Estructura bàsica d'un computador

El processador com a generalització de les màquines algorísmiques

Lluís Ribas i Xirgo

PID_00163596

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció 5

Objectius 6

1. Màquines d'estats 7
1.1. Màquines d'estats finits com a controladors 8
1.1.1. Procediment de materialització de controladors amb circuits seqüencials 10
1.2. Màquines d'estats finits esteses 16
1.3. Màquines d'estats-programa 26
1.4. Màquines d'estats algorísmiques 41

2. Màquines algorísmiques 52
2.1. Esquemes de càlcul 52
2.2. Esquemes de càlcul segmentats 55
2.3. Esquemes de càlcul amb recursos compartits 56
2.4. Materialització d'esquemes de càlcul 57
2.5. Representació de màquines algorísmiques 60
2.6. Materialització de màquines algorísmiques 61
2.7. Cas d'estudi 65

3. Arquitectura bàsica d'un computador 70
3.1. Màquines algorísmiques generalitzables 71
3.1.1. Exemple de màquina algorísmica general 72
3.2. Màquina elemental 79
3.2.1. Màquines algorísmiques d'unitats de control 81
3.2.2. Màquines algorísmiques microprogramades 83
3.2.3. Una màquina amb arquitectura de Von Neumann 86
3.3. Processadors 91
3.3.1. Microarquitectures 91
3.3.2. Microarquitectures amb *pipelines* 92
3.3.3. Microarquitectures paral·leles 93
3.3.4. Microarquitectures amb CPU i memòria diferenciades 94
3.3.5. Processadors de propòsit general 96
3.3.6. Processadors de propòsit específic 96
3.4. Computadors 98
3.4.1. Arquitectura bàsica 98
3.4.2. Arquitectures orientades a aplicacions específiques 102

Resum 104

Exercicis d'autoavaluació 107

Solucionari 112

Glossari 133

Bibliografia 137

Introducció

En el mòdul anterior s'ha vist que els circuits seqüencials serveixen per a detectar l'ordre temporal d'una sèrie d'esdeveniments (és a dir, seqüències de bits) i que també, a partir de les seves sortides, poden controlar tot tipus de sistemes. Com que la seva funcionalitat es pot representar amb un graf d'estats, sovint es parla de **màquines d'estats**. Així doncs, s'ha vist com es poden construir circuits seqüencials a partir de les representacions de les màquines d'estats corresponents.

De fet, la majoria de sistemes digitals, incloent-hi els computadors, són sistemes seqüencials complexos compostos, finalment, per una munió de màquines d'estats i d'elements de processament de dades. O, dit d'una altra manera, els sistemes seqüencials complexos estan constituïts per un conjunt d'**unitats de control** i d'**unitats de processament**, que materialitzen les màquines d'estats i els blocs de processament de dades respectivament.

En aquest mòdul s'aprendrà a abordar el problema d'analitzar i sintetitzar circuits seqüencials de manera sistemàtica. De fet, molts dels problemes que es proposen en circuits seqüencials del tipus "quan es compleixi una condició sobre uns determinats valors d'entrada, estant en un estat determinat, es donarà una sortida que respon a uns càlculs concrets i que pot correspondre a un estat diferent del primer" es poden resoldre millor si es pensa només en termes de màquines d'estats que no pas si es prova de fer el disseny de les unitats de processament i de control per separat. Passa el mateix en casos una mica més complexos en què la frase anterior comença amb "si es compleix" o "mentre es compleixi", o si els càlculs impliquen força passos.

Finalment, això ajudarà a comprendre com es construeixen els processadors i com treballen, que són el component principal de tot computador.

El mòdul s'acaba mostrant les arquitectures generals dels computadors de manera que serveixi de base per a comprendre el funcionament d'aquestes màquines.

Objectius

La finalitat d'aquest mòdul és que s'aprenguin els conceptes fonamentals per a l'anàlisi i la síntesi dels diversos components d'un computador vist com a sistema digital seqüencial complex. A continuació, s'enumeren els objectius particulars que s'han d'assolir havent acabat l'estudi d'aquest mòdul.

1. Conèixer els diversos models de les màquines d'estats i de les arquitectures de controlador amb camí de dades.

2. Haver adquirit una experiència bàsica en la tria del model de màquina d'estats més adequat per a la resolució d'un problema concret.

3. Saber analitzar circuits seqüencials i extraure'n el model corresponent.

4. Ser capaç de dissenyar circuits seqüencials a partir de grafs de transicions d'estats.

5. Conèixer el procés de disseny de màquines algorísmiques i entendre els criteris que l'afecten.

6. Tenir la capacitat de dissenyar màquines algorísmiques a partir de programes senzills.

7. Haver après el funcionament dels processadors com a màquines algorísmiques d'interpretació de programes.

8. Conèixer l'arquitectura bàsica dels processadors.

9. Tenir habilitat per a analitzar les diferents opcions de materialització dels processadors.

10. Haver adquirit nocions bàsiques de l'arquitectura dels computadors.

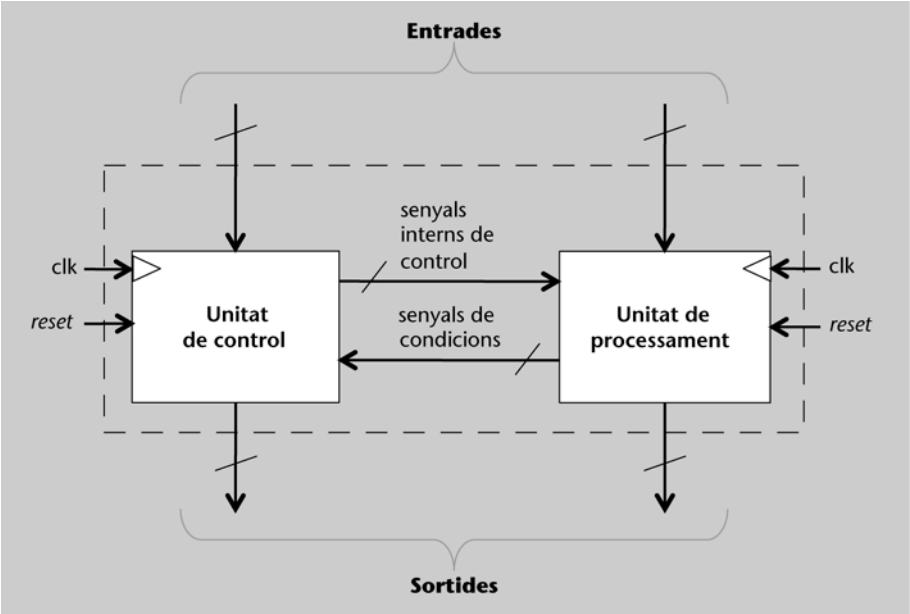
1. Màquines d'estats

Els circuits seqüencials són, de fet, màquines d'estats el comportament de les quals es pot representar amb un graf de transició d'estats. Si a cada estat s'associa l'execució d'una determinada operació, llavors les màquines d'estats són una manera d'ordenar l'execució d'un conjunt d'operacions. Aquest ordre queda determinat per una seqüència d'entrades i l'estat inicial de la màquina corresponent.

