOGIC_VECTOR (7 downto 0);  --Sortida dada a set-segment
            TOV0 : out  STD_LOGIC;                        --Sortida OVF timer0
            OC0a : out  STD_LOGIC;                        --Sortida comparadorA timer0
            OC0b : out  STD_LOGIC;                        --Sortida comparadorB timer0
            TxD : out  STD_LOGIC;                         --Sortida transmissió USART
            RxD : in  STD_LOGIC;                          --Entrada recepció USART
            ----------------------------------------------------------------------------------------------
        );
end eduP12;
```

Figura 10.5. Configuració de EduP12

El processador és un nucli fixat que ve totalment descrit a nivell vhdl, a excepció de l'entrada/sortida. Es pot observar que és en els ports on el processador es personalitza més. L'entrada/sortida del processador configura el nucli de lligam entre processador i el món exterior. És normal, així, que sigui l'estructura més variable d'una aplicació a l'altra. És en els ports que s'haurà d'interactuar en cada nova aplicació.

En la plataforma inicial de EduP12 s'han previst els següents components:

- El propi processador, intern, que no es veu reflectit en l'entrada/sortida.
- La memòria que tampoc té reflex en l'entrada/sortida.
- El mòdul d'interrupcions que sí que afecta en alguns ports, i que s'ha de tenir present si es vol fer servir.

- Els següents perifèrics:
    - Port d'entrada de 12 bits *A*.
    - Port de sortida de 12 bits *B*.
    - Port d'entrada per interrupció externa (flanc de pujada i flanc de baixada) *inInt*.
    - Port de sortida per a set-segments, que consta dels ports selecció de set-segments *ss_sel* i bus de dades *ss_dat*.
    - Unitat de comunicació UART, que té l'entrada *RxD* i la sortida *TxD*. Quan s'empra la UART convé realitzar el mòdul d'interrupcions per atendre cada nova dada que es rep.
    - Un timer amb dos comparadors, que té com a sortides la sortida d'excés *TOV0* i les de comparació *OC0a* i *OC0b*.

10.3 Normes a tenir presents per a la introducció de nous perifèrics

La creació de perifèrics per connectar-los amb el processador passa per tenir present una sèrie de consideracions:

- El perifèric es descriu a nivell de vhdl. L'entitat del perifèric contindrà les entrades i/o sortides cap a l'exterior i la connexió amb el processador, que es pot fer en un *wrapper*.
- El perifèric es connecta al processador dintre del mòdul *ports*, que és un fitxer vhdl estructurat.
- La identitat dels registres de control i/o dades dels perifèrics es determina a partir d'una adreça especificada en el paquet pk_eduP12.vhd. Tot registre que ha de ser llegit o escrit pel processador requereix d'adreça.
- Quan s'ha de connectat un senyal de dades de sortida, el propi bus de dades de sortida del processador es connecta al registre de sortida del perifèric. La connexió és directa. En senyals de dades d'entrada, la dada que arriba de l'exterior s'envia cap al mòdul *ports* (on es connecta el nostre perifèric) i aquest la multiplexa amb les altres senyals d'entrada per enviar-la cap al processador.
- Tots els senyals de dades d'entrada i/o sortida que es comuniquen amb l'exterior s'han de propagar des del perifèric, passant pels mòduls ports i processador (l'entitat més alta del processador) fins a l'esquemàtic que en realitza la connexió amb els pins de la FPGA. Cal recordar, en aquest sentit, que tot nou senyal d'entrada/sortida especificat en l'esquemàtic (*top level*) necessita de l'assignació d'un pin en el moment de síntesis de la FPGA (en Xilinx es defineix en *PACE* després de fer la síntesis i abans del mapeig i routing).

10.3.1 Exemple d'introducció d'un port de sortida PORTB en eduP12.

Anem a suposar que posem de nou el PORTB d'entrada d'ampla 12 bits en el processador eduP12. Els passos a seguir són els següents:

1. Es defineix el port en un fitxer vhdl. La figura 10.6 mostra la descripció d'aquest component. Es pot observar que porta especificat una adreça, que correspon a l'adreça d'entrada/sortida del port que s'ha de definir en el paquet *PK_eduP12*.

```
...
use work.PK_eduP12.all;
entity oPort is
        Port ( rst : in  STD_LOGIC;
```

```
            ck : in  STD_LOGIC;
            ld: in  STD_LOGIC;
            addr : in  STD_LOGIC_VECTOR (mio-1 downto 0);    --Única per cada perifèric
            aPort : in  STD_LOGIC_VECTOR (mio-1 downto 0);
            d : in  STD_LOGIC_VECTOR (n-1 downto 0);
            q : out  STD_LOGIC_VECTOR (n-1 downto 0)
        );
end oPort;

architecture Behavioral of oPort is
signal wr: STD_LOGIC;
begin
        wr <= '1' when ((addr=aPORT) and ld='1') else '0';
        process (rst, ck)
        begin
            if rst='1' then q <= (others=>'0');
            elsif (ck'event and ck = '0') then
                if wr='1' then q <= d;
                end if;
            end if;
        end process;
end Behavioral;
```

Figura 10.6. Descripció del nou port de sortida PORTB

2. Un cop definit el port, cal definir el *wrapper*. Donat que en aquest cas és molt simple, el propi port fa de *wrapper*, pel que no calen accions posteriors.
3. S'ha de connectar el port en l'entitat *ports* del processador. *Ports*, com a tal, és un codi vhdl estructurat. Per tant, els passos a seguir són:
    a. Primer es declara com a component el nou port.
    b. Després es crida com a component en l'arquitectura.
    c. Finalment el port de sortida s'ha de declarar en l'entitat de *ports*.

    La figura 10.7 mostra l'adjunció del nou port en el component ports.

```
...
use work.PK_eduP12.all;
entity ports is
        Port ( ...
            --Afegir ports quan calgui
            ...
            PORTB : out  STD_LOGIC_VECTOR (n-1 downto 0);                    --Sortida PortB
            ...
        );
end ports;

architecture Behavioral of ports is
component oPort
        Port ( rst : in  STD_LOGIC;
            ck : in  STD_LOGIC;
            ld: in  STD_LOGIC;
            addr : in  STD_LOGIC_VECTOR (mio-1 downto 0);
```

```
            aPort : in  STD_LOGIC_VECTOR (mio-1 downto 0);
            d : in  STD_LOGIC_VECTOR (n-1 downto 0);
            q : out  STD_LOGIC_VECTOR (n-1 downto 0)
        );
end component;
...                         -- Declaració altres components

signal sortidaPORTB : STD_LOGIC_VECTOR (n-1 downto 0);
...

begin
        --Definició de sortides
        ...
        PORTB <= sortidaPORTB;
        ...  --Altres sortides

        --Especificació de components d'entrada/sortida
        PortBout: oPort port map (rst, ck, wr, addr, aPORTB, fromCPU, sortidaPORTB);
        ...  --Altres components
end Behavioral;
```

Figura 10.7. Adjunció del nou port en el component *ports*. Es mostren els passos que s'han de seguir per incloure un nou port de sortida en *ports.*

4. En l'entitat *EduP12* s'ha de posar el nou port de sortida. La figura 10.8 mostra com s'ha adjuntat el nou port en el processador. Cal recordar que, en aquest mateix fitxer, s'ha d'actualitzar la declaració del component ports que, ara, té una nova sortida, així com també cal posar-la en la seva crida.

```
...
use work.PK_eduP12.all;
entity eduP12 is
        Port ( ...
            --Afegir ports quan calgui----------------------------------------------------------------
            ...
            PORTB : out  STD_LOGIC_VECTOR (n-1 downto 0);                    --Sortida PortB
            ...
        );
end eduP12;

architecture Behavioral of eduP12 is
        ...  -- Components
        component ports
            Port ( ...
            -------Afegir ports quan calgui-----------------------------------------------------------
            ...
            PORTB : out  STD_LOGIC_VECTOR (n-1 downto 0);                    --Sortida PortB
            ...
            ------------------------------------------------------------------------------------------
            );
        end component;
```

```
        signal ...
begin
    -- Definició dels components
    perifèrics: ports port map ( ...
        --Afegir ports quan calgui----------------------------------------------------
        ...
        PORTB => PORTB,
        ...
        ------------------------------------------------------------------------
    );
end Behavioral;
```

Figura 10.8. Declaració del component *EduP12.* S'han de posar els senyals d'entrada/sortida (en aquest cas de sortida). Només es mostren les instruccions que cal introduir.

5. Es genera el nou símbol processador (que ara té una sortida més) i s'actualitza el símbol en l'esquemàtic (top-level) del processador.
6. Un cop sintetitzat s'han de posar els pins de la FPGA que corresponen a aquest port.

10.3.2 Exemple d'introducció d'un port d'entrada PORTA en eduP12.

La introducció d'un port d'entrada en el processador segueix un camí similar al que s'ha emprat per posar el port de sortida. La única diferència està en el tractament de l'adreça del bus. Anem a suposar que posem de nou el PORTA d'entrada d'ampla 12 bits en el processador eduP12. Els passos a seguir són els següents:

1. Es defineix el port en un fitxer vhdl. La figura 10.9 mostra la descripció d'aquest component. Es pot observar que porta especificat una adreça, que correspon a l'adreça d'entrada/sortida del port que s'ha de definir en el paquet *PK_eduP12*.

```
...
use work.PK_eduP12.all;
entity iPort is
    Port ( rst : in  STD_LOGIC;
        ck : in  STD_LOGIC;
        d : in  STD_LOGIC_VECTOR (n-1 downto 0);
        q : out  STD_LOGIC_VECTOR (n-1 downto 0)
    );
end iPort;

architecture Behavioral of iPort is
begin
    process (rst, ck)
    begin
        if rst='1' then q <= (others=>'0');
        elsif (ck'event and ck = '0') then q <= d;
        end if;
    end process;
end Behavioral;
```

Figura 10.9. Descripció del nou port d'entrada

2. Un cop definit el port, cal definir el *wrapper*. Donat que en aquest cas és molt simple, el propi port fa de *wrapper*, pel que no calen accions posteriors.
3. S'ha de connectar el port dintre el component *ports* del processador. *Ports*, com a tal, és un codi vhdl estructurat. Per tant, els passos a seguir són:
    a. Primer es declara com a component el nou port.
    b. Després es crida com a component en l'arquitectura.
    c. S'ha de connectar el senyal d'entrada en la instrucció vhdl que agrupa a totes les entrades dintre el bus d'entrada del processador.
    d. Finalment el port de sortida s'ha de declarar en l'entitat de *ports*.
    La figura 10.10 mostra l'adjunció del nou port en el component ports.

```
...
use work.PK_eduP12.all;
entity ports is
Port (...
        --Afegir ports quan calgui
        PORTA : in  STD_LOGIC_VECTOR (n-1 downto 0);          --Entrada PortA
        PORTB : out  STD_LOGIC_VECTOR (n-1 downto 0);         --Sortida PortB
        ...
);
end ports;

architecture Behavioral of ports is
...
component iPort
        Port ( rst : in  STD_LOGIC;
            ck : in  STD_LOGIC;
            d : in  STD_LOGIC_VECTOR (n-1 downto 0);
            q : out  STD_LOGIC_VECTOR (n-1 downto 0)
        );
end component;
...
signal qin : STD_LOGIC_VECTOR (n-1 downto 0);
...
begin
        --Sortides
        PORTB <= sortidaPORTB;
        ...
        --Bus d'entrada -- S'hi han de posar tots els busos que es poden llegir
        toCPU <= qin when (addr = aPORTA) else
            ...else
            (others=>'0');
        --Especificació de components d'entrada/sortida
        PortAin: iPort port map (rst, ck, PORTA, qin);
        PortBout: oPort port map (rst, ck, wr, addr, aPORTB, fromCPU, sortidaPORTB);
        ...
end Behavioral;
```

Figura 10.10. Adjunció del nou port d'entrada en *ports*. Cal fixar-se en aquest cas en el multiplexor del mòdul *ports* que condueix la dada dels ports d'entrada cap al bus d'entrada del processador.

4. En l'entitat *EduP12* s'ha de posar el nou port d'entrada. Els passos a seguir són els mateixos que s'han emprat per introduir el port de sortida en l'apartat 10.3.2.
5. Es genera el nou símbol processador (que ara té una sortida més) i s'actualitza el símbol en l'esquemàtic (top-level) del processador.
6. Un cop sintetitzat s'han de posar els pins de la FPGA que corresponen a aquest port.

10.3.3 Exemple de treball amb ports d'entrada i de sortida

Com ja s'ha introduït en el capítol 8, l'entrada/sortida del processador es fa emprant les instruccions IN i OUT. La figura 10.11 mostra el fitxer *programa.vhd* complert que agafa l'entrada del PORTA i l'envia al PORTB sempre que s'hagi modificat. Es pot observar que, en la creació del programa, només cal canviar les instruccions que conformen el programa.