Així doncs, els circuits seqüencials es poden organitzar en dues parts: la que s'ocupa de les transicions d'estats i que implementen les funcions d'excitació de la màquina d'estats corresponent, i la que s'ocupa de fer les operacions amb les dades tant per a determinar condicions de transició com per a calcular resultats de sortida. Com ja s'ha dit en la introducció, la primera es denomina **unitat de control** i la segona, **unitat de processament**.

La figura 1 il·lustra aquesta organització. La unitat de control és una màquina d'estats que rep informació de l'exterior per mitjà de senyals d'entrada i també de la unitat de processament, en forma d'indicadors de condicions. Cada estat de la màquina corresponent té associades unes sortides que poden ser directament a l'exterior o cap a la unitat de processament, perquè faci uns càlculs determinats. Aquests càlculs es fan amb dades de l'exterior i també internes, ja que la unitat de processament també disposa d'elements de memòria. Els resultats d'aquests càlculs es poden observar des de l'exterior i, per això, aquesta unitat també té senyals de sortida.

Figura 1. Organització d'un circuit seqüencial



En el mòdul anterior s'ha explicat com funcionen els circuits seqüencials que materialitzen una màquina d'estats amb poc o gens de processament. En aquest apartat es tractarà de com es poden relacionar els estats amb les operacions, de manera que la representació de la màquina d'estats també inclogui tota la informació necessària per a la implementació de la unitat de processament.

1.1. Màquines d'estats finits com a controladors

Les **màquines d'estats finits** (o FSM, de l'anglès *finite state machines*) són un model de representació del comportament de circuits (i de programes) molt adequat per als controladors.

Un **controlador** és una entitat amb capacitat d'actuar sobre una altra per a portar-la a un estat determinat.

El regulador de potència d'un calefactor és un controlador de la intensitat del radiador. Cada posició del regulador (és habitual que siguin les següents: 0 per a apagat, 1 per a mitja potència i 2 per a potència màxima) seria un estat del controlador i també un objectiu per a la part controlada. És a dir, es pot interpretar que la posició 1 del regulador significa que cal portar el radiador a l'estat de consum d'una intensitat mitjana de corrent. El mateix exemple illustra la diferència amb una màquina d'estats no finits: si el regulador consistís en una roda que es pot girar fins a situar-se en qualsevol posició entre 0 i 2, llavors caldria un nombre (teòricament) infinit d'estats.

Un altre exemple de controlador és el del mecanisme d'ompliment d'aigua del dipòsit d'una tassa de vàter. En aquest cas, el sistema que s'ha de controlar té dos estats: el dipòsit pot estar ple (PLE) o pot no estar-ho (NO_PLE).

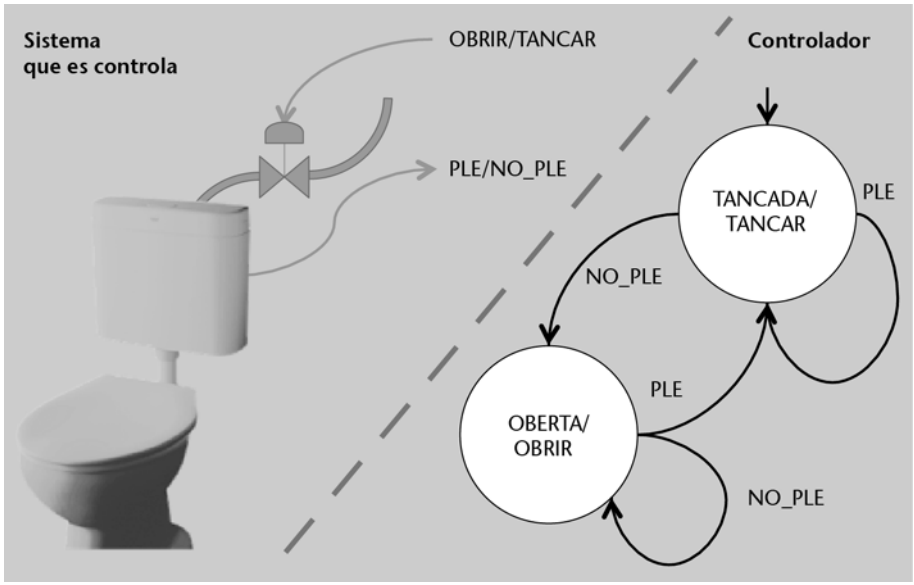
El mecanisme de control d'ompliment d'aigua s'ocupa de portar el sistema a l'estat de PLE de manera autònoma. En altres paraules, el controlador té una referència interna de l'estat al qual ha de portar el sistema que controla. Per a fer-ho, n'ha de detectar l'estat i, amb aquesta dada d'entrada, determinar quina acció ha de fer: obrir l'aixeta d'ompliment (OBRIR) o tancar-la (TANCAR). A més, cal tenir present que el controlador ha de mantenir l'aixeta oberta mentre el dipòsit no estigui ple i tancada sempre que estigui ple. Això fa que hagi de "recordar" en quin estat es troba: amb l'aixeta oberta (OBERTA) o tancada (TANCADA).

El funcionament dels controladors es pot representar amb un graf d'estats i també amb taules de transicions i de sortides. En aquest sentit, cal tenir present que els **estats captats** dels sistemes que controlen constitueixen les entra-

des de la màquina d'estats corresponent, i que les **accions** són les sortides vinculades als estats dels mateixos controladors.

Seguint amb l'exemple, els estats captats són PLE/NO_PLE, les accions són TANCAR i OBRIR i l'objectiu per al sistema controlat, que estigui PLE. L'estat inicial del controlador d'ompliment de la cisterna ha de ser TANCADA, que està associat a l'acció de TANCAR. D'aquesta manera, en començar el funcionament del controlador queda garantit que no hi haurà sobreeiximent del dipòsit. Si ja està PLE, s'ha de quedar en el mateix estat. Però, si es detecta que està NO_PLE, llavors s'ha d'omplir. Cosa que s'aconsegueix passant a l'estat d'OBERTA, en què s'obre l'aixeta. L'aixeta es manté oberta fins que es detecti que el dipòsit està PLE. En aquest moment, s'ha de TANCAR i es passa a l'estat de TANCADA.

Figura 2. Esquema del sistema amb el graf d'estats del controlador



La taula de transicions d'estats és la següent:

Estat del controlador	Estat del dipòsit	Estat següent del controlador
Estat actual	Entrada	Estat futur
TANCADA	NO_PLE	OBERTA
TANCADA	PLE	TANCADA
OBERTA	NO_PLE	OBERTA
OBERTA	PLE	TANCADA

I la taula de sortides és la següent:

Estat del controlador	Acció
Estat actual	Sortida
OBERTA	OBRIR
TANCADA	TANCAR

La **materialització** del controlador d'ompliment del dipòsit d'aigua d'una tassa de vàter es pot resoldre molt eficaçment sense cap circuit seqüencial: la majoria de cisternes porten una vàlvula d'admissió d'aigua controlada per un mecanisme amb una boia que sura. Amb tot, aquí es farà amb un circuit digital que implementi la màquina d'estats corresponent, un sensor de nivell i una electrovàlvula. Per a això, cal que es codifiqui tota la informació en binari.

En aquest cas, l'observació de l'estat del sistema (la cisterna del vàter) es fa copsant les dades del sensor, que són binàries: PLE (1) o NO_PLE (0). Les actuacions del controlador es fan activant l'electrovàlvula, que ha de ser una que normalment estigui tancada (0) i que es mantingui oberta mentre estigui activada (1). L'estat del controlador també s'ha de codificar en binari, per exemple, fent que coincideixi amb la codificació binària de l'acció de sortida. Així doncs, per a fer el circuit seqüencial del controlador del dipòsit, s'ha de fer una codificació binària de totes les dades, tal com es mostra en la taula següent:

Estat del dipòsit		**Estat del controlador**		**Acció**	
Identificació	**Codi**	**Controlador**	**Codi**	**Identificació**	**Codi**
NO_PLE	0	TANCADA	0	TANCAR	0
PLE	1	OBERTA	1	OBRIR	1

Com s'ha vist en l'exemple anterior, l'estat percebut del sistema que es controla és constituït per les diferents dades d'entrada del controlador. Per a evitar confusions en les denominacions dels dos tipus d'estats, es farà referència a l'estat percebut com a **entrades**.

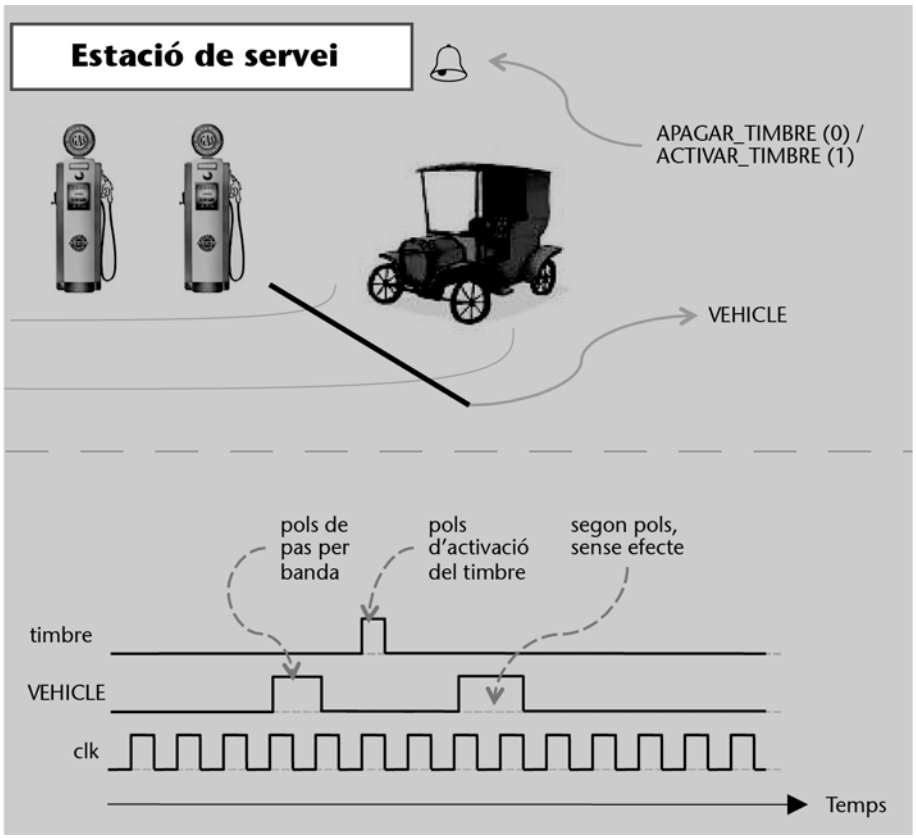
1.1.1. Procediment de materialització de controladors amb circuits seqüencials

Per a dissenyar un circuit seqüencial que implementi una màquina d'estats corresponent a un controlador s'han de seguir els passos que es descriuran a continuació. Per a il·lustrar-los, s'exemplificaran amb el cas d'un controlador que fa sonar un timbre d'avís d'entrada de vehicles en una estació de servei (figura 3).

1) D'acord amb la funcionalitat que s'ha d'implementar, decidir quines accions s'han de fer sobre el sistema que es controla.

En aquest cas, només hi ha dues accions: fer sonar el timbre (ACTIVAR_TIMBRE) o deixar-lo apagat (APAGAR_TIMBRE). Tal com es mostra en la part inferior de la figura 3, per a activar el timbre, n'hi ha prou d'enviar un pols a 1 a l'aparell corresponent. Per simplicitat, se suposa que només ha de durar un cicle de rellotge.

Figura 3. Sistema d'avís d'entrada de vehicles a una estació de servei



2) Determinar quina informació s'ha d'obtenir del sistema que es controla o, dit d'una altra manera, determinar les entrades del controlador.

Per al cas d'exemple, el controlador necessita saber si algun vehicle passa pel vial d'entrada. Per a això s'hi pot tenir un cable sensible a la pressió fixat a terra. Si s'activa, vol dir que un vehicle el trepitja i que s'ha de fer sonar el timbre d'avís. Per tant, hi ha d'haver una única entrada, VEHICLE, que serveixi perquè el controlador sàpiga si algun vehicle està accedint a l'estació. En altres paraules, el controlador pot "veure" l'estació en dos estats (o, si es vol, "fotos") diferents, segons el valor de VEHICLE.

Es considera que tots els vehicles tenen dos eixos i que només es fa sonar el timbre quan passa el primer (si hi passés un vehicle de més de dos eixos, com un camió, el timbre sonaria més d'un cop; amb tot, aquests casos no es tenen en compte per a l'exemple).

3) Dissenyar la màquina d'estats del controlador.

Es tracta de construir el graf d'estats que concordi amb el comportament que s'espera del conjunt: que soni un timbre d'alerta quan un vehicle entri en l'estació de servei.

Per a fer-ho, es comença amb un estat inicial de repòs en què es desconeixen les entrades i es decideix cap a quins estats ha de passar segons totes les possi-

bles combinacions d'entrada. I, per a cadascun d'aquests estats de destinació, es fa el mateix tenint en compte els estats que ja s'han creat.

L'estat inicial (INACTIU) tindria associada l'acció d'APAGAR_TIMBRE per a garantir que, estigués com estigués anteriorment, en engegar el controlador el timbre no sonés. La màquina d'estats hi ha de restar fins que no es detecti el pas d'un vehicle, és a dir, fins que VEHICLE = 1. En aquest cas, passarà a un nou estat, POLS_1, per a determinar quan ha acabat de passar el primer eix del vehicle i, per tant, quan s'ha de fer sonar el timbre.