```
...
use work.PK_eduP12.all;
package programa is
        type mostra is record
            dir: integer range 0 to 2**mp-1;
            valor: std_logic_vector(15 downto 0);
        end record;
        type contingut_mem is array (natural range <>) of mostra;
        constant test: contingut_mem:=(
            --origen     --.org 0;
            --les adreces inicials indiquen interrupcions
            --adreça 0 --> la posició memòria conté un salt a l'adreça origen del programa
            ------------------------------------------------------------------------------------------------------------
            -- Programa: TEST IN-OUT.
            --    Es mira si hi ha dada en PORTA
            --                          ..i es treu pel PORTB sempre que sigui nova
            ------------------------------------------------------------------------------------------------------------
             (   0, "0100000000011111"),    -- 401F - RJMP main (+31)              ->a main
            (   1, "0000000000000000"),     -- 0000 - NOP
            --
            (  32, "0011000111110000"),     -- 31F0 - LDI Z,
            (  33, "0000000011111111"),     -- 0x0FF
            (  34, "0111100111110001"),     -- 79F1 - loope: OUT PORTB, Z
            (  35, "0111000100000000"),     -- 7100 - loopi: IN R16, PORTA          --Agafar dada
            (  36, "0001101111110000"),     -- 1BF0 - CP Z, R16
            (  37, "1111011111101001"),     -- EFF9 - BRZE loopi (-3)
            (  38, "0000011111110000"),     -- 07F0 - MOV Z, R16
            (  39, "0100111111111010"),     -- 4FF9 - RJMP loope (-6)
            (  40, "0000000000000000")      -- 0000 - NOP
            ------------------------------------------------------------------------------------------------------------
        );
end;
```

Figura 10.11. Exemple de programa per EduP12. És la descripció del paquet vhdl que es sintetitza en la FPGA i que descriu el programa que s'executa. El programa es personalitza entre la instrucció 0 i la 40, en aquest cas.

10.4 Introducció de nous perifèrics amb interrupcions

La interrupció (com s'ha introduït en el capítol 7) és un mecanisme de detecció d'events que potencia enormement la capacitat de procés del processador ja que permet que aquest pugui dedicar el seu temps en processos propis de càlcul i només presti atenció als events quan aquests ocorrin. Treballant en processos d'adquisició de senyal en temps real la interrupció és un mecanisme indispensable en el processador.

La introducció de perifèrics amb interrupció porta una lleugera complicació a la introducció d'un port sense interrupcions. Els passos que cal seguir són els següents:

1. Tot event d'interrupció ha de durar, exactament, un cicle de rellotge. Les especificacions del mòdul detector d'interrupció imposa aquesta restricció per evitar duplicitat en la interrupció. Aquest senyal és una sortida del perifèric que s'ha de transportar cap al mòdul *ports*.
2. En el mòdul *ports* els senyals d'interrupció es recullen i s'envien cap al mòdul de tractament d'interrupció.
3. Tota interrupció ha de tenir una adreça d'interrupció assignada. El mòdul ports analitza quan es produeix una interrupció i mira quina adreça és. Aquesta adreça és única. Aquesta adreça ha de correspondre en el programa amb l'adreça que conté l'adreça en la que s'inicia la rutina de servei de la interrupció.

A continuació es mostra un exemple d'introducció d'un port amb interrupcions. Tot i que l'exemple tracta d'un port d'interrupcions externes, cal tenir present que el mecanisme d'interrupció es pot fer servir en qualsevol port i no necessàriament ha d'anar lligat amb entrades externes. Dues interrupcions molt comunes, per exemple, introduïdes en microcontroladors són la d'excés en comptadors i la de recepció de dada en una UART. En el primer cas indica al processador que el comptador/timer ha arribat al seu valor més alt i s'ha reiniciat, mentre que en el segon indica que el buffer d'entrada de la UART hi ha guardada la nova dada que s'acaba de rebre.

10.4.1 Exemple: introducció del port d'interrupcions externes PortD

EduP12 es subministra en la seva versió simple amb un port d'entrada per interrupció de 4 bits. En aquest apartat es mostren els detalls més importants que cal tenir present quan s'introdueixen interrupcions.

1. El primer pas és descriure el mòdul d'entrada. En aquest cas és com un component normal d'entrada, però que només genera un pols d'un cicle de duració cada cop que canvia una de les seves entrades. El mòdul pot ser programat per detectar canvi d'entrada per flanc positiu i/o canvi d'entrada per flanc negatiu. Com que es consideren 4 entrades es poden tenir fins a 8 interrupcions possibles. La figura 10.12 mostra l'entitat del component i la part interna que genera el pols quan es produeix interrupció. En aquest component *addr* és l'adreça del registre de control del perifèric que estableix les interrupcions que s'activen (quins bits i amb quins flancs), *dfromCPU* és el bus de dades que prové del processador i porta el senyal de control, *dfromOut* és el bus de dades extern, *qtoCPU* és el bus de dades que va cap al processador i *qInt* és el bus d'interrupcions del port que va cap al processador.

```
--INTERRUPCIONS EXTERNES
--El registre Ctrl conté l'activació d'interrupcions
--Cada bit té activació per flanc de pujada (bit+sign.) i flanc de baixada(bit-sign.).
...
use work.PK_eduP12.all;
entity intPort is
        Port ( ...
        addr : in  STD_LOGIC_VECTOR (mio-1 downto 0);        -- Adreça del registre de control
        dfromCPU : in  STD_LOGIC_VECTOR (n-1 downto 0);   -- Programa registre de control d'int's
```

```
        dfromOut : in  STD_LOGIC_VECTOR (ints-1 downto 0);  -- Bus de dades extern
        qtoCPU : out  STD_LOGIC_VECTOR (n-1 downto 0);      -- Bus de dades cap a CPU
        qInt : out  STD_LOGIC_VECTOR (2*ints-1 downto 0)    -- Senyals d'interrupció (8 possibles)
        );
end intPort;
architecture Behavioral of intPort is
signal q0, q1, q2, q3, flancUp, flancDown : STD_LOGIC_VECTOR (ints-1 downto 0);
signal ctrl: STD_LOGIC_VECTOR (2*ints-1 downto 0);
begin
        ...
        --Interrupcions
        flancUp <= not(q3) and q2 and q1 and q0;                  -- Cerca de flanc de pujada
        flancDown <= q3 and not(q2) and not(q1) and not(q0);      -- Cerca de flanc de baixada
        process(rst, ck)
        begin
            if rst='1' then qInt<=(others=>'0');
            elsif ck'event and ck='0' then
                for i in 0 to ints-1 loop
                    qInt(2*i)<=flancDown(i) and ctrl(2*i);
                    qInt(2*i+1)<=flancUp(i) and ctrl(2*i+1);
                end loop;
            end if;
        end process;
end Behavioral;
```

Figura 10.12. Mòdul d'entrada per interrupció PortD

2. Un cop definit el component s'ha d'introduir en el component *Ports* amb els altres perifèrics. En aquest cas, apart de les entrades o sortides del component, també s'han d'introduir les interrupcions. La figura 10.13 mostra les parts corresponents a la introducció del component en el mòdul de perifèrics. Es pot observar (s'ha posat en cursiva) que les accions a fer en la introducció del nou mòdul són: (i) declarar com a component el perifèric, (ii) definir el nou perifèric, (iii) declarar els senyals interns quan calgui, (iv) especificar el bus de dades del perifèric que es connecta en el bus d'entrada, (v) especificar els senyals d'interrupcions que s'acoblen en el bus d'interrupcions i (vi) posar els senyals d'entrada/sortida normals d'interrupció del perifèric en *ports*.

```
...
use work.PK_eduP12.all;
entity ports is
        Port ( ...

            --(vi)----------------------------Afegir ports quan calgui
            ...
            PORTint : in  STD_LOGIC_VECTOR (ints-1 downto 0);--Entrada port interrupcions
            ....
            ---------------------------------- Sortides d'interrupció
            interrupts : out STD_LOGIC_VECTOR (2**nInt-1 downto 0); --Comú per tots
            ...
        );
end ports;
```