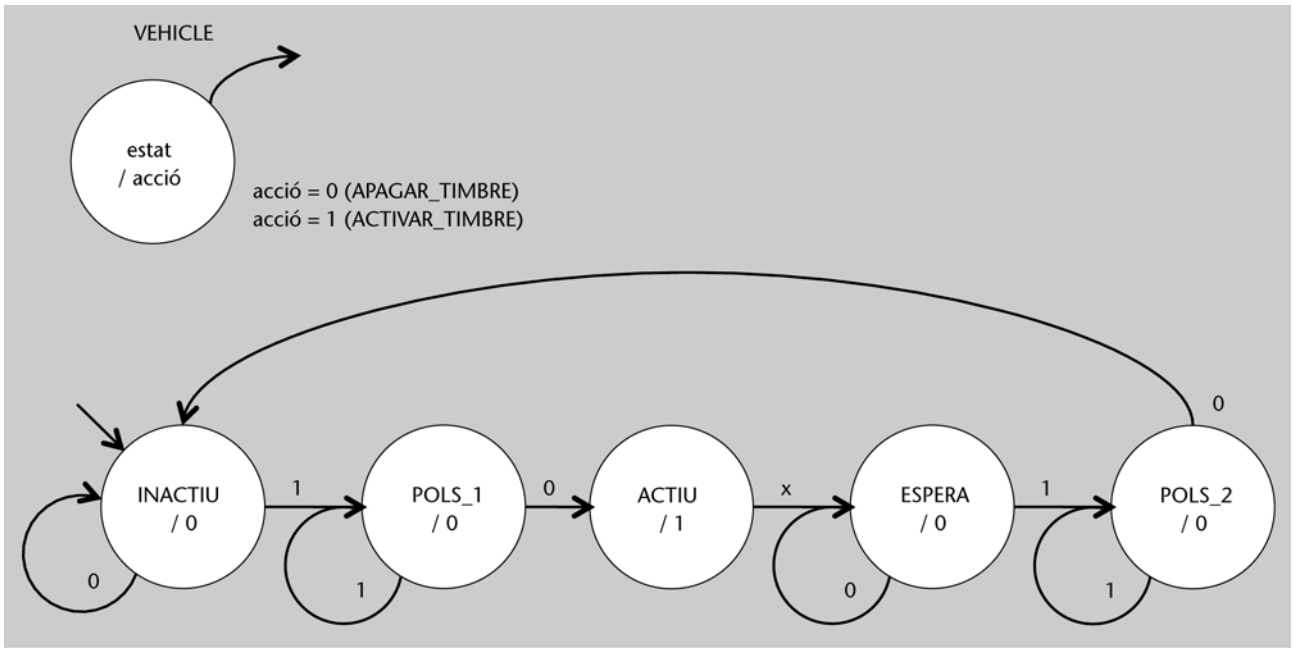
S'ha de tenir present que la velocitat de mostreig del senyal VEHICLE, és a dir, el nombre de lectures d'aquest bit per unitat de temps, és la del rellotge del circuit: molt més gran que la velocitat de pas de qualsevol vehicle. Per tant, la detecció d'un pols es fa en completar una seqüència del tipus 01...10, amb un nombre indefinit d'uns.

Així doncs, la màquina passa a l'estat POLS_1 després d'haver detectat un flanc de pujada (pas de 0 a 1) a VEHICLE, i no n'ha de sortir fins que hi hagi un flanc de baixada que indiqui la finalització del primer pols. En aquest moment ha de passar a un estat en què s'activi el timbre, ACTIU.

L'estat ACTIU és l'únic associat a l'acció d'ACTIVAR_TIMBRE. Com que això només cal fer-ho durant un cicle de rellotge, amb independència del valor que hi hagi a VEHICLE, la màquina d'estats passarà a un estat diferent. Aquest nou estat esperarà el pols corresponent al pas del segon eix del vehicle. De fet, en caldran dos: l'estat d'espera de flanc de pujada (ESPERA) i, després, l'estat d'espera de flanc de baixada (POLS_2) i, per tant, de finalització del pols.

Havent acabat del segon pols, s'ha de passar de nou a l'estat inicial, INACTIU. D'aquesta manera, es queda a l'espera que vingui un altre vehicle.

Figura 4. Graf d'estats del sistema d'avís d'entrada de vehicles



Amb això ja es pot construir el diagrama d'estats corresponent. En aquest cas, com que només hi ha un senyal d'entrada, és fàcil comprovar que s'han tingut en compte tots els casos possibles (totes les "fotos" del sistema que es controla) en cada estat.

Amb vista a la construcció d'un controlador robust, cal comprovar que no hi hagi seqüències d'entrada que provoquin transicions no desitjades.

Per exemple, segons el diagrama d'estats, el pas d'ACTIU a ESPERA es fa sense que importi el valor de VEHICLE, cosa que pot fer perdre un 1 d'un pols del tipus ...010... Com que, en l'estat següent, VEHICLE torna a ser 0, el pols no es llegeix i la màquina d'estats començarà a fer sonar el timbre, en els vehicles següents, en passar el segon eix i no pas el primer. Aquest cas, però, és difícil que passi quan la velocitat de mostreig és molt més elevada que la de canvi de valors a VEHICLE. Amb tot, també es pot alterar el diagrama d'estats perquè no sigui possible (es pot fer com a exercici voluntari).

4) Codificar en binari les entrades, els estats del controlador i les sortides.

Tot i que, en el procés de disseny de la màquina d'estats, les entrades i la sortida ja es poden expressar directament en binari, encara queda la tasca de codificació dels estats del controlador. Generalment s'opta per dues polítiques:

- tants bits com estats diferents i només un d'actiu per a cada estat (en anglès: *one-hot bit*) o

- numerar-los amb nombres binaris naturals, del 0 al nombre d'estats que hi hagi. Habitualment, l'estat codificat amb 0 és l'estat inicial, ja que facilita la implementació de la inicialització (*reset*) del circuit corresponent.

Del graf d'estats de la figura 4 es pot deduir la taula de transicions següent:

Estat actual				**Entrada**	**Estat següent**			
Identificació	q_2	q_1	q_0	**VEHICLE**	**Identificació**	q_2^+	q_1^+	q_0^+
INACTIU	0	0	0	0	INACTIU	0	0	0
INACTIU	0	0	0	1	POLS_1	0	0	1
POLS_1	0	0	1	0	ACTIU	0	1	0
POLS_1	0	0	1	1	POLS_1	0	0	1
ACTIU	0	1	0	x	ESPERA	0	1	1
ESPERA	0	1	1	0	ESPERA	0	1	1
ESPERA	0	1	1	1	POLS_2	1	0	0
POLS_2	1	0	0	0	INACTIU	0	0	0
POLS_2	1	0	0	1	POLS_2	1	0	0

En aquest cas, la codificació dels estats segueix la numeració binària.

La taula de sortides és la següent:

Estat actual				Sortida	
Identificació	q_2	q_1	q_0	Símbol	Timbre
INACTIU	0	0	0	APAGAR_TIMBRE	0
POLS_1	0	0	1	APAGAR_TIMBRE	0
ACTIU	0	1	0	ACTIVAR_TIMBRE	1
ESPERA	0	1	1	APAGAR_TIMBRE	0
POLS_2	1	0	0	APAGAR_TIMBRE	0

Com a apunt final, cal observar que si el codi de l'estat ESPERA fos 101 i no 011, la sortida seria, directament, q_1.

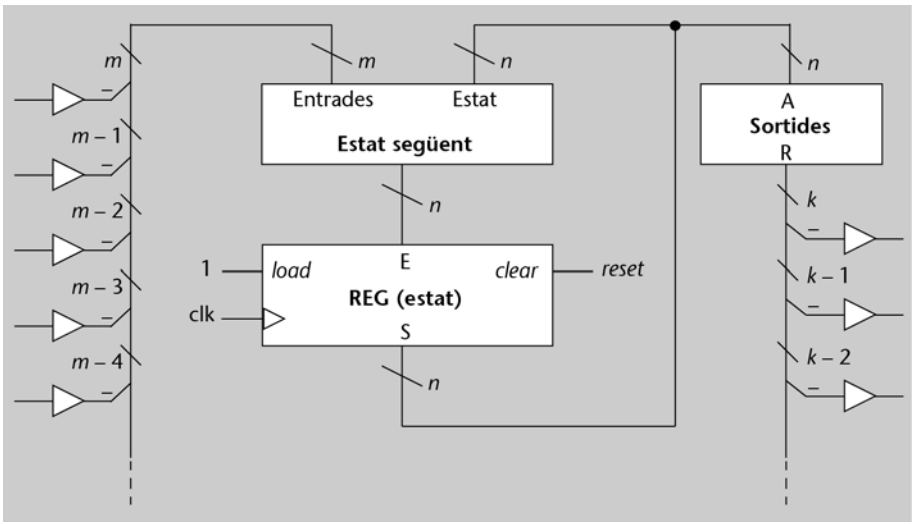
5) Dissenyar els circuits corresponents.