```
architecture Behavioral of ports is
...
-- (i) Declaració com a component del nou perifèric
component intPort
        Port ( ...
                addr : in  STD_LOGIC_VECTOR (mio-1 downto 0);
                dfromCPU : in  STD_LOGIC_VECTOR (n-1 downto 0);
                dfromOut : in  STD_LOGIC_VECTOR (ints-1 downto 0);
                qtoCPU : out  STD_LOGIC_VECTOR (n-1 downto 0);
                qInt : out  STD_LOGIC_VECTOR (2*ints-1 downto 0)
        );
end component;
...
-- (iv) Declaració de senyals interns
signal qCPUfromINT : STD_LOGIC_VECTOR (n-1 downto 0);
signal qIntFromIntPort: STD_LOGIC_VECTOR (2*ints-1 downto 0);
...
begin
        ...
        -- (iv) Bus d'entrada -- S'hi han de posar tots els busos que es poden llegir
        toCPU <= ... else
                qCPUfromINT when (addr = aCTRLIntExt) else... else
                (others=>'0');
        -- (v) Especificació d'interrupcions
        Interrupcions: process(qIntFromIntPort, ...)
        begin
        interrupts <= (others=>'0');
                        --Int. 0 (main)                  – No existeix, però s'ha de reservar l'adreça 0
        interrupts(2*ints downto 1) <= qIntFromIntPort;          --Int. 8 a 1: IntExtPort
        interrupts(...)<= ...;                                   --Anar posant interrupcions
        end process;
        --Especificació de components d'entrada/sortida
        ...
        -- (ii) Definició del nou perifèric
        PortDInt: intPort port map (rst, ck, wr, addr, fromCPU, PORTint, qCPUfromINT,
        qIntFromIntPort);
        ...
end Behavioral;
```

Figura 10.13. Mòdul d'entrada per interrupció PortD. En cursiva es mostra la instrucció que introdueix les interrupcions del perifèric en *ports*.

S'ha d'observar que, a nivell d'interrupcions, no cal fer res més que introduir-les sota el senyal *interrupts*. *Interrupts* és un bus de dimensió definida pel nombre d'interrupcions que permet el sistema (actualment posat a 16 en PK_edu12P) i que s'encarrega d'agrupar-les en el procés *interrupcions*.

El programa de la figura 8.7 mostra una aplicació en la que s'empra un polsador connectat a un bit del port IntPortD per detectar l'arribada d'una nova dada que la mostra en el port de sortida que es troba connectat als leds.

10.5 Personalització sobre la placa Digilent Spartan3

El processador tal com està definit s'ha provat sobre la placa Digilent Spartan-3 (figura 10.14). És una placa educativa construïda sobre una FPGA Spartan3 XC3S200 de Xilinx amb múltiples ports que resulta ideal per a petites aplicacions comercials.

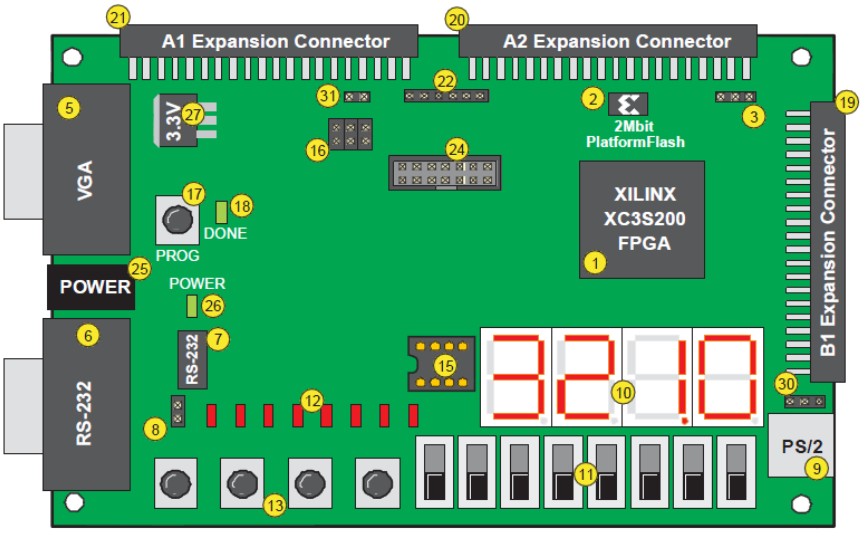


Figura 10.14. Placa Digilent Spartan-3

La figura 10.4 mostra el diagrama de blocs i el *pinout* que s'ha establert en EduP12 en l'esquemàtic del processador, el nivell més alt de la jerarquia. El *pinout* està d'acord amb el patillatge emprat en la placa. En l'apèndix 4 es troba la relació de pins dels perifèrics emprats i la seva connexió en la FPGA de la placa. Aquesta taula fa falta per a qualsevol personalització del processador amb perifèrics sobre aquesta placa.

10.5 Personalització sobre la placa Digilent Spartan3

BIBLIOGRAFIA

En la literatura es pot trobar molta bibliografia referent als fonaments dels computadors i a la síntesis de circuits digitals emprant llenguatges d'alt nivell com ara el VHDL.

Aquí no es vol fer un recull de tota aquesta bibliografia existent, sinó que només es vol deixar constància de bibliografia addicional que complementa el contingut exposat en el llibre amb l'objectiu que sigui el propi lector qui, a partir d'aquest punt de continuïtat, pugui cercar aquell material que més s'adeqüi a les seves inquietuds i li permetin aprofundir una mica més en el tema.

Aquesta bibliografia fa referència a tres aspectes complementaris en el llibre:

- Bibliografia d'aprofundiment sobre el funcionament dels computadors:
    - A. Prieto, A. Lloris, J.C. Torres. Introducción a la informática. 4a edició. Edit. Mc Graw Hill. 2006.

      Llibre complementari a la matèria. Expandeix el contingut donat en aquest llibre.
    - W. Stallings. Organización y arquitectura de ordenadores. Edit. Prentice Hall. 1996.

      Llibre que entra en profunditat en l'arquitectura del computador. És un llibre per aprofundir en la matèria de fonaments de computadors.
- Bibliografia sobre VHDL.
    - Ashenden. VHDL: professors book.

      És un llibre essencial en l'aprenentatge del VHDL que mostra el llenguatge des dels seus vessants de model i de síntesis.
    - Shojolm. VHDL for designers. Edit. Prentice Hall.

      Es tracta d'un llibre tècnic i pràctic que va per feina i mostra la part sintetitzable del VHDL.
- Manuals d'ús.
    - *Spartan-3 Starter Kit Board. User Guide*. UG130 (v1.1). Xilinx. May 13, 2005.

      És el manual d'ús de la placa que s'ha emprat per provar tots els exemples introduïts en el llibre. Indispensable si es vol expandir l'ús del processador amb altres perifèrics emprant la placa Saprtan-3.

Bibliografia

Annex A1.

TAULA RESUM DEL REPERTORI D'INSTRUCCIONS DEL PROCESSADOR EDUP12

La taula mostra el conjunt d'instruccions emprat en el processador EduP12. Per a cada instrucció la taula mostra la següent informació:

- Joc d'instruccions classificades per tipologia
- Nº de paraules que ocupa de la instrucció
- Activació del registre d'estat davant l'execució de la instrucció
- Número de cicles d'execució de la instrucció.

La nomenclatura seguida en la taula és la següent:

- Rd, Rs: registres destí i font
- X, Y, Z: registres X, Y i Z. Registres específics d'adreçament corresponents als registres R29, R30 i R31, respectivament.
- C, Z, N, V, I: Bits de carreteig, zero, negatiu, excés i interrupció
- K, k: constant en adreçament i constant genèrica
- b: bit d'estat
- PC: Comptador de programes
- SP: apuntador a pila

Taula resum del repertori d'instruccions del processador eduP12

Instrucció			Bits d'estat	Codificació	Paraules	Cicles
Aritmètico-lògiques de doble registre						
MOV Rd, Rs	Rd ← Rs	Moure	-	0000 01sd dddd ssss	1	3
AND Rd, Rs	Rd ← Rd∧Rs	I-lògica	C, Z, N, V	0010 00sd dddd ssss	1	3
ADC Rd, Rs	Rd ← Rd+Rs+C	Suma amb carreteig	C, Z, N, V	0000 11sd dddd ssss	1	3
ADD Rd, Rs	Rd ← Rd+Rs	Suma	C, Z, N, V	0000 10sd dddd ssss	1	3
CP Rd, Rs	-← Rd-Rs	Comparar	C, Z, N, V	0001 10sd dddd ssss	1	3
CPC Rd, Rs	-← Rd-(Rs+C)	Comparar amb carreteig	C, Z, N, V	0001 11sd dddd ssss	1	3
EOR Rd, Rs	Rd ← Rd⊕Rs	Or-exclusiva lògica	C, Z, N, V	0010 10sd dddd ssss	1	3
OR Rd, Rs	Rd ← Rd∨Rs	Or lògica	C, Z, N, V	0010 01sd dddd ssss	1	3
SBC Rd, Rs	Rd ← Rd-(Rs+C)	Resta amb carreteig	C, Z, N, V	0001 01sd dddd ssss	1	3
SUB Rd, Rs	Rd ← Rd-Rs	Resta	C, Z, N, V	0001 00sd dddd ssss	1	3
TST Rd, Rs	- ← Rd∧Rs	Test bit a bit	C, Z, N, V	0000 01sd dddd ssss	1	3
Aritmètico-lògiques de registre simple						
ASR Rd	Rd ← Rd(11)&Rd(11...1), C←Rd(0)	Desplaçament aritmètic a la dreta	C, Z, N, V	0011 --0d dddd 1100	1	3
CLR Rd	Rd ← 0 (EOR Rd, Rd)	Posar a 0	C, Z, N, V	EOR Rd, Rd	1	3
COM Rd	Rd ← (2^n-1)-Rd	Complementar (C1)	C, Z, N, V	0011 --0d dddd 1010	1	3
DEC Rd	Rd ← Rd-1	Decrementar	C, Z, N, V	0011 --0d dddd 1001	1	3
INC Rd	Rd ← Rd+1	Incrementar	C, Z, N, V	0011 --0d dddd 1000	1	3
LSL Rd	Rd ← Rd(10...0) & 0, C←Rd(11) -ADD	Desplaçament lògic a l'esquerra	C, Z, N, V	ADD Rd, Rd	1	3
LSR Rd	Rd ← 0&Rd(11...1), C←Rd(0)	Desplaçament lògic a la dreta	C, Z, N, V	0011 --0d dddd 1101	1	3
NEG Rd	Rd ← 2^n-Rd	Negar (C2)	C, Z, N, V	0011 --0d dddd 1011	1	3
ROL Rd	Rd←Rd(10...0) & C, C←Rd(11)	Rotació a l'esquerra	C, Z, N, V	0011 --0d dddd 1111	1	3
ROR Rd	Rd ← C & Rd(11...1), C ←Rd(0)	Rotació a dreta	C, Z, N, V	0011 --0d dddd 1110	1	3
SWAP Rd	Rd ← Rd(3...0) & Rd(11...4)	Rotació de 4 bits a la dreta sense carry	Z, N, V	0011 –1 d dddd ----	1	3
Operacions amb immediat (doble paraula)						
LDI Rd, k	Rd ← k	Càrrega amb immediat	-	0011 ---d dddd 0000	2	4
ADDI Rd, k	Rd ← Rd+k	Suma amb immediat	C, Z, N, V	0011 ---d dddd 0001	2	4
ANDI Rd, k	Rd ← Rd∧k	I lògica amb immediat	C, Z, N, V	0011 ---d dddd 0010	2	4
CPI Rd, k	- ← Rd-k	Comparar amb immediat	C, Z, N, V	0011 ---d dddd 0110	2	4

Taula resum del repertori d'instruccions del processador eduP12

ORI Rd, k	Rd ← Rd∨k	O lògica amb immediat	C, Z, N, V	0011 ---d dddd 0011	2	4
SBCI Rd, k	Rd ← Rd-(k+C)	Resta amb carreteig i immediat	C, Z, N, V	0011 ---d dddd 0101	2	4
SUBI Rd, k	Rd ← Rd-k	Resta amb immediat	C, Z, N, V	0011 ---d dddd 0100	2	4
TSTI Rd, k	- ← Rd∧k	I-lògica amb immediat sense guarda	C, Z, N, V	0011 ---d dddd 0111	2	4
Instruccions de salt condicional						
BRBC b, k	Si SR(s)=0 aleshores PC ← PC+k+1	Salt si bit b del SR és 1		1111 kkkk kkkk kbbb	1	3
BRBS b, k	Si SR(s)=1 aleshores PC ← PC+k+1	Salt si bit b del SR és 0		1110 kkkk kkkk kbbb	1	3
BRCC k	Si C=0 aleshores PC ← PC+k+1	Salt si carreteig és 0		1111 kkkk kkkk k000	1	3
BRCS k	Si C=1 aleshores PC ← PC+k+1	Salt si carreteig és 1		1110 kkkk kkkk k000	1	3
...BRSH k	Si C=0 aleshores PC ← PC+k+1	Salt si igual o major		1111 kkkk kkkk k000	1	3
...BRLO k	Si C=1 aleshores PC ← PC+k+1	Salt si menor		1110 kkkk kkkk k000	1	3
...BRNZ k, BRNE k	Si Z=0 aleshores PC ← PC+k+1	Salt si diferent		1111 kkkk kkkk k001	1	3
...BRZE k, BREQ k	Si Z=1 aleshores PC ← PC+k+1	Salt si igual		1110 kkkk kkkk k001	1	3
...BRPL k	Si N=0 aleshores PC ← PC+k+1	Salt si positiu		1111 kkkk kkkk k010	1	3
...BRMI k	Si N=1 aleshores PC ← PC+k+1	Salt si negatiu		1110 kkkk kkkk k010	1	3
BRVC k	Si V=0 aleshores PC ← PC+k+1	Salt si excés és 0		1111 kkkk kkkk k011	1	3
BRVS k	Si V=1 aleshores PC ← PC+k+1	Salt si excés és 1		1110 kkkk kkkk k011	1	3
BRID k	Si I=0 aleshores PC ← PC+k+1	Salt si interrupció inhabilitada		1111 kkkk kkkk k100	1	3
BRIE k	Si I=1 aleshores PC ← PC+k+1	Salt si interrupció habilitada		1110 kkkk kkkk k100	1	3
Instruccions de salt incondicional						
ICALL z (o *y* o *x*)	Pila ←PC, PC ← Z, SP ← SP +1	Crida indirecte a subrutina		0110 zyx- ---- 0011	1	4
IJMP z (o *y* o *x*)	PC ← Z	Salt indirecte		0110 zyx- ---- 0010	1	3
RCALL k	Pila ←PC, PC ← PC + k+1, SP ← SP +1	Crida relativa a subrutina		0101 kkkk kkkk kkkk	1	4
RJMP k	PC ← PC + k+1	Salt relatiu		0100 kkkk kkkk kkkk	1	3
RET	SP ← SP-1, PC ← pila	Retorn de subrutina		0110 000- ---- 0100	1	4
RETI	SP ← SP-1, PC ← pila	Retorn d'interrupció		0110 000- ---- 0101	1	4
Instruccions de càrrega amb memòria de programa						
LPM Rd, Z (o Y o X)	Rd ← mem(Z), (o Y o X)	Càrrega indirecta		1100 zyxd dddd 1100	1	4
LPM Rd, +Z (o Y o X)	Z ← Z+1, Rd ← mem(Z), (o Y o X)	Càrrega indirecta amb pre-increment		1100 zyxd dddd 1101	1	4
LPM Rd, -Z (o Y o X)	Z ← Z-1, Rd ← mem(Z), (o Y o X)	Càrrega indirecta amb pre-decrement		1100 zyxd dddd 1111	1	4

Taula resum del repertori d'instruccions del processador eduP12

Instruccions de càrrega/guarda amb memòria de dades						
LDS Rd, K	Rd ← (K)	Càrrega directa (doble paraula)		1100 ---d dddd 0100	2	5
LD Rd, Z (o Y o X)	Rd ← mem(Z), (o Y o X)	Càrrega indirecta		1100 zyxd dddd 0000	1	4
LD Rd, +Z (o Y o X)	Z ← Z+1, Rd ← mem(Z), (o Y o X)	Càrrega indirecta amb pre-increment		1100 zyxd dddd 0001	1	4
LD Rd, -Z (o Y o X)	Z ← Z-1, Rd ← mem(Z), (o Y o X)	Càrrega indirecta amb pre-decrement		1100 zyxd dddd 0011	1	4
LDD Rd, Z+q	Rd ← mem(Z+q)	Càrrega indirecte amb desplaçament en Z		1000 qqqd dddd qqqq	1	4
LDD Rd, Y+q	Rd ← mem(Y+q)	Càrrega indirecte amb desplaçament en Y		1010 qqqd dddd qqqq	1	4
STS K, Rs	mem(K) ← Rs	Guarda directa (doble paraula)		1101 ---d dddd 0100	2	5
ST Z (o Y o X) , Rs	mem(Z) ← Rs, (o Y o X)	Guarda indirecte		1101 zyxd dddd 0000	1	4
ST +Z (o Y o X), Rs	Z ← Z+1, mem(Z) ← Rs, (o Y o X)	Guarda indirecta amb pre-increment		1101 zyxd dddd 0001	1	4
ST -Z (o Y o X), Rs	Z ← Z-1, mem(Z) ← Rs, (o Y o X)	Guarda indirecta amb pre-decrement		1101 zyxd dddd 0011	1	4
STD Z+q, Rs	(Z+q) ← Rs	Guarda indirecte amb desplaçament en Z		1001 qqqd dddd qqqq	1	4
STD Y+q, Rs	(Y+q) ← Rs	Guarda indirecte amb desplaçament en Y		1011 qqqd dddd qqqq	1	4
Instruccions d'entrada/sortida						
IN Rd, PORT	Rd ← PORT	Entrada de port		0111 0AAd dddd AAAA	1	3
OUT PORT, Rs	PORT ← Rs	Sortida a port		0111 1AAd dddd AAAA	1	3
De registre de status						
BSET b	RegistreEstat(b) ← 1	Posar a 1 el bit *b* del registre d'estat		0000 0001 0000 0sss	1	3
BCLR b	RegistreEstat(b) ← 0	Posar a 1 el bit *b* del registre d'estat		0000 0001 0000 1sss	1	3
SEI	RegistreEstat(7) ← 1	Posar a 1 el bit 7 del registre d'estat		0000 0001 0000 0111	1	3
CLI	RegistreEstat(7) ← 0	Posar a 1 el bit 7 del registre d'estat		0000 0001 0000 1111	1	3
Altres instruccions						
PUSH Rs	Pila ← Rs, SP ← SP +1	Posar valor de registre Rs en pila		0110 000d dddd 0111	1	4
POP Rd	SP ← SP-1, Rd ← Pila	Treure de pila a registre Rd		0110 000d dddd 0110	1	4
SAVE	Pila ← SR, SP ← SP +1	Guardar en pila el registre d'estat		0110 100d dddd 0111	1	4
RESTORE	SP ← SP-1, SR ← Pila	Tornar de pila a registre d'estat		0110 100d dddd 0110	1	4
NOP		No fer res		0000 0000 0000 0000	1	3

Annex A2.

CICLE D'INSTRUCCIÓ DEL JOC D'INSTRUCCIONS DEL PROCESSADOR EDUP12.

Aquest apèndix fa una revisió de les instruccions del processador EduP12 agrupades per tipologia d'instrucció.

S'explica el funcionament, afectació al registre d'estat i cicle d'instrucció per a cada instrucció o per tipologia d'instrucció.

La nomenclatura emprada és la mateixa de l'apèndix A1.

A2.1. El cicle d'instrucció en EduP12

El cicle d'instrucció del joc d'instruccions d'EduP12 consta de les fases de cerca de la instrucció i execució, d'acord amb la figura A2.1

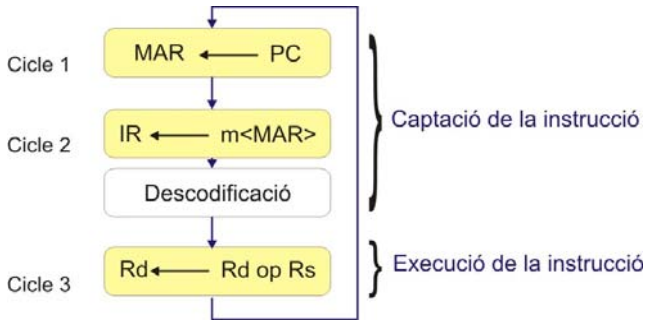


Figura A2.1

La fase de cerca de la instrucció necessita dos cicles de rellotge per fer la cerca, mentre que la fase d'execució pot durar més d'un cicle d'instrucció. Així com la fase de cerca és comuna a totes les instruccions, la fase d'execució és pròpia de cada grup d'instruccions o de cada instrucció.

Fase de cerca de la instrucció

La fase de cerca de la instrucció es realitza en dos cicles de rellotge. Es realitza la següent transferència de dades:

$\Phi 1$: pMAR ← PC

$\Phi 2$: IR ← m<pMAR>, PC ← PC+1
(Descodificació)

La figura A2.2a mostra el cablejat que s'estableix durant la fase de cerca de la instrucció. La figura A2.2b mostra en un diagrama temporal el sincronisme entre els diferents senyals.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

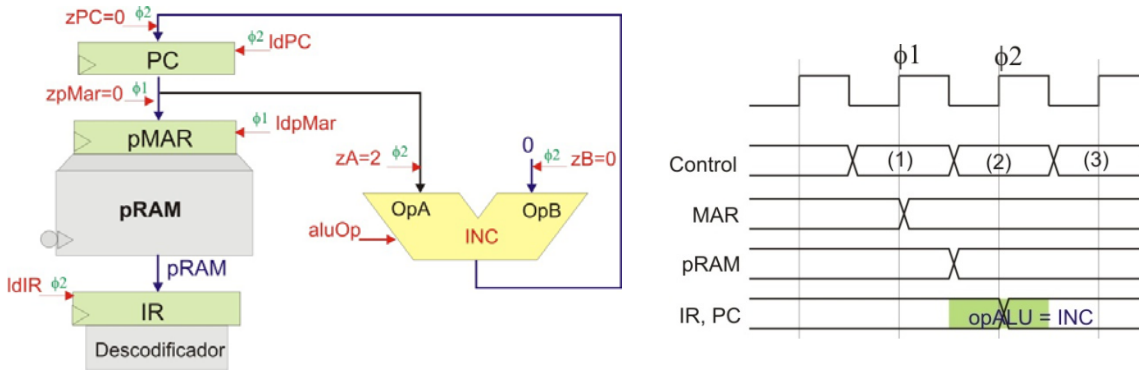


Figura A2.2. a) Fase de cerca en la CPU. b) Diagrama temporal

Donat que la fase de cerca de la instrucció és comuna per a totes les instruccions no s'explicitarà més.

El descodificador s'encarrega de trobar les adreces dels registres, del port d'entrada/sortida i de les constants quan cal treballar amb aquestes dades.

A2.2. Instruccions aritmètico-lògiques de doble registre

A2.2.1 Instrucció ADC: *Add with Carry*

- Format de la instrucció: **ADC Rd, Rs**
- Operació: Sumar el contingut dels dos registre amb el carreteig i guardar el resultat en Rd: Rd ← Rd+(Rs+C)
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan el resultat sigui de signe diferent als operands.

A2.2.2 Instrucció ADD: *Add*

- Format de la instrucció: **ADD Rd, Rs**
- Operació: Sumar el contingut dels dos registres i guardar el resultat en Rd: Rd ← Rd+Rs
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan el resultat sigui de signe diferent als operands.

A2.2.3 Instrucció AND. *Logical AND*

- Format de la instrucció: **AND Rd, Rs**
- Operació: Fer la I-lògica bit a bit de dos registres i guardar el resultat en Rd:

  Rd ← Rd∧Rs
- Canvis en registre d'estat:

C=0.
Z=1 sempre que el resultat sigui 0
N=1 sempre que el bit més significatiu sigui 1.
V=0.

A2.2.4 Instrucció CP: *Compare*

- Format de la instrucció: **CP Rd, Rs**
- Operació: Restar el contingut dels dos registres i no guardar el resultat: Rd ← Rd-Rs
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.2.5 Instrucció CPC: *Compare with Carry*

- Format de la instrucció: **CPC Rd, Rs**
- Operació: Restar el contingut dels dos registres amb el carreteig i no guardar el resultat: Rd ← Rd-(Rs+C)
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.2.6 Instrucció EOR: *Exclusive-Or*

- Format de la instrucció: **EOR Rd, Rs**
- Operació: Fer la or-exclusiva bit a bit de dos registres i guardar el resultat en Rd:

  Rd ← Rd⊕Rs
- Canvis en registre d'estat:

  C=0.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=0.

A2.2.7 Instrucció MOV. *Move*

- Format de la instrucció: **MOV Rd, Rs**
- Operació: Transferir el contingut del registre Rs al Rd i guardar el resultat en Rd:

  Rd ← Rs.
- Canvis en registre d'estat: No afecta.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.2.8 Instrucció OR: *Logical OR*

- Format de la instrucció: **OR Rd, Rs**
- Operació: O-lògica bit a bit de dos registres i guardar el resultat en Rd:

  Rd ← Rd∨Rs
- Canvis en registre d'estat:

  C=0.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=0.

A2.2.9 Instrucció SBC. *Substract with Carry*

- Format de la instrucció: **SBC Rd, Rs**
- Operació: Restar el contingut dels dos registres amb el carreteig i guardar el resultat en Rd: Rd ← Rd-(Rs+C)
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.2.10 Instrucció SUB. *Substract*

- Format de la instrucció: **SUB Rd, Rs**
- Operació: Restar el contingut dels dos registres i guarda el resultat en Rd: Rd ← Rd-Rs
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.2.11 Instrucció TST: *Test*

- Format de la instrucció: **TST Rd, Rs**
- Operació: Fer la I-lògica bit a bit del contingut dels dos registres i no guarda el resultat: Rd ← Rd∧Rs
- Canvis en registre d'estat:

  C=0.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=0.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.2.12 Fase d'execució, genèrica per les instruccions amb doble registre

La fase d'execució realitza l'operació amb les dades dels registres descodificats per la instrucció i dura un cicle de rellotge (figura A2.3). Els senyals addrRd i addrRs venen donats pel descodificador d'instruccions.

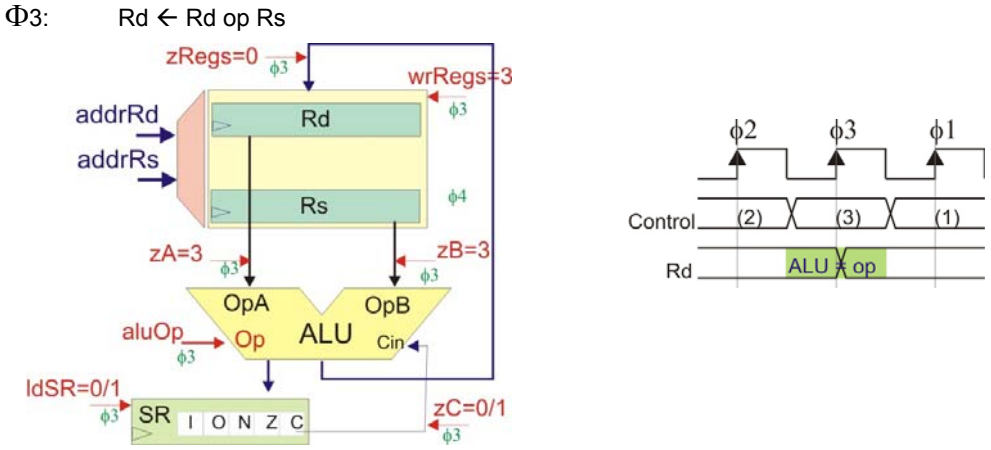


Figura A2.3. a) Fase d'execució d'instruccions aritmètico-lògiques de doble registre. b) Diagrama temporal

Com a excepció la instrucció MOV no actualitza el registre d'estat.

A2.3. Instruccions aritmètico-lògiques de registre simple

A2.3.1 Instrucció ASR: *Arithmetic Shift to Right*

- Format de la instrucció: **ASR Rd**.

  Operació: Desplaçament aritmètic a la dreta: Rd ← Rd(11) & Rd(11...1), C←Rd(0).

- Canvis en registre d'estat:

  C=Rd(0).
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V= C ⊕ N

A2.3.2 Instrucció CLR: *Clear register*

- Format de la instrucció: **CLR Rd**
- Operació: Fer Rd ← 0.
- Implementada com a EOR Rd, Rd.

A2.3.3 Instrucció COM: *Complement*

- Format de la instrucció: **COM Rd**
- Operació: Complement a la base disminuïda: Rd ← 0xFF-Rd
- Canvis en registre d'estat:

  C=1.
  Z=1 sempre que el resultat sigui 0

N=1 sempre que el bit més significatiu sigui 1.
V=0.

A2.3.4 Instrucció DEC: *Decrement*

- Format de la instrucció: **DEC Rd**
- Operació: Decrementar el contingut del registre Rd: Rd ← Rd-1.
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan el resultat sigui de signe diferent a l'operand.

A2.3.5 Instrucció INC: *Increment*

- Format de la instrucció: **INC Rd**

  Operació: Incrementar el contingut del registre Rd: Rd ← Rd+1.
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan el resultat sigui de signe diferent a l'operand.

A2.3.6 Instrucció LSL: *Logical Shift to Left*

- Format de la instrucció: **LSL Rd**
- Operació: Desplaçament lògic a l'esquerra: Rd ← Rd(10...0) & 0, C ←Rd(11).
- Implementada com a ADD Rd, Rd.

A2.3.7 Instrucció LSR: *Logical Shift to Right*

- Format de la instrucció: **LSR Rd**

  Operació: Desplaçament lògic a la dreta: Rd ← 0 & Rd(11...1), C ←Rd(0).
- Canvis en registre d'estat:

  C= Rd(0).
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V= C ⊕ N.

A2.3.8 Instrucció NEG: *Negation*

- Format de la instrucció: **NEG Rd**
- Operació: Complement a la base: Rd ← 0x00-Rd.
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan el resultat sigui de signe diferent als operands.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.3.9 Instrucció ROL: *Rotation Left*

- Format de la instrucció: **ROL Rd**
- Operació: Rotació a l'esquerra amb carry: Rd ← Rd(10...0) & C, C ←Rd(11).
- Canvis en registre d'estat:

  C= Rd(11).
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V= C ⊕ N.

A2.3.10 Instrucció ROR: *Rotation Right*

- Format de la instrucció: **ROR Rd**
- Operació: Rotació a la dreta amb carry: Rd ← C & Rd(11...1), C ←Rd(0).
- Canvis en registre d'estat:

  C= Rd(0).
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V= C ⊕ N.

A2.3.11 Instrucció SWAP: *Rotació 4 bits a la dreta*

- Format de la instrucció: **SWAP Rd**
- Operació: Rotació de 4 bits a la dreta sense carry: Rd ← Rd(3...0) & Rd(11...4).
- Canvis en registre d'estat:

  C es manté.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V= C ⊕ N.

A2.3.12 Fase d'execució, genèrica per les instruccions d'operand simple

La fase d'execució és similar a la fase d'execució de les instruccions aritmètico-lògiques de doble registre (veure figura A2.3). Tan sols cal fer que el senyal de control Z_B valgui 0. També dura un cicle de rellotge:

$\Phi 3$: Rd ← op Rd

A2.4. Instruccions aritmètico-lògiques amb immediats

A2.4.1 Instrucció LDI: *Load Immediat*

- Format de la instrucció: **LDI Rd, k**

  Operació: Càrrega a registre de constant: Rd ← k.
- Canvis en registre d'estat: no el modifica.