El model de construcció d'una FSM com a controlador que pren senyals binaris d'entrada i en dóna de sortida es basa en dos elements: un registre per a emmagatzemar l'estat, i una part combinacional que calcula l'estat següent i les sortides, tal com es veu en la figura 5.

Elements adaptadors

En els circuits sovint es fan servir uns elements adaptadors (o *buffers*) que serveixen per a transmetre senyals sense pèrdua. El símbol d'aquests elements és com el de l'inversor sense la boleta. En els esquemes dels circuits, també es fan servir per a indicar el sentit de propagació del senyal i, en el cas d'aquest material, per a indicar si es tracta de senyals d'entrada o de sortida.

Figura 5. Arquitectura d'un controlador basat en FSM

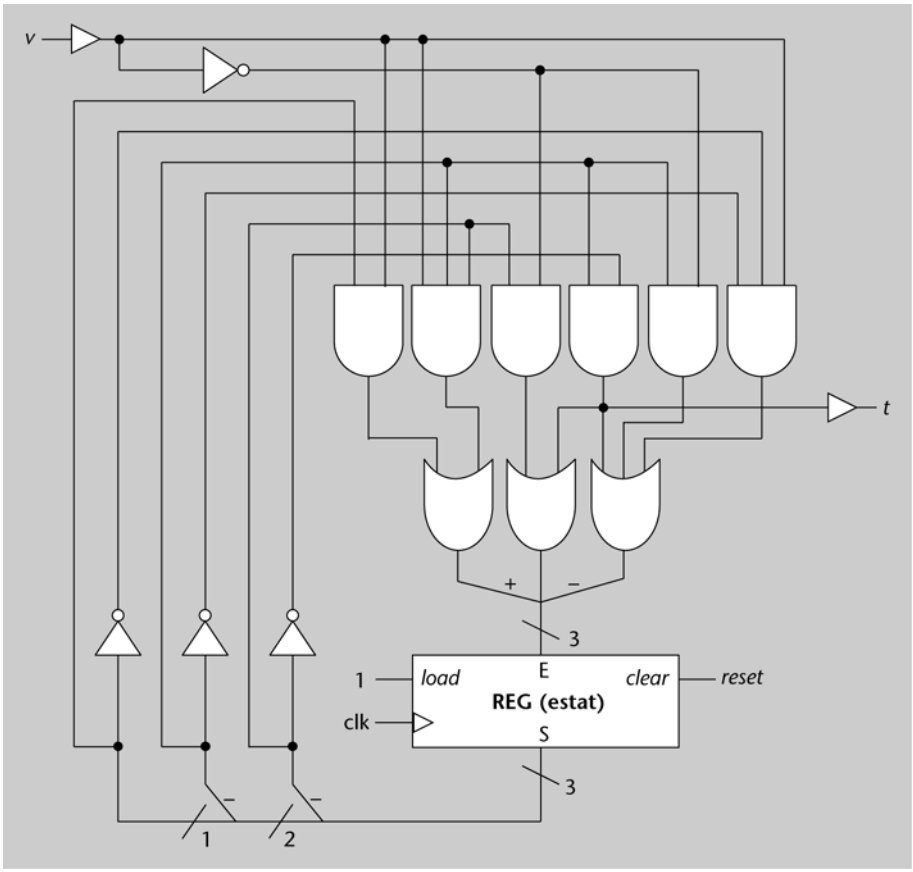


Tot i que la part combinacional se separa en dues parts –la que calcula l'estat següent i la que calcula els senyals de sortida–, hi pot haver elements comuns compartits.

Per al controlador del timbre d'avís d'entrada de vehicles a l'estació de servei, el circuit del controlador binari és el que es mostra en la figura 6. En el circuit, el senyal d'entrada v és el que indica la detecció de pas d'eix de vehicles (VEHICLE), i t (timbre), el de sortida que permet engegar el timbre. Les expressions de les funcions lògiques de càlcul de l'estat següent comparteixen un terme (q_1q_0') que, a més, es pot aprofitar com a funció de sortida (t). Les entra-

des dels sensors són al costat esquerre, i les sortides cap a l'actuador (el que fa l'acció, que és el timbre), al dret.

Figura 6. Esquema de l'FSM controladora del timbre d'avís d'entrada



En aquest cas, s'ha optat per fer-lo amb portes lògiques (es deixa, com a exercici, la comprovació que el circuit de la figura 6 correspon a les funcions lògiques de les taules de transició i de sortida del controlador).

En casos més complexos, és convenient fer servir blocs combinacionals més grans i també memòries ROM.

Activitats

1. En les comunicacions en sèrie entre dispositius, l'emissor pot enviar un pols a 1 perquè el receptor ajusti la velocitat de mostreig de l'entrada. Dissenyeu un circuit detector de velocitat que funcioni amb un rellotge a la freqüència més ràpida de comunicació i que doni, com a sortida, el valor de divisió de la freqüència a què rep els polsos a 1. Els valors de sortida poden ser els següents:

- 00 (no hi ha detecció),
- 01 (cap divisió, el pols a 1 dura el mateix que un pols de rellotge),
- 10 (el pols dura dos cicles de rellotge) i
- 11 (el pols d'entrada és a 1 durant tres cicles de rellotge).

Si un pols d'entrada supera els tres cicles de rellotge, la sortida també serà 11.

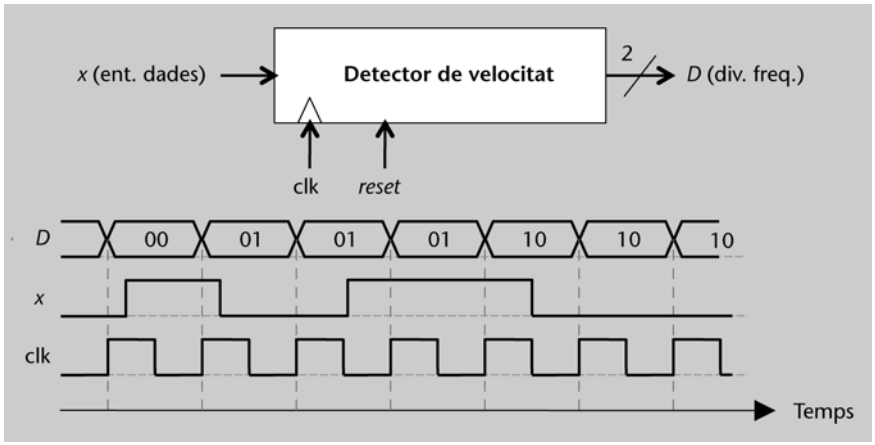
A tall d'exemple, en la figura 7 trobareu una il·lustració del funcionament del detector de velocitat.

Fixeu-vos que, en el segon pols a 1 de *x*, la sortida passa de 00 a 10 progressivament. És a dir, compta el nombre de cicles en què es detecta l'entrada a 1. A més, resta en el mateix

estat cada cop que acaba un compte, ja que, just després d'un cicle amb l'entrada a 0, el circuit manté la sortida al darrer compte que ha fet.

Feu el diagrama d'estats i la taula de transicions corresponent.

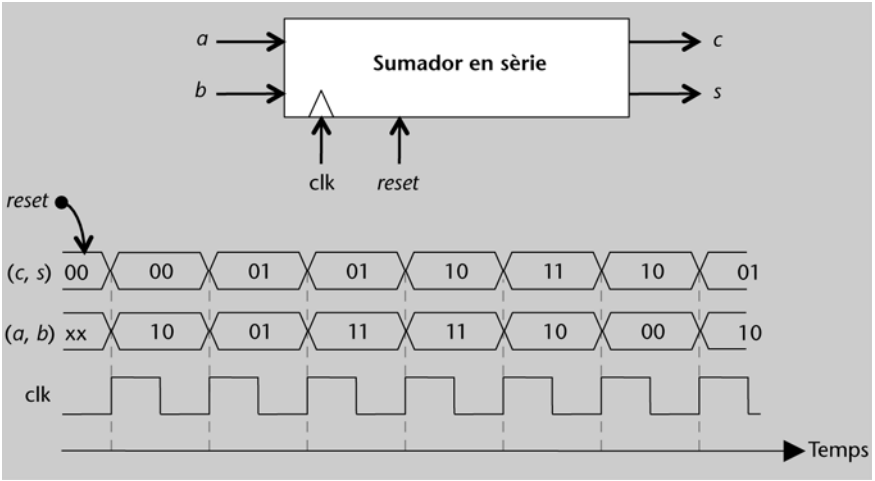
Figura 7. Esquema del detector de velocitat de transmissió



2. Dissenyeu un sumador en sèrie. És un circuit seqüencial amb dues entrades, *a* i *b*, i dues sortides, *c* i *s*, que són, respectivament, el ròssec (*carry*) i el resultat (suma) de l'operació de suma dels bits *a* i *b* juntament amb el possible ròssec anterior. Inicialment, la sortida ha de ser (0, 0). És a dir, el ròssec inicial és 0 i el bit menys significatiu del nombre que es forma amb la sèrie de sortida és 0.

En la figura 8 hi ha una seqüència de valors d'entrada i les sortides corresponents. Fixeu-vos que el ròssec de sortida també es té en compte en el càlcul de la suma següent.

Figura 8. Esquema d'un sumador en sèrie



Cal que primer feu el diagrama d'estats i, llavors, la taula de veritat de les funcions de transició corresponents. Compareu el resultat de les funcions de transició amb les d'un sumador complet.

1.2. Màquines d'estats finits esteses

Les FSM dels controladors tracten amb entrades (estats dels sistemes que controlen) i sortides (accions) que, de fet, són bits. Ara bé, sovint, les dades d'entrada i les de sortida són valors numèrics que, evidentment, també es codifiquen en binari, però que tenen una amplada d'uns quants bits.

En aquests casos, hi ha l'opció de representar cada condició d'entrada o cada possible càlcul de sortida amb un únic bit. Per exemple, que el nivell d'un dipò-

sit no superi un llindar es pot representar per un senyal d'un únic bit que es pot anomenar *per_sota_de_llindar*, o que el grau d'obertura d'una vàlvula s'hagi d'incrementar es pot veure com un senyal d'1 bit amb un nom com *obre_més*.

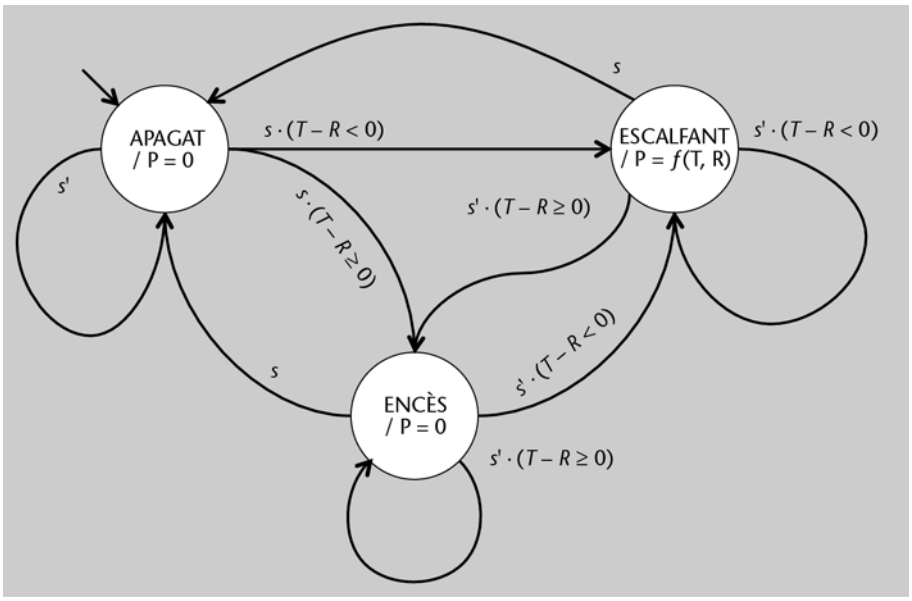
Amb tot, és més convenient incloure, en els grafs d'estats, no solament els bits d'entrada i els de sortida sinó també les avaluacions de condicions sobre valors d'entrada i els càlculs per a obtenir els valors de sortida d'una manera directa, sense haver de fer cap traducció inicial a valors d'un o pocs bits. És a dir, el model de màquines d'estats finits "s'estén" per a incloure-hi operacions amb dades. Per això es parla d'**FSM esteses** o **EFSM** (*extended* FSM).

Per a il·lustrar-ho, es pot pensar en un calefactor amb un termòstat que el controli: en funció de la diferència entre la temperatura desitjada (la referència) i la que es mesura (l'estat del sistema que es controla), es determina la potència del calefactor (l'estat següent del controlador). En aquest cas, el controlador rep dades d'entrada que són valors numèrics i determina a quina potència posa el radiador, un nombre enter entre 0 i 2, per exemple.

El funcionament del calefactor és senzill. Quan està apagat, la potència de calefacció ha de ser 0. Quan s'encén, pot estar en dos estats diferents: escalfant, si la diferència de temperatura entre la mesurada i la de referència és negativa, o actiu (encès, però no escalfant) si la temperatura que es detecta és igual o superior a la que es vol.

El graf d'estats següent és el corresponent a un model EFSM del sistema que s'ha comentat. En el graf, *s* és un senyal d'entrada que fa de commutador (*switch*, en anglès) i que cada cop que s'activa (es posa a 1) apaga o encén el termòstat segons si està encès o no, respectivament; *T* és un nombre binari que representa la temperatura que es mesura a cada moment; *R* el valor de la referència (la temperatura desitjada), i *P* un nombre binari entre 0 i 2 que es calcula en funció de *T* i de *R*, $f(T, R)$.

Figura 9. Graf d'estats d'un termòstat



En l'EFSM de l'exemple es combinen senyals de control d'un únic bit amb senyals de dades de més d'un bit. De les últimes, n'hi ha dues d'entrada, T i R, i una de sortida, P.