A2.4.2 Instrucció ADDI: *Add with Immediat*

- Format de la instrucció: **ADDI Rd, k**

- Operació: Suma de registre amb constant, amb guarda a registre: Rd ← Rd+k.
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan el resultat sigui de signe diferent als operands.

A2.4.3 Instrucció ANDI: *AND with Immediat*

- Format de la instrucció: **ANDI Rd, k**
- Operació: I-lògica bit a bit de registre amb constant, amb guarda a registre:

  Rd ← Rd∧k.
- Canvis en registre d'estat:

  C=0.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=0.

A2.4.4 Instrucció CPI: *Compare with Immediat*

- Format de la instrucció: **CPI Rd, k**
- Operació: Restar de registre un valor constant, sense guarda a registre Rd: Rd ← Rd-k.
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.4.5 Instrucció ORI: *OR with Immediat*

- Format de la instrucció: **ORI Rd, k**
- Operació: O-lògica bit a bit de registre amb constant: Rd ← Rd∨k.
- Canvis en registre d'estat:

  C=0 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=0 quan el resultat sigui de signe diferent a l'operand.

A2.4.6 Instrucció SBCI: *Substract with Immediat with Carry*

- Format de la instrucció: **SBCI Rd, k**
- Operació: Restar de registre un valor constant amb carreteig, amb guarda a registre: Rd ← Rd-(k+C).
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0

Cicle d'instrucció del joc d'instruccions del processador EduP12.

N=1 sempre que el bit més significatiu sigui 1.
V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.4.7 Instrucció SUBI: *Substract with Immediat*

- Format de la instrucció: **SUBI Rd, k**
- Operació: Restar de registre un valor constant, amb guarda a registre Rd: Rd ← Rd-k.
- Canvis en registre d'estat:

  C=1 quan l'operació produeix excés.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=1 quan essent de diferent signe els operands, el resultat sigui de signe diferent al subtrahend.

A2.2.11 Instrucció TSTI: *Test with Immediat*

- Format de la instrucció: **TSTI Rd, k**
- Operació: Fer la I-lògica bit a bit del contingut dels dos registres i no guarda el resultat: Rd ← Rd∧k
- Canvis en registre d'estat:

  C=0.
  Z=1 sempre que el resultat sigui 0
  N=1 sempre que el bit més significatiu sigui 1.
  V=0.

A2.4.8 Fase d'execució, genèrica per les instruccions amb constants

La fase d'execució necessita de dos cicles per executar la instrucció. D'acord amb la figura A2.4 les fases són:

Φ3: pMAR ← PC, PC ← PC+1

Φ4: Rd ← Rd op m<pRAM>

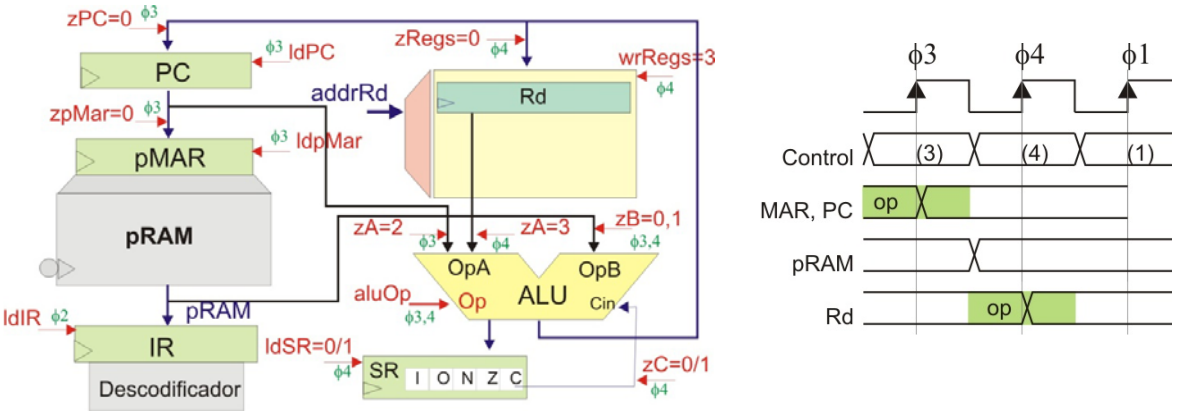


Figura A2.4. a) Fase d'execució (requereix dos cicles de rellotge) per instruccions aritmètico-lògiques amb immediats. b) Diagrama temporal

L'excepció al cablejat de la figura A2.4 ve donada per la instrucció LDI, que no realitza l'operació aritmètica.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.5. Instruccions de salt condicional

A2.5.1 Instrucció BRBS: *Branch if Bit is Set*

- Format de la instrucció: **BRBS b, k**
- Operació: Si ei bit de salt està activat (a 1-lògic): PC ← PC+k+1

  Si no està activat: Continuar en seqüència.
- Canvis en registre d'estat: no el modifica.

A2.5.2 Instrucció BRBS: *Branch if Bit is Clear*

- Format de la instrucció: **BRBC b, k**
- Operació: Si ei bit de salt no està activat (a 0-lògic): PC ← PC+k+1

  Si no està activat: Continuar en seqüència.
- Canvis en registre d'estat: no el modifica.

Aquestes dues instruccions admeten instruccions específiques per preguntar sobre un bit concret. La taula A2.1 resumeix les instruccions acceptades per la màquina:

Instrucció	Operació	Condició sobre bit	Bit (b)
BRCC k	Branch if Carry is Clear	C = 0	0
BRCS k	Branch if Carry is Set	C = 1	0
BRSH k	Branch if Same or Higher	C = 0	0
BRLO k	Branch if Lower	C = 1	0
BRNZ (BRNE) k	Branch if Non Zero	Z = 0	1
BRZE (BREQ) k	Branch if Zero	Z = 1	1
BRPL k	Branch if Plus	N = 0	2
BRMI k	Branch if Minus	N = 1	2
BRVC k	Branch if Overflow is Clear	V = 0	3
BRVS k	Branch if Carry Set	V = 1	3
BRID k	Branch if Interrupt Disabled	I = 0	7
BRIE k	Branch if Interrupt Enabled	I = 1	7

Taula A2.1 Instruccions de salt condicional sobre bit particular

A2.5.3 Fase d'execució

La fase d'execució és realitza en un únic cicle.

$\Phi 3$: SaltCondicional=1 ? PC ← PC+k : continuar

Així, la fase d'execució de la instrucció de salt condicional és continuar en seqüència quan no hi ha salt. Quan hi salt es tracta com si fos un salt incondicional relatiu des de la posició actual, i la seva execució és igual a la de la instrucció RJMP (figura A2.8).

A2.6. Instruccions de salt incondicional

A2.6.1 Instrucció ICALL: *Indirect call*

- Format de la instrucció: **ICALL**

- Operació: Salta a la subrutina que es troba en la posició de memòria indicada pel registre Z, al mateix temps que guarda l'adreça (que es troba en PC) de retorn:

  Pila ← PC, PC ← Z, SP ← SP+1.

- El nombre Z pot ser positiu o negatiu, expressat en complement a la base.
- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La figura A2.5 mostra la fase d'execució. Les taques que es realitzen durant els dos cicles de la fase d'execució són:

  Φ3: dMAR ← SP, SP ← SP-1

  dRAM ← PC (s'executa durant el flanc de baixada del rellotge)

  Φ4: PC ← (Z)

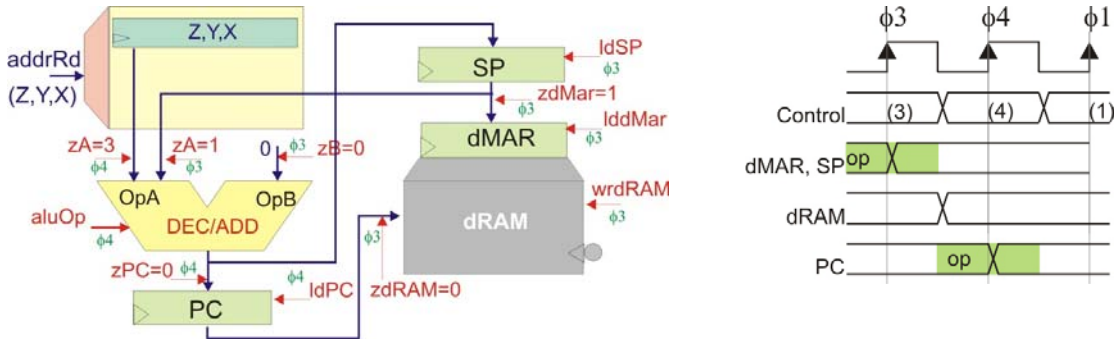


Figura A2.5. a) Fase d'execució de la instrucció ICALL. b) Diagrama temporal

A2.6.2 Instrucció IJMP: *Indirect Jump*

- Format de la instrucció: **IJMP**
- Operació: Salta a la posició de memòria indicada pel registre Z: PC ← Z
- El nombre Z pot ser positiu o negatiu, expressat en complement a la base.
- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La figura A2.6 mostra la fase d'execució. S'executa l'operació:

  Φ3: PC ← (Z)

  Que es correspon amb el segon cicle de la fase d'execució de la instrucció ICALL.

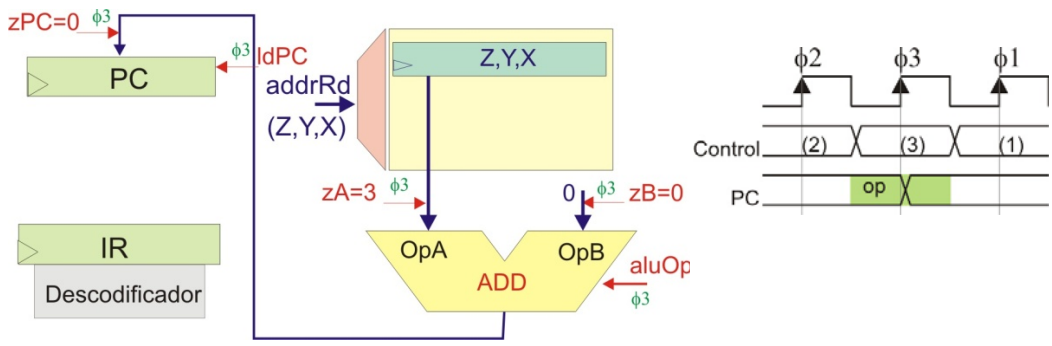


Figura A2.6. a) Fase d'execució del la instrucció IJMP. b) Diagrama temporal

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.6.3 Instrucció RCALL: *Relative Call*

- Format de la instrucció: **RCALL, k**
- Operació: Salta a la subrutina que es troba a k posicions de memòria, al mateix temps que guarda l'adreça (que es troba en PC) de retorn:

  Pila ← PC, PC ← PC+k+1, SP ← SP+1.
- El nombre k pot ser positiu o negatiu, expressat en complement a la base.
- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La figura A2.7 mostra la fase d'execució. La fase d'execució dura dos cicles de rellotge:

  Φ3: dMAR ← SP, SP ← SP-1
  dRAM ← PC (s'executa durant el flanc de baixada del rellotge)

  Φ4: PC ← PC+k

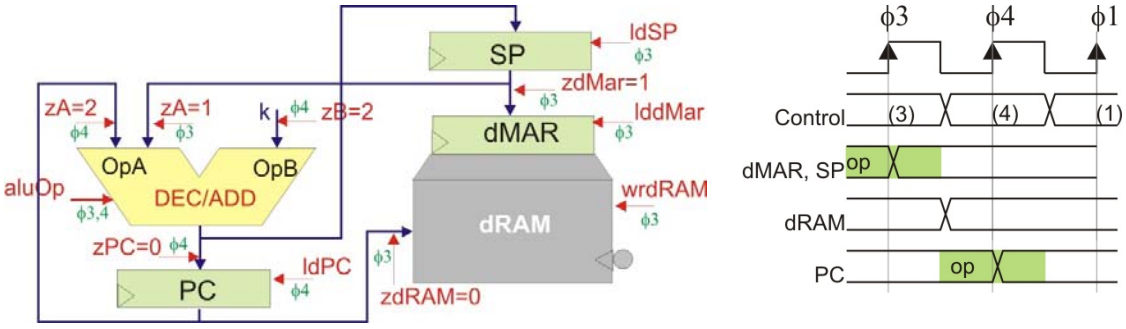


Figura A2.7. a) Fase d'execució de la instrucció RACLL. b) Diagrama temporal

A2.6.4 Instrucció RJMP: *Relative Jump*

- Format de la instrucció: **RJMP k**
- Operació: Salta a k posicions de memòria: PC ← PC+k+1
- El nombre k pot ser positiu o negatiu, expressat en complement a la base.
- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La figura A2.8 mostra la fase d'execució.

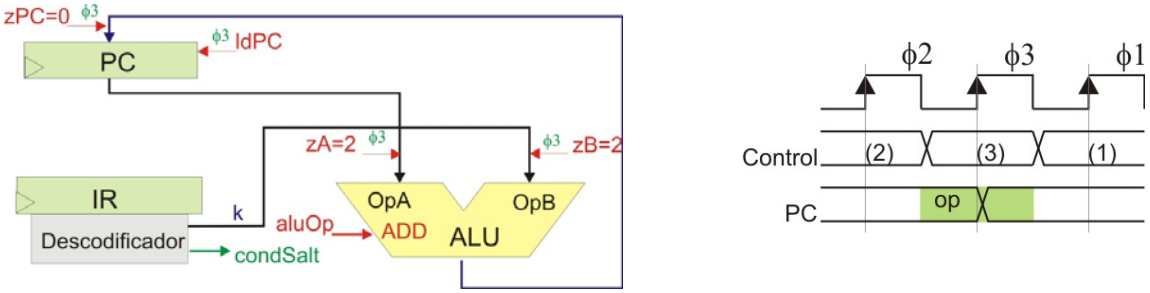


Figura A2.8. a) Fase d'execució per la instrucció RJMP. b) Diagrama temporal

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.6.5 Instrucció RET: *Return*

- Format de la instrucció: **RET**
- Operació: Retorna de subrutina. Cerca l'adreça de retorn de la pila:

  SP ← SP-1, PC ← Pila.
- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La figura A2.6 mostra la fase d'execució.
- La fase d'execució dura dos cicles de rellotge:

  Φ3: dMAR ← (SP+1)

  Φ4: PC ← <dRAM(dMAR)>

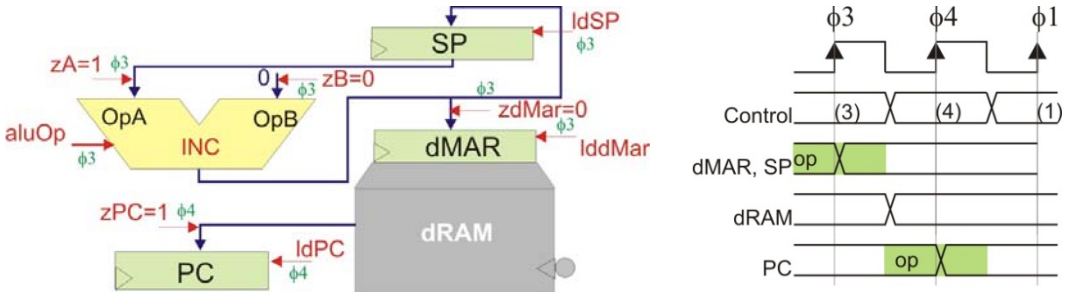


Figura A2.9. a) Fase d'execució de la instrucció RET. b) Diagrama temporal

A2.6.6 Instrucció RETI: *Return from Interrupt*

- Format de la instrucció: **RETI**
- Operació: Retorna de subrutina d'interrupció. Cerca l'adreça de retorn de la pila:

  SP ← SP-1, PC ← Pila, Habilita possibilitat d'interrupció.
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és comuna a la de la instrucció RET. Tan sols s'ha de tenir en compta de reactivar la capacitat d'interrupció donat que se surt del tractament d'una rutina de servei d¡interrupció.

Φ3: dMAR ← (SP+1), ackISR ← 0

Φ4: PC ← <dRAM(dMAR)>

A2.7. Instruccions de càrrega amb memòria de programa

A2.7.1 Instrucció LPM: *Load from Program Memory*

- Format de la instrucció: **LPM Rd, Z** (o Y, o X)
- Operació: La instrucció carrega en un registre una dada de memòria de programa adreçada pel registre Z: Rd ← memòriaPrograma(Z) (o Y o X)
- Canvis en registre d'estat: no el modifica.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

Fase d'execució

- La figura A2.6 mostra la fase d'execució de la instrucció LPM. S'executa en dos cicles de rellotge:

  Φ3: pMAR ← Z (o X o Y)

  Φ4: Rd ← pRAM<pMAR>

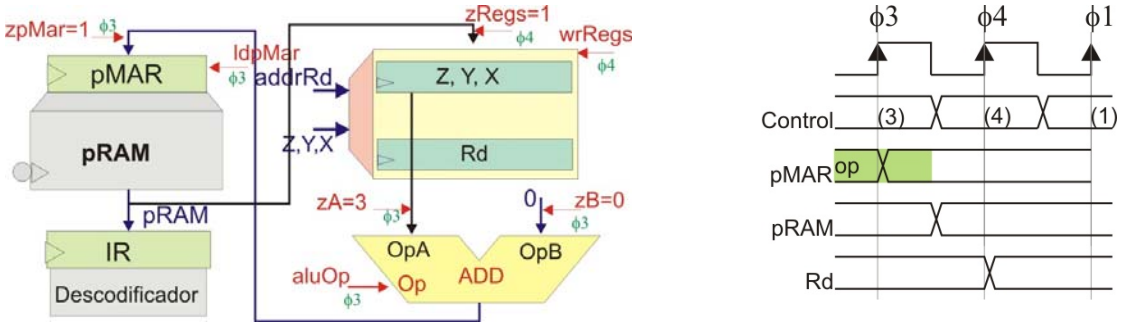


Figura A2.10. a) Fase d'execució d'instruccions LPM. b) Diagrama temporal

A2.7.2 Instrucció LPM +: *Load from Program Memory with pre-Increment*

- Format de la instrucció: **LPM Rd, +Z** (o Y, o X)
- Operació: La instrucció carrega en un registre una dada de memòria de programa adreçada pel registre Z prèviament incrementat. Es realitzen l'operació:

  Rd ← memòriaPrograma(Z+1) (o Y o X)

- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La fase d'execució és la mateixa que per la instrucció LPM (figura A2.10), tan sols realitzant el canvi de l'operació de la UAL en la fase Φ3, incrementant Z (enlloc de la suma).
- La fase d'execució dura, per tant, els dos cicles següents:

  Φ3: pMAR ← (Z+1) (o amb els registres X o Y)

  Φ4: Rd ← pRAM<pMAR>

A2.7.3 Instrucció LPM -: *Load from Program Memory with pre-Decrement*

- Format de la instrucció: **LPM Rd, -Z** (o Y, o X)
- Operació: La instrucció carrega en un registre una dada de memòria de programa adreçada pel registre Z prèviament decrementat. Es realitzen dues operacions:

  Rd ← memòriaPrograma(Z-1) (o Y o X)

- Canvis en registre d'estat: no el modifica.

Fase d'execució

- La fase d'execució és la mateixa que per la instrucció LPM (figura A2.10), tan sols realitzant el canvi de l'operació de la UAL en la fase Φ3, decrementant Z (enlloc de la suma).
- La fase d'execució dura, per tant, els dos cicles següents:

  Φ3: pMAR ← (Z-1) (o amb els registres X o Y)

  Φ4: Rd ← pRAM<pMAR>

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.8. Instruccions de càrrega amb memòria de dades

A2.8.1 Instrucció LDS: *Load Direct from Data Memory*

- Format de la instrucció: **LD Rd, K**
- Operació: La instrucció carrega en un registre la dada de memòria de dades adreçada per K: Rd ← memòriaDades(K)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La figura A2.11 mostra la fase d'execució de LDS. Necessita tres cicles de rellotge, que realitzen les següents operacions:

Φ3: pMAR ← PC, PC ← PC+1

Φ4: dMAR ← k (=<pRAM passant per ALU)

Φ5: Rd ← dRAM<dMAR>

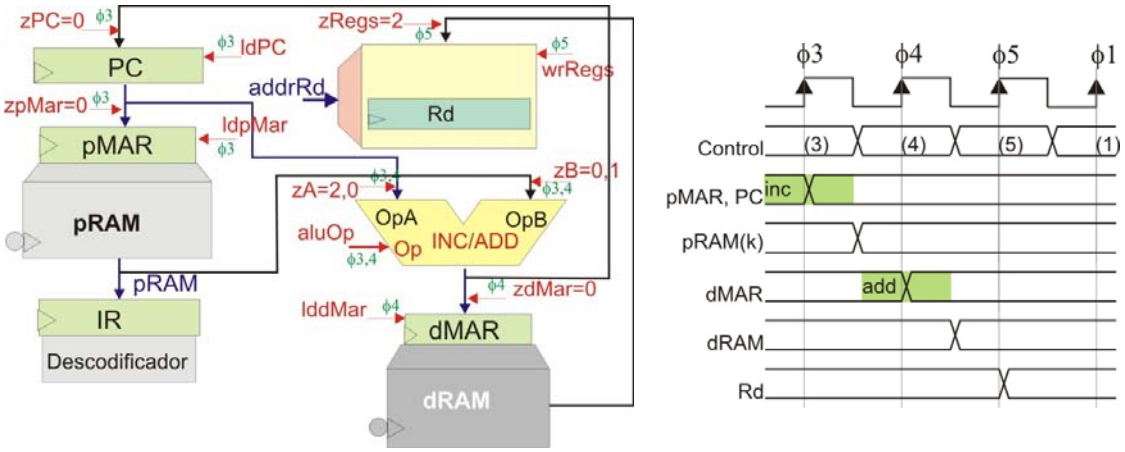


Figura A2.11. a) Fase d'execució per instruccions de càrrega directa. b) Diagrama temporal

A2.8.2 Instrucció LD: *Load Indirect from Data Memory*

- Format de la instrucció: **LD Rd, Z** (o amb els registres Y o X)
- Operació: La instrucció carrega en un registre una dada de memòria de dades adreçada pel registre Z (o Y, o X): Rd ← memòriaDades(Z), (o amb els registres Y o X)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

En la figura A2.12 es mostra la fase d'execució de les instruccions de càrrega indirecta. Es necessiten dos cicles de rellotge i es realitzen les següents operacions:

Φ3: dMAR ← (Z), o dels registres Y o Z.

Φ4: Rd ← dRAM<dMAR>

Cicle d'instrucció del joc d'instruccions del processador EduP12.

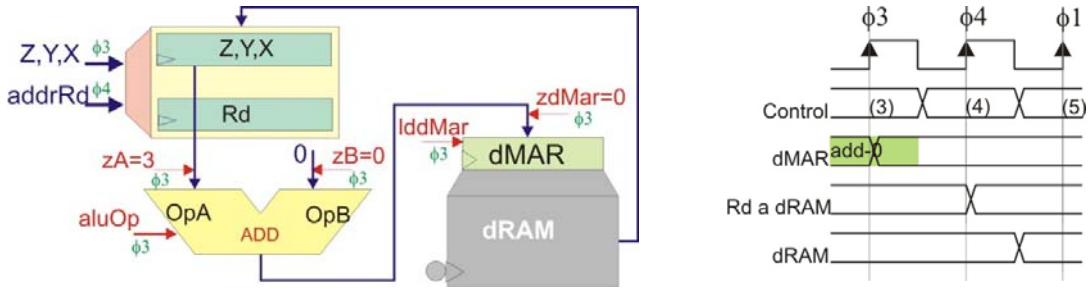


Figura A2.12. a) Fase d'execució per instruccions de càrrega indirecta. b) Diagrama temporal

A2.8.3 Instrucció LD +: *Load Indirect from Data Memory with pre-Increment*

- Format de la instrucció: **LD Rd, +Z** (o amb els registres Y o X)
- Operació: La instrucció carrega en un registre una dada de memòria de dades adreçada pel registre Z (o Y, o X) prèviament incrementat. L'operació és

  Rd ← memòriaDades(Z+1) (o amb els registres Y o X)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és la mateixa que per la instrucció LD (figura A2.12). Donat que l'única diferència és el pre-increment, s'aprofita el pas per la ALU de la dada provinent del registre (Z, Y o X) per incrementar-la.

A2.8.4 Instrucció LD -: *Load Indirect from Data Memory with pre-Decrement*

- Format de la instrucció: **LD Rd, -Z** (o amb els registres Y o X)
- Operació: La instrucció carrega en un registre una dada de memòria de dades adreçada pel registre Z (o Y, o X) prèviament decrementat. L'operació és

  Rd ← memòriaDades(Z-1) (o amb els registres Y o X)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és la mateixa que per la instrucció LD (figura A2.12). Donat que l'única diferència és el pre-decrement, s'aprofita el pas per la ALU de la dada provinent del registre (Z, Y o X) per decrementar-la.

A2.8.5 Instrucció LDD: *Load Indirect from Data Memory with Displacement*

- Format de la instrucció: **LDD Rd, Z+q** (o amb el registre Y)
- Operació: La instrucció carrega en un registre la dada de memòria de dades adreçada pel registre Z desplaçat en q posicions, on q és un natural. Es realitza l'operació:

  Rd ← memòriaDades(Z+q) (o amb Y)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és la mateixa que per la instrucció LD (figura A2.12). Donat que l'única diferència és el desplaçament, s'aprofita el pas per la ALU de la dada provinent del registre (Z, Y o X) per sumar el desplaçament q.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

A2.8.6 Instrucció STS: *Store Direct to Data Memory*

- Format de la instrucció: **STS K, Rs**
- Operació: Guarda el contingut del registre Rs en la posició de memòria de dades adreçada per K: memòriaDades(K) ← Rs
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La figura A2.13 mostra la fase d'execució de LDS. Necessita tres cicles de rellotge, que realitzen les següents operacions:

Φ3: pMAR ← PC, PC ← PC+1

Φ4: dMAR ← k (=<pRAM passant per ALU)

Φ5: dRAM<dMAR> ← Rd

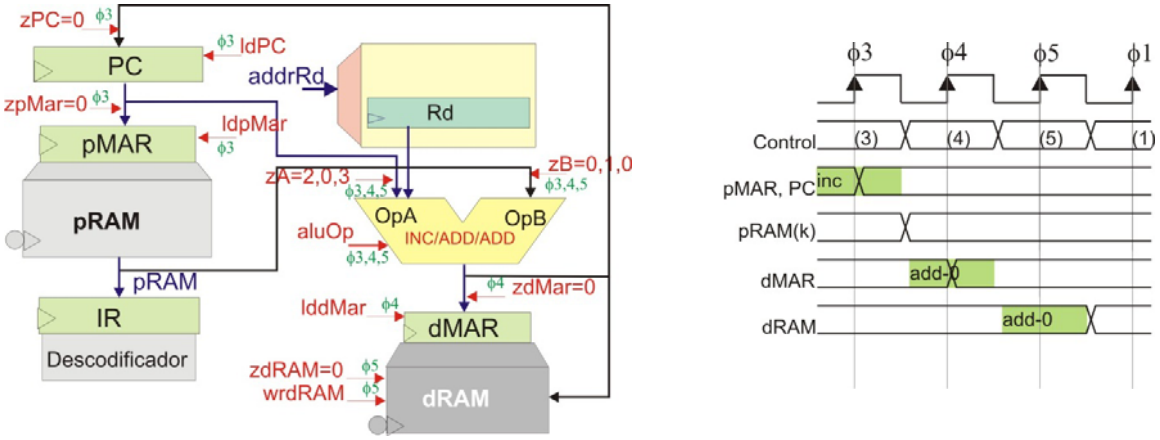


Figura A2.13. a) Fase d'execució de la instrucció de guarda directa. b) Diagrama temporal

A2.8.7 Instrucció ST: *Store Indirect to Data Memory*

- Format de la instrucció: **ST Z, Rs** (o amb els registres Y o X)
- Operació: Guarda el contingut del registre Rs en la posició de memòria de dades adreçada pel registre Z (o Y, o X): memòriaDades(Z) ← Rs, (o amb els registres Y o X)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La figura A2.14 mostra la fase d'execució de LDS. Necessita tres cicles de rellotge, que realitzen les següents operacions:

Φ3: pMAR ← PC, PC ← PC+1

Φ4: dMAR ← k (=<pRAM passant per ALU)

Φ5: Rd ← dRAM<dMAR>

Cicle d'instrucció del joc d'instruccions del processador EduP12.

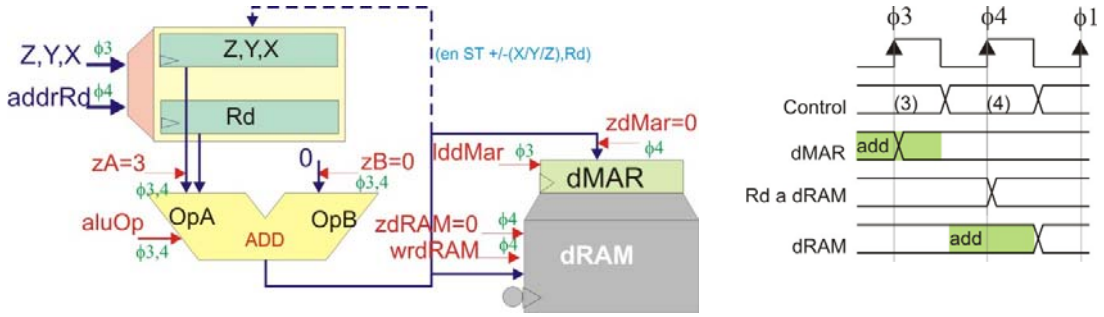


Figura A2.14. a) Fase d'execució per instruccions de guarda indirecta. b) Diagrama temporal

A2.8.8 Instrucció ST +: *Store Indirect to Data Memory with pre-Increment*

- Format de la instrucció: **ST Rs, +Z** (o amb els registres Y o X)
- Operació: Guarda el contingut del registre Rs en la posició de memòria de dades adreçada pel registre Z (o Y, o X) prèviament incrementat. L'operació és

  memòriaDades(Z+1) ← Rs (o amb els registres Y o X)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és la mateixa que per la instrucció LD (figura A2.12). Donat que l'única diferència és el pre-increment, s'aprofita el pas per la ALU de la dada provinent del registre (Z, Y o X) per incrementar-la.

A2.8.9 Instrucció ST -: *Store Indirect to Data Memory with pre-Decrement*

- Format de la instrucció: **ST –Z, Rs** (o amb els registres Y o X)
- Operació: Guarda el contingut del registre Rs en la posició de memòria de dades adreçada pel registre Z (o Y, o X) prèviament decrementat. L'operació és

  memòriaDades(Z-1) ← Rs (o amb els registres Y o X)
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és la mateixa que per la instrucció LD (figura A2.12). Donat que l'única diferència és el pre-decrement, s'aprofita el pas per la ALU de la dada provinent del registre (Z, Y o X) per decrementar-la.

A2.8.10 Instrucció STD: *Store Direct to Data Memory with Displacement*

- Format de la instrucció: **STD Z+q, Rs** (o amb el registre Y)
- Operació: Guarda el contingut del registre Rs en la posició de memòria de dades adreçada pel registre Z desplaçat en q posicions, on q és un natural. Es realitza l'operació: memòriaDades(Z+q) ← Rs (o amb Y)
- Canvis en registre d'estat: no el modifica.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

Fase d'execució

La fase d'execució és la mateixa que per la instrucció LD (figura A2.12). Donat que l'única diferència és la suma del desplaçament, s'aprofita el pas per la ALU de la dada provinent del registre (Z, Y o X) per realitzar-la.

A2.9. Instruccions d'entrada/sortida

A2.9.1 Instrucció IN: *Input from PORT*

- Format de la instrucció: **IN Rd, PORT**
- Operació: Entra a Rd una dada d'un port d'entrada de la memòria d'entrada/sortida: Rd ← PORT. La instrucció conté, com a camps, l'adreça del registre d'entrada i l'adreça de la posició de memòria corresponent al perifèric d'entrada. Ambdues adreces es descodifiquen en la fase Φ2.

  La memòria d'entrada/sortida disposa de 64 posicions
- Canvis en registre d'estat: no es modifica.

Fase d'execució

La figura A2.15 mostra la fase d'execució de la instrucció entrada de port. La UCP s'encarrega, senzillament, de llegir del port d'entrada.

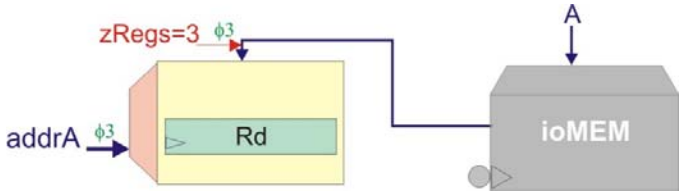


Figura A2.15. a) Fase d'execució per instruccions d'entrada de port.

Els registres de memòria d'entrada actualitzen les dades en flanc de baixada.

Es realitza l'operació:

Φ3: Rd ← ioMem<ioAddr>

A2.9.2 Instrucció OUT: *Output to PORT*

- Format de la instrucció: **OUT PORT, Rs**
- Operació: Posa en un port de sortida de la memòria d'entrada/sortida el contingut de Rs: PORT ← Rs

  La memòria d'entrada/sortida disposa de 64 posicions
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La figura A2.16 mostra la fase d'execució de la instrucció de sortida de port. La UCP s'encarrega, senzillament, d'enviar la dada cap al port de sortida.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

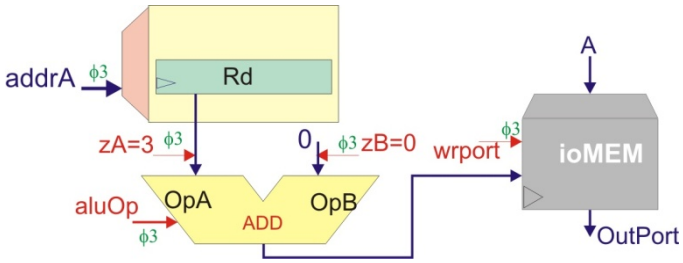


Figura A2.16. a) Fase d'execució per instruccions de sortida a port.

Es realitza l'operació:

Φ3: ioMem<ioAddr> ← Rs

A2.10. Altres instruccions

A2.10.1 Instrucció PUSH: *Push to Stack*

- Format de la instrucció: **PUSH Rs**
- Operació: Guarda en pila, en la posició apuntada per l'apuntador de pila, el contingut de Rd. Després decrementa l'apuntador a pila. Són dues operacions:

  Pila ← Rs
  SP ← SP-1
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase necessita dos cicles de rellotge (figura A2.17).

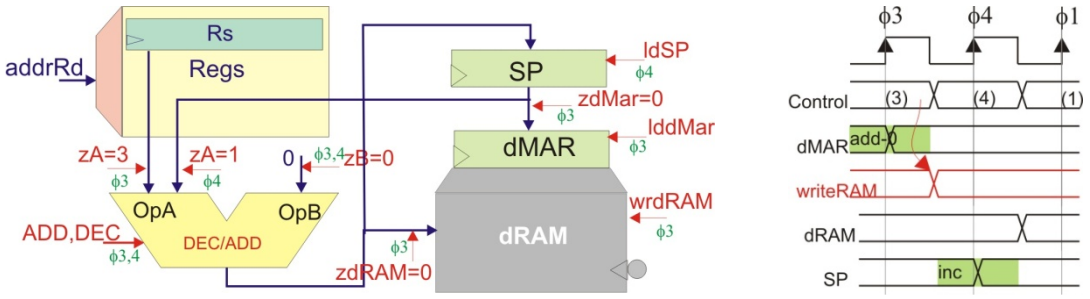


Figura A2.17. a) Fase d'execució per instruccions de guarda a pila. b) Diagrama temporal

A2.10.2 Instrucció POP: *Pop from Stack*

- Format de la instrucció: **POP Rd**
- Operació: Primer incrementa l'apuntador a pila i després recupera a Rd la dada que es troba en la pila. Són dues operacions:

  SP ← SP+1
  Rd ← Pila
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La figura A2.18 mostra la fase d'execució que es realitza en dos cicles de rellotge.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

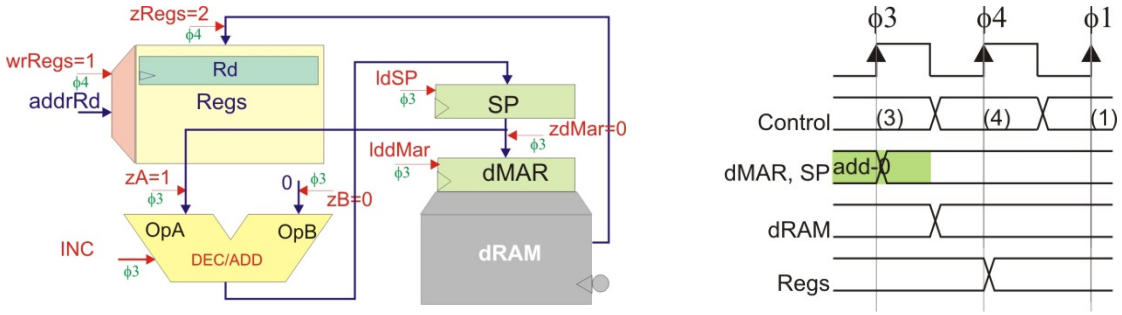


Figura A2.18. a) Fase d'execució per instruccions de recuperació de pila.
b) Diagrama temporal

A2.10.3 Instrucció SAVE: *Save Status Register to Stack*

- Format de la instrucció: **SAVE**
- Operació: Guarda en pila, en la posició apuntada per l'apuntador de pila, el contingut del registre d'estat. Després decrementa l'apuntador a pila. Són dues operacions:

  Pila ← Registre d'estat
  SP ← SP-1
- Canvis en registre d'estat: no el modifica.
- La figura A2.19 mostra l'execució de la instrucció. La cronologia és semblant a la de la instrucció PUSH.
- Recordar que cal guardar el registre d'estat sempre que s'executi una rutina d'interrupció que el pugui modificar.

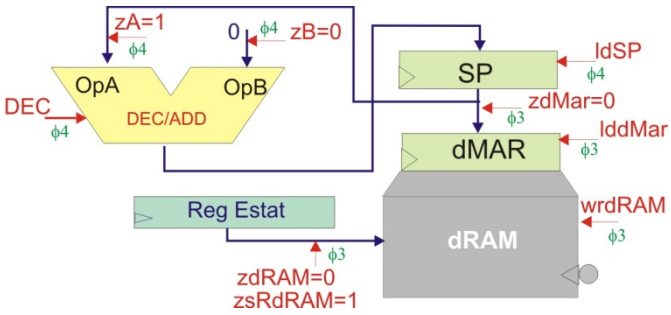


Figura A2.19. Fase d'execució per instruccions de recuperació de pila.

A2.10.4 Instrucció RESTORE: *Restore from Stack*

- Format de la instrucció: **RESTORE**
- Operació: Primer incrementa l'apuntador a pila i després recupera el registre d'estat. Són dues operacions:

  SP ← SP+1
  Registre d'estat ← Pila
- Canvis en registre d'estat: no el modifica.
- La figura A2.20 mostra l'execució de la instrucció. La cronologia és semblant a la de la instrucció POP.

Cicle d'instrucció del joc d'instruccions del processador EduP12.

- Recordar que s'ha de recuperar el registre d'estat sempre que s'hagi executat una rutina d'interrupció en la que s'hagi guardat prèviament.

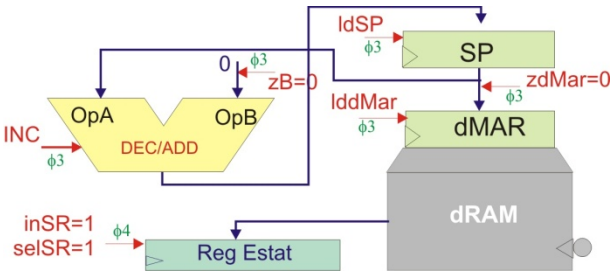


Figura A2.20. Fase d'execució per instruccions de recuperació de pila.

A2.10.5 Instrucció NOP: *No Operation*

- Format de la instrucció: **NOP**
- Operació: No fer res.

  Permet fer temps d'espera sense fer cap operació
- Canvis en registre d'estat: no el modifica.

Fase d'execució

La fase d'execució és no fer res. Deixa passar, per tant, un cicle de rellotge.

Annex A3.

CONSTANTS INICIALS GENÈRIQUES EN EDUP12.

A3.1 Constants inicials genèriques en EduP12.

El fitxer PK_eduP12.vhd conté les constants genèriques emprades en la versió presentada del processador eduP12.