Cal tenir present que els càlculs de les condicions com $s \cdot (T - R < 0)$ o $s \cdot (T - R \geq 0)$ sempre han de donar un valor lògic (en tos dos casos, hi ha una resta que es compara amb 0, i dóna com a resultat un valor lògic amb el qual es pot operar, fent el producte lògic, amb s). Els càlculs per als valors de sortida com $f(T, R)$ no tenen aquesta restricció, ja que poden ser de més d'1 bit de sortida, com és el cas de P, que en fa servir dos per a representar els valors 0, 1 i 2.

El disseny del circuit seqüencial corresponent es proposarà en l'activitat número 3, un cop s'hagi vist un exemple de materialització d'EFSM.

Les EFSM permeten, doncs, tenir en compte comportaments més complexos, en associar directament càlculs de condicions d'entrada i dels resultats de les sortides als arcs i nodes dels grafs corresponents. Com es veurà, el model de construcció dels circuits que materialitzen les EFSM és molt similar al de les FSM, però inclouen la part de processament de les dades binàries numèriques.

El procés de disseny d'un d'aquests circuits s'exemplificarà amb un **comptador**, que és un element freqüent en molts circuits seqüencials, ja que és la base dels temporitzadors, rellotges, velocímetres i d'altres components útils per als controladors.

En aquest cas, el comptador que es dissenyarà és similar als que ja s'han vist en el mòdul anterior, però que compta fins a un nombre donat, $M \geq 1$. És a dir, és un comptador de 0 a $M - 1$ "programable", que no començarà a comptar fins que no arribi un senyal d'inici (*begin*) i s'aturarà automàticament en arribar al valor $M - 1$. M es llegirà amb el senyal d'inici. També hi ha un senyal de sortida d'1 bit que indica quan s'ha arribat al final (*end*) del compte. En arrencar el funcionament, el senyal *end* ha de ser 1 perquè el comptador no està comptant, cosa equivalent a haver acabat un possible compte anterior.

Per comparació amb un comptador autònom de 0 a $M - 1$, com els que s'han vist en el mòdul anterior i que es poden construir a partir d'una FSM sense entrades, el que ara es proposa és fer-ne un que ho faci depenent d'un senyal extern i, per tant, que no sigui autònom.

Els comptadors autònoms de 0 a $M - 1$ tenen M estats i passen de l'un a l'altre segons la funció següent:

$$C^+ = (C + 1) \text{ MOD } M$$

És a dir, l'estat següent (C^+) és l'increment del nombre que representa l'estat (C) en mòdul M.

Aquesta operació es pot fer mitjançant el producte lògic entre la condició que l'estat tingui el codi binari d'un nombre inferior a (M – 1) i cadascun dels bits del nombre incrementat en 1:

$$C^+ = (C + 1) \text{ AND } (C < (M - 1))$$

A diferència del comptador autònom, el programable, a més dels estats associats a cadascun dels M passos, ha de tenir estats d'espera i de càrrega de valor límit, M, que és extern. Fer un graf d'estats d'un comptador així no és gens pràctic: un comptador en què M i C fossin de 8 bits tindria, com a mínim, $2^8 = 256$ estats.

Les EFSM permeten representar el comportament d'un comptador de manera més compacta perquè els càlculs amb C es poden deixar com a operacions amb dades emmagatzemades amb una variable associada, a part del càlcul de l'estat següent.

Una **variable** és un element del model d'EFSM que emmagatzema una dada que pot canviar (o variar, d'això el seu nom) de forma similar a com canvia d'estat. De la mateixa manera, la materialització del circuit corresponent necessita un registre per a cada variable, a més del registre per a l'estat.

Les variables són anàlogues als estats: poden canviar de valor en cada transició de la màquina. Així doncs, tal com hi ha funcions per al càlcul dels estats següents, hi ha funcions per al càlcul dels valors següents de les variables. Per aquest motiu es fa servir la mateixa notació (un signe més en posició de superíndex) per a indicar el valor següent de l'estat s (s^+) i per a indicar el valor següent d'una variable v (v^+).

Des d'una altra perspectiva, es pot veure una variable com un element intern que, d'una banda, proporciona una dada d'entrada (en el comptador, seria C) i, de l'altra, rep una dada de sortida que serà l'entrada en el cicle següent (en el comptador, seria C^+). En altres paraules, el contingut actual d'una variable (en el comptador, C) forma part de les entrades de l'EFSM i, entre les sortides de la mateixa màquina, hi ha el senyal que transporta el contingut de la mateixa variable en l'estat següent (en el comptador, C^+).

Variable d'una EFSM

En general, qualsevol variable d'una EFSM es pot transformar en informació d'estat, tot generant una màquina d'estats amb un nombre d'estats que pot arribar a ser tan gran com el producte del nombre de valors diferents de la variable pel nombre d'estats que tenia l'EFSM original. Per exemple, la transformació de l'EFSM en una FSM implica transformar tots els valors de la variable C en estats de la màquina resultant, ja que "ha de recordar" quin és el valor del compte actual (en aquest cas, el producte és per 1, ja que l'EFSM original només té un estat).

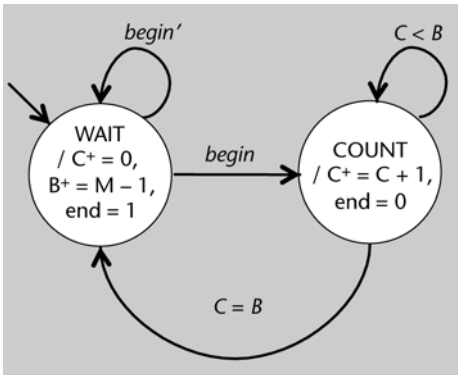
D'aquesta manera, es pot fer un model d'EFSM per a un comptador autònom que tingui un únic estat en què l'acció consisteixi a carregar el contingut de fer el càlcul de C^+ al registre corresponent.

Seguint amb els models d'EFSM per als comptadors programables, cal tenir en compte que han de tenir un estat addicional. Així doncs, l'EFSM d'un comptador programable té un estat més que la d'un que sigui autònom, és a dir, en

té dos. Un d'espera (WAIT) que s'activi el senyal d'entrada *begin* i el necessari per a fer el compte (COUNT) fins al màxim prefixat, *M*. En el primer, a més d'esperar que *begin* sigui 1, s'actualitza el valor de la variable *B* amb el valor de l'entrada *M*, menys 1. Així, en cas d'iniciar un compte, *B* contindrà el nombre més gran a què ha d'arribar. En altres paraules, *B* és la variable que ajuda a recordar el màxim del compte, amb independència del valor que hi hagi a l'entrada *M* en qualsevol moment posterior a l'inici del compte.

El comptador programable té dues sortides: la del mateix valor del compte, que s'emmagatzema en una variable *C*, i una altra d'1 bit, *end*, que indica quan està comptant o quan està en espera. El graf d'estats corresponent a aquest comportament es mostra en la figura 10.

Figura 10. Graf d'estats d'un comptador programable



Del comportament representat en el graf de la figura 10 es pot veure que el comptador espera a rebre un 1 per *begin* per a començar a comptar. Cal tenir present que, mentre compta (és a dir, en l'estat COUNT), el valor de *begin* no importa. En acabat el compte, sempre passa per l'estat WAIT i, per tant, entre un compte i el següent ha de transcórrer, com a mínim, un cicle de rellotge.