```
--Constants genèriques
constant n: natural:=12;                --ampla de bus
constant mp: natural:=11;               --ampla bus memòria programa
constant md: natural:=8;                --ampla bus memòria dades
constant mio: natural:=6;               --ampla bus adreces ports entrada/sortida
constant nInt: positive:=4;             --ampla bus nombre interrupcions

--adreces ports d'entrada/sortida
constant aPORTA: std_logic_vector(mio-1 downto 0) :="000000";--Port d'entrada simple
constant aPORTB: std_logic_vector(mio-1 downto 0) :="000001";--Port de sortida simple
constant ints: natural :=4;             --Nombre d'interrupcions previstos en Ext0
constant aCTRLIntExt: std_logic_vector(mio-1 downto 0):="000011";--Ctrl port Ext0
constant aSS0: std_logic_vector(mio-1 downto 0):="000100";    --Dígit 0 set-segments
constant aSS1: std_logic_vector(mio-1 downto 0):="000101";    --Dígit 1 set-segments
constant aSS2: std_logic_vector(mio-1 downto 0):="000110";    --Dígit 2 set-segments
constant aSS3: std_logic_vector(mio-1 downto 0):="000111";    --Dígit 3 set-segments
constant aCNT0: std_logic_vector(mio-1 downto 0)   :="010000";--Comptador timer0
constant aCTRL0: std_logic_vector(mio-1 downto 0)  :="010001";--Registre de control timer0
constant aCMPA0: std_logic_vector(mio-1 downto 0) :="010010";--Comparador A del timer0
constant aCMPB0: std_logic_vector(mio-1 downto 0) :="010011";--Comparador B del timer0
constant ADDRCTRLUSARTLITE: std_logic_vector(5 downto 0):="100000";--Control UsartLite
constant ADDRRXDUSARTLITE: std_logic_vector(5 downto 0)  :="100001";--RxD UsartLite
constant ADDRTXDUSARTLITE: std_logic_vector(5 downto 0)  :="100010";--TxD UsartLite
```

A3.2 Relació d'adreces entre assemblador i codi VHDL d'EduP12.

S'adjunta l'equivalent de noms d'adreces que existeix entre l'assemblador i el codi VHDL. Aquesta taula només és necessària si es vol treballar sobre el processador en el codi VHDL.

Comentari	Nom en codi EduP12	Nom en assemblador	Adreça
Port d'entrada	aPORTA	PORTA	000000
Port de sortida	aPORTB	PORTB	000001
Port d'entrada amb interrupcions	aPORTE	PORTE	000010
Control port entrada amb interrupc.	aCTRLIntExt	EXT0ctrl	000011
7-segments 0	aSS0	SS0	000100

Constants inicials genèriques en EduP12.

7-segments 1	aSS1	SS1	000101
7-segments 2	aSS2	SS2	000110
7-segments 3	aSS3	SS3	000111
Comptador del timer 0	aCNT0	TIMER0	010000
Control del timer 0	aCTRL0	TIMER0ctrl	010001
Comparador A del timer 0	aCMPA0	TIMER0cmpA	010010
Comparador B del timer 0	aCMPB0	TIMER0cmpB	010011
Control de la Uart 0	ADDRCTRLUSARTLITE	UART0ctrl	100000
Registre RxD de la Uart 0	ADDRRXDUSARTLITE	UART0RxD	100001
Registre RxD de la Uart 1	ADDRTXDUSARTLITE	UART0TxD	100010

A3.3 assignació inicial d'adreces d'interrupció.

A continuació es llista l'assignació inicial d'adreces de rutina de servei d'interrupció. Aquets llistat s'ha d'acoblar en el fitxer *Ports.vhd* cada cop que s'adjunta una nova interrupció en el processador.

```
interrupts <= (others=>'0');                --Int. 0 (main) i ...
interrupts(8 downto 1) <= qIntFromIntPort; --Int. 8 a 1: Interrupcions ports entrada
interrupts(9)<= ovfi0;                      --Int. 9: Interrupció per overflow del timer0
interrupts(10)<= ocai0;                     --Int. 10: Interrupció de comparació A del timer0
interrupts(13)<= ocbi0;                     --Int. 11: Interrupció de comparació B del timer0
interrupts(14)<= ITxD;                      --Int. 14: Interrupció TxD
interrupts(15)<= IRxD;                      --Int. 15: Interrupció RxD
```

Nota: Inicialment les interrupcions 12 i 13 no es troben assignades.

Annex A4.

PERSONALITZACIÓ DE L'ENTRADA/SORTIDA D'EDUP12 SOBRE LA PLACA SPARTAN3 DE DIGILENT.

Pins	Connexió en FPGA	Funció en placa Spartan-3
Pins generals de funcionament		
RST	l14	Botó 3 (UserReset) (13)
CLK	t9	
Pins emprats en el PORTA (entrada)		
A<11>	nc	-
A<10>	nc	-
A<9>	nc	-
A<8>	c8	Expansor A2, pin 12 (20)
A<7>	k13	Commutador 7 (11)
A<6>	k14	Commutador 6 (11)
A<5>	j13	Commutador 5 (11)
A<4>	j14	Commutador 4 (11)
A<3>	h13	Commutador 3 (11)
A<2>	h14	Commutador 2 (11)
A<1>	g12	Commutador 1 (11)
A<0>	f12	Commutador 0 (11)
Pins emprats en el PORTB (sortida)		
B<11>	b4	Expansor A2, pin 17 (20)
B<10>	nc	-
B<9>	nc	-
B<8>	nc	-
B<7>	p11	Led7 (12)
B<6>	p12	Led6 (12)
B<5>	n12	Led5 (12)
B<4>	p13	Led4 (12)
B<3>	n14	Led3 (12)
B<2>	l12	Led2 (12)
B<1>	p14	Led1 (12)
B<0>	k12	Led0 (12)
Pins d'entrada amb interrupció. Es pot emprar com a port d'entrada PORTE quan no s'empren interrupcions		
nINT<3>	l13	Botó 2 (13)
nINT<2>	m14	Botó 1 (13)
nINT<1>	m13	Botó 0 (13)
nINT<0>	c7	Expansor A2, pin 10 (20)

Personalització de l'entrada/sortida d'EduP12 sobre la placa Spartan3 de Digilent.

Pins utilitzats en el set-segments		
SS_SEL<3>	e13	Selecció ss 3 (10)
SS_SEL<2>	f14	Selecció ss 2 (10)
SS_SEL<1>	g14	Selecció ss 1 (10)
SS_SEL<0>	d14	Selecció ss 0 (10)
SS_DAT<7>	p16	Led ss pd (10)
SS_DAT<6>	n16	Led ss G (10)
SS_DAT<5>	f13	Led ss F (10)
SS_DAT<4>	r16	Led ss E (10)
SS_DAT<3>	p15	Led ss D (10)
SS_DAT<2>	n15	Led ss C (10)
SS_DAT<1>	g13	Led ss B (10)
SS_DAT<0>	e14	Led ss A (10)
Pins utilitzats pel timer		
TOV0	e6	Expansor A2, pin 4 (20)
OC0a	c5	Expansor A2, pin 6 (20)
OC0b	c6	Expansor A2, pin 8 (20)
Pins de comunicació UART		
RxD	t13	Connector RxD DB9 (6)
TxD	r13	Connector TxD DB9 (6)

Nota al port de pins d'entrada amb interrupció:

El port intEXT0 que està connectat als pins *nINT<>* està predefinit com a un port d'entrada de 4 bits. Nogensmenys, donat que en la implementació en la placa Spartan3 es connecta a tres polsadors i a una entrada del port d'expansió, es pot fer servir com a un port d'entrada per a aquests dispositius i, aleshores, agafa el nom de PORTE, convertint-se (quan no s'empren interrupcions) en un port d'entrada de 4 bits.

Annex A5.

CODIS VHDL DELS PROGRAMES.

Es relacionen els codis màquina dels programes assemblador presentats en el capítol 9. Es presenten en el format que s'han emprat per a ser carregats (com a codi VHDL) en memòria en la placa Spartan.

A5.1 Codi VHDL exemple 1.

```
library ieee;
use ieee.std_logic_1164.all;
use work.PK_eduP12.all;
package exem9_1 is
    type mostra is record
        dir: integer range 0 to 2**mp-1;
        valor: std_logic_vector(15 downto 0);
    end record;
    type contingut_mem is array (natural range <>) of mostra;
    constant test: contingut_mem:=(
--initial addresses are for interrupts
--address 0 --> jump to main program
-----------------------------------------------------------------
(0, "0101000000000111"), -- 0x5007 - rcall 7
(1, "0111000100000010"), -- 0x7102 - in r16 000010
(2, "0011000100000010"), -- 0x3102 - andi r16 0x002
(3, "0000000000000010"), -- 0x2
(4, "1110111111011001"), -- 0xEFD9 - brze -5
(5, "0111000100000000"), -- 0x7100 - in r16 000000
(6, "0111100100000001"), -- 0x7901 - out 000001 r16
(7, "0100111111111000"), -- 0x4FF8 - rjmp -8
(8, "0011000100000000"), -- 0x3100 - ldi r16 0xfff
(9, "0000111111111111"), -- 0xFFF
(10, "0011000100001001"), -- 0x3109 - dec r16
(11, "1111111111110001"), -- 0xFFF1 - brnz -2
(12, "0110000000000100"), -- 0x6004 - ret
-----------------------------------------------------------------
( 2047, "0000000000000000")   -- nop
);
end;
```

Codis VHDL dels programes.

A5.2 Codi VHDL exemple 1 amb interrupcions.

```
library ieee;
use ieee.std_logic_1164.all;
use work.PK_eduP12.all;
package exem9_1b is
    type mostra is record
        dir: integer range 0 to 2**mp-1;
        valor: std_logic_vector(15 downto 0);
    end record;
    type contingut_mem is array (natural range <>) of mostra;
    constant test: contingut_mem:=(
--initial adresses are for interrupts
--adress 0 --> jump to main program
-----------------------------------------------------------------
(0, "0100000000011111"), -- 0x401F - rjmp 31
--
(4, "0100000000101101"), -- 0x402D - rjmp 45
(5, "0000000000000000"), -- 0x0 - nop
--
(32, "0011000100000000"), -- 0x3100 - ldi r16 0x008
(33, "0000000000001000"), -- 0x8
(34, "0111100100000011"), -- 0x7903 - out 000011 r16
(35, "0000000100000111"), -- 0x107 - sei
(36, "0000000000000000"), -- 0x0 - nop
(37, "0100111111111110"), -- 0x4FFE - rjmp -2
(38, "0000000000000000"), -- 0x0 - nop
--
(50, "0111000100000000"), -- 0x7100 - in r16 000000
(51, "0111100100000001"), -- 0x7901 - out 000001 r16
(52, "0110000000000101"), -- 0x6005 - reti
(53, "0000000000000000"), -- 0x0 - nop
-----------------------------------------------------------------
( 2047, "0000000000000000")   -- nop
);
end;
```

Codis VHDL dels programes.

A5.3 Codi VHDL exemple 2.

```
library ieee;
use ieee.std_logic_1164.all;
use work.PK_eduP12.all;
package exem9_3 is
    type mostra is record
        dir: integer range 0 to 2**mp-1;
        valor: std_logic_vector(15 downto 0);
    end record;
    type contingut_mem is array (natural range <>) of mostra;
    constant test: contingut_mem:=(
--initial adresses are for interrupts
--adress 0 --> jump to main program
-----------------------------------------------------------------
(0, "0100000000011111"), -- 0x401F - rjmp 31
--
(32, "0011000000000000"), -- 0x3000 - ldi r0 0x3f
(33, "0000000000111111"), -- 0x3F
(34, "0011000000010000"), -- 0x3010 - ldi r1 0x6
(35, "0000000000000110"), -- 0x6
(36, "0101000000001101"), -- 0x500D - rcall 13
(37, "0100111111111110"), -- 0x4FFE - rjmp -2
--
(50, "0111100000000111"), -- 0x7807 - out 000111 r0
(51, "0111100000010100"), -- 0x7814 - out 000100 r1
(52, "0101000000101111"), -- 0x502F - rcall 47
(53, "0111100000000100"), -- 0x7804 - out 000100 r0
(54, "0111100000010101"), -- 0x7815 - out 000101 r1
(55, "0101000000101100"), -- 0x502C - rcall 44
(56, "0111100000000101"), -- 0x7805 - out 000101 r0
(57, "0111100000010110"), -- 0x7816 - out 000110 r1
(58, "0101000000101001"), -- 0x5029 - rcall 41
(59, "0111100000000110"), -- 0x7806 - out 000110 r0
(60, "0111100000010111"), -- 0x7817 - out 000111 r1
(61, "0101000000100110"), -- 0x5026 - rcall 38
(62, "0110000000000100"), -- 0x6004 - ret
(63, "0000000000000000"), -- 0x0 - nop
--
(100, "0011000100010000"), -- 0x3110 - ldi r17 0xc3
(101, "0000000011000011"), -- 0xC3
(102, "0011000100000000"), -- 0x3100 - ldi r16 0xfff
(103, "0000111111111111"), -- 0xFFF
(104, "0011000100001001"), -- 0x3109 - dec r16
(105, "1111111111110001"), -- 0xFFF1 - brnz -2
(106, "0011000100011001"), -- 0x3119 - dec r17
(107, "1111111111010001"), -- 0xFFD1 - brnz -6
(108, "0110000000000100"), -- 0x6004 - ret
(109, "0000000000000000"), -- 0x0 - nop
-----------------------------------------------------------------
( 2047, "0000000000000000")   -- nop
);
end;
```

Codis VHDL dels programes.

A4.4 Codi VHDL exemple 3.

```
library ieee;
use ieee.std_logic_1164.all;
use work.PK_eduP12.all;
package exem9_2 is
    type mostra is record
        dir: integer range 0 to 2**mp-1;
        valor: std_logic_vector(15 downto 0);
    end record;
    type contingut_mem is array (natural range <>) of mostra;
    constant test: contingut_mem:=(
--initial adresses are for interrupts
--adress 0 --> jump to main program
-----------------------------------------------------------------
(0, "0100000000011111"), -- 0x401F - rjmp 31
--
(4, "0100001111100011"), -- 0x43E3 - rjmp 995
--
(15, "0100010000111100"), -- 0x443C - rjmp 1084
--
(32, "0011000111010000"), -- 0x31D0 - ldi r29 1616
(33, "0000011001010000"), -- 0x650
(34, "1100001100001100"), -- 0xC30C - lpm r16 r29
(35, "0111100100000111"), -- 0x7907 - out 000111 r16
(36, "0111100100000110"), -- 0x7906 - out 000110 r16
(37, "0111100100000101"), -- 0x7905 - out 000101 r16
(38, "0111100100000100"), -- 0x7904 - out 000100 r16
(39, "0011000100000000"), -- 0x3100 - ldi r16 0x10c
(40, "0000000100001100"), -- 0x10C
(41, "0111110100000000"), -- 0x7D00 - out 100000 r16
(42, "0011000100000000"), -- 0x3100 - ldi r16 0x008
(43, "0000000000001000"), -- 0x8
(44, "0111100100000011"), -- 0x7903 - out 000011 r16
(45, "0000000100000111"), -- 0x107 - sei
(46, "0010100000000000"), -- 0x2800 - clr r0
(47, "0011000000000110"), -- 0x3006 - cpi r0 0
(48, "0000000000000000"), -- 0x0
(49, "1110111111101001"), -- 0xEFE9 - brze -3
(50, "0101000000110001"), -- 0x5031 - rcall 49
(51, "0100111111111010"), -- 0x4FFA - rjmp -6
(52, "0000000000000000"), -- 0x0 - nop
--
(100, "1100000000010100"), -- 0xC014 - lds r1 0
(101, "0000000000000000"), -- 0x0
(102, "0011000000010010"), -- 0x3012 - andi r1 0x00f
(103, "0000000000001111"), -- 0xF
(104, "0011000111010000"), -- 0x31D0 - ldi r29 1600
(105, "0000011001000000"), -- 0x640
(106, "0000100111010001"), -- 0x9D1 - add r29 r1
(107, "1100001000011100"), -- 0xC21C - lpm r1 r29
(108, "0111100000010100"), -- 0x7814 - out 000100 r1
(109, "1100000000010100"), -- 0xC014 - lds r1 0
(110, "0000000000000000"), -- 0x0
```

Codis VHDL dels programes.

```
(111, "0011001000010000"), -- 0x3210 - swap r1
(112, "0011000000010010"), -- 0x3012 - andi r1 0x00f
(113, "0000000000001111"), -- 0xF
(114, "0011000111010000"), -- 0x31D0 - ldi r29 1600
(115, "0000011001000000"), -- 0x640
(116, "0000100111010001"), -- 0x9D1 - add r29 r1
(117, "1100001000011100"), -- 0xC21C - lpm r1 r29
(118, "0111100000010101"), -- 0x7815 - out 000101 r1
(119, "0110000000000100"), -- 0x6004 - ret
(120, "0000000000000000"), -- 0x0 - nop
--
(1000, "0110100000000111"), -- 0x6807 - save
(1001, "0111000100000000"), -- 0x7100 - in r16 000000
(1002, "0111100100000001"), -- 0x7901 - out 000001 r16
(1003, "0111110100000010"), -- 0x7D02 - out 100010 r16
(1004, "1101000100000100"), -- 0xD104 - sts 0 r16
(1005, "0000000000000000"), -- 0x0
(1006, "0011000000001000"), -- 0x3008 - inc r0
(1007, "0110100000000110"), -- 0x6806 - restore
(1008, "0110000000000101"), -- 0x6005 - reti
(1009, "0000000000000000"), -- 0x0 - nop
--
(1100, "0110100000000111"), -- 0x6807 - save
(1101, "0111101100000001"), -- 0x7501 - in r16 100001
(1102, "0111100100000001"), -- 0x7901 - out 000001 r16
(1103, "1101000100000100"), -- 0xD104 - sts 0 r16
(1104, "0000000000000000"), -- 0x0
(1105, "0011000000001000"), -- 0x3008 - inc r0
(1106, "0110100000000110"), -- 0x6806 - restore
(1107, "0110000000000101"), -- 0x6005 - reti
(1108, "0000000000000000"), -- 0x0 - nop
--
(1600, "0000000000111111"), -- 0x3F - .dw 0x3f
(1601, "0000000000000110"), -- 0x6 - .dw 0x6
(1602, "0000000001011011"), -- 0x5B - .dw 0x5b
(1603, "0000000001001111"), -- 0x4F - .dw 0x4f
(1604, "0000000001100110"), -- 0x66 - .dw 0x66
(1605, "0000000001101101"), -- 0x6D - .dw 0x6d
(1606, "0000000001111101"), -- 0x7D - .dw 0x7d
(1607, "0000000000000111"), -- 0x7 - .dw 0x7
(1608, "0000000001111111"), -- 0x7F - .dw 0x7f
(1609, "0000000001100111"), -- 0x67 - .dw 0x67
(1610, "0000000001011111"), -- 0x5F - .dw 0x5f
(1611, "0000000001111100"), -- 0x7C - .dw 0x7c
(1612, "0000000001011000"), -- 0x58 - .dw 0x58
(1613, "0000000001011110"), -- 0x5E - .dw 0x5e
(1614, "0000000001111001"), -- 0x79 - .dw 0x79
(1615, "0000000001110001"), -- 0x71 - .dw 0x71
(1616, "0000000010000000"), -- 0x80 - .dw 0x80
------------------------------------------------------------
( 2047, "0000000000000000")   -- nop
);
end;
```

Resum

Introducció als fonaments dels computadors amb EduP12 és una introducció pràctica als fonaments dels computadors basada en el processador EduP12, un processador RISC de 12 bits d'arquitectura Harvard, que disposa d'un bon repertori d'instruccions i admet un conjunt divers de modes d'adreçament, fet que el fa adequat per a introduir el lector en els fonaments del computador. EduP12 és un processador de codi obert descrit en VHDL i preparat per a ser implementat sobre FPGA.

El llibre complementa l'estudi del processador amb una primera part que és una introducció als fonaments del computador, basada en una introducció als sistemes de numeració emprats en els sistemes digitals, una introducció als sistemes digitals i al funcionament del processador. Aquesta primera part és de lectura obligada per a tot lector que no tingui un coneixement bàsic del computador si després vol entendre el funcionament de qualsevol processador i, en concret, d'EduP12.

La segona part del llibre descriu en profunditat el processador EduP12. S'introdueix la CPU i la seva arquitectura, el codi màquina i modes d'adreçament i l'ús de perifèrics en el sistema. En capítols posteriors s'aprofundeix en l'arquitectura del processador amb la introducció d'interrupcions i la personalització del processador introduint aquells perifèrics que es requereixi en cada aplicació.