Com ja s'ha vist en l'exemple del calefactor amb termòstat, les entrades i les sortides no s'hi representen en binari sinó amb expressions simbòliques, és a dir, amb símbols que representen operadors i operands. Per a les transicions es fan servir expressions que, un cop avaluades, han de tenir un resultat lògic, com $C < B$ o *begin*'. Per als estats, s'utilitzen expressions que determinen els valors de les sortides que s'hi associen.

En aquest cas, hi ha dues possibilitats:

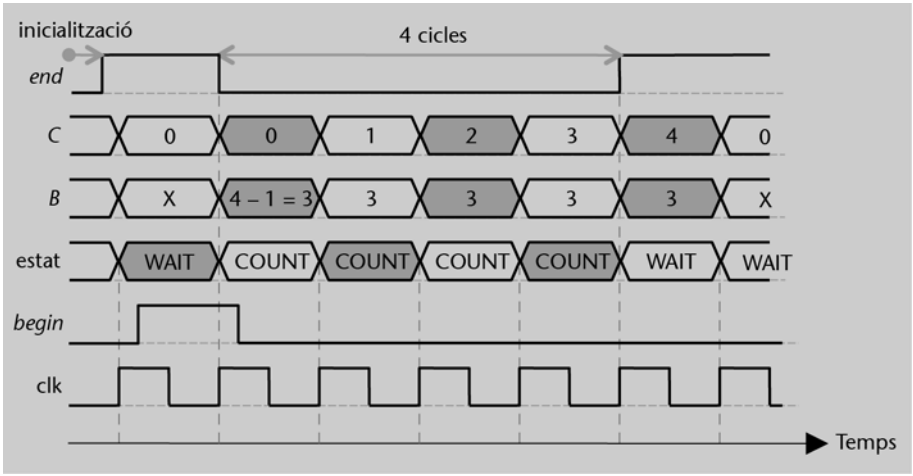
1) Per als senyals de sortida, s'expressa el valor que han de tenir en aquest estat. Per exemple, en arribar a l'estat COUNT, *end* serà 0.

2) Per a les variables, s'expressa el valor que adquiriran **en l'estat següent**. Per exemple, en l'estat COUNT, la variable *C* s'ha d'incrementar en una unitat per a l'estat següent. Perquè quedi clar, s'hi posa el superíndex amb el signe de més: $C^+ = C + 1$.

Perquè quedi més clar el funcionament d'aquest comptador programable, tot seguit es mostra un diagrama de temps per a un compte fins a 4. Es tracta d'un cas que comença amb una inicialització, motiu pel qual *end* és a 1 i *C* a 0. Just després de la inicialització, la màquina es troba en l'estat inicial de WAIT, la variable *B* ha adquirit un valor indeterminat *X*, *C* continua estant a 0, i el senyal de sortida *end* és 1 perquè no està comptant. Estant en aquest estat rep l'activació de *begin*, és a dir, la indicació de començar el compte i, juntament amb aquesta indicació, rep el valor de 4 per l'entrada *M* (senyal que no es mostra en el cronograma). Amb aquestes condicions, l'estat següent serà el de comptar (COUNT), i el valor següent de les variables *C* i *B* serà 0 i 4 – 1, respectivament. Per tant, en arribar a COUNT, $B = 3$ i $C = 0$.

És molt important tenir present que el contingut de les variables canvia en acabat de l'estat actual.

Figura 11. Cronograma d'exemple per al comptador



Cal fixar-se que el nombre de cicles que transcorren entre el cicle de rellotge en què s'ha posat *end* a 0 i el que marca el final (*end* = 1) és igual a *M*. També es pot observar que les variables prenen el valor de la sortida corresponent a l'inici de l'estat següent. Per aquest motiu, $C = 4$ en el primer WAIT després d'acabar el compte (això es podria evitar fent que C^+ es calculés amb mòdul *M*, de manera que, en lloc d'assignar el valor 4 s'hi assignés un 0).

A partir del grafs d'estats s'obtenen les taules de transicions d'estats i de sortides següents.

Estat actual	Entrada	Estat futur
WAIT	*begin'*	WAIT
WAIT	*begin*	COUNT
COUNT	$(C < B)$	COUNT
COUNT	$(C = B)$	WAIT

Estat actual	Sortida
WAIT	$C^+ = 0$, $B^+ = M - 1$, $end = 1$
COUNT	$C^+ = C + 1$, $B^+ = B$, $end = 0$

En aquest cas, en la taula de sortides s'han especificat els valors per a les variables en tots els casos, encara que no canviïn, com és el cas de la variable *B* a COUNT.

Amb vista a la materialització del circuit corresponent, cal codificar en binari tota la informació:

- La codificació dels senyals d'un únic bit, d'entrada o de sortida, és directa. En l'exemple, són *begin* i *end*.

- Els termes de les expressions lògiques de les transicions que contenen referències a valors numèrics es transformen en senyals d'entrada d'un únic bit que en representen el resultat lògic. Normalment, són termes que contenen operadors de relació. Per al comptador n'hi ha un que determina si $C < B$, i un altre per a $C = B$. Als senyals d'1 bit que s'obtenen s'hi fa referència amb els termes entre parèntesis, tal com apareixen en la taula anterior.

Expressions de resultat lògic

Les expressions de resultat lògic també es poden identificar amb senyals d'un únic bit que, al seu torn, s'han d'identificar amb noms significatius com, per exemple, *CltB*, de C *is less than* B, i *CeqB*, de C *is equal to* B.

- Les expressions que indiquen els càlculs per a obtenir els valors dels senyals numèrics de sortida poden ser úniques per a cada un o no. El més habitual, però, és que un determinat senyal de sortida pugui tenir diverses opcions. En el comptador, C pot ser 0 o $C + 1$, i B pot ser B o el valor de l'entrada M disminuït en una unitat. Quan hi ha diverses expressions que donen valors diferents per a una mateixa sortida s'introdueix un "selector", és a dir, un senyal de diversos bits que selecciona quin dels resultats s'ha d'assignar a una determinada sortida. Així doncs, caldria un selector per a C i un altre per a B, tots dos d'1 bit, ja que hi ha dos casos possibles per a cada nou valor de les variables B i C. La codificació dels selectors és la de la taula següent:

Operació	Selector	Efecte sobre la variable
$B^+ = B$	0	Manteniment del contingut
$B^+ = M - 1$	1	Canvi de contingut
$C^+ = 0$	0	Emmagatzematge del valor constant 0
$C^+ = C + 1$	1	Emmagatzematge del resultat de l'increment

Cal tenir en compte que els selectors es materialitzen, habitualment, com a entrades de control de multiplexors de busos connectats a les sortides corresponents. En aquest cas, són senyals de sortida lligats a variables i, per tant, als registres pertinents. Per aquest motiu, els efectes sobre les variables són equivalents al fet que els registres associats carreguin els valors dels senyals de sortida que se seleccionin.

En les taules següents es mostra la codificació de les entrades i de les sortides tal com s'ha establert. Quant a les entrades, s'ha de tenir en compte que les condicions $(C < B)$ i $(C = B)$ són oposades i, per tant, n'hi ha prou, per exemple, de fer servir $(C < B)$ i $(C < B)'$, que és equivalent a $(C = B)$. En la taula, es mostren els valors dels dos senyals. La codificació dels estats és una codificació directa, amb l'estat inicial WAIT amb codi 0. Els senyals de sortida són *sB* (selector de valor següent de la variable B), *sC* i *end*. El valor de C també és una sortida del comptador.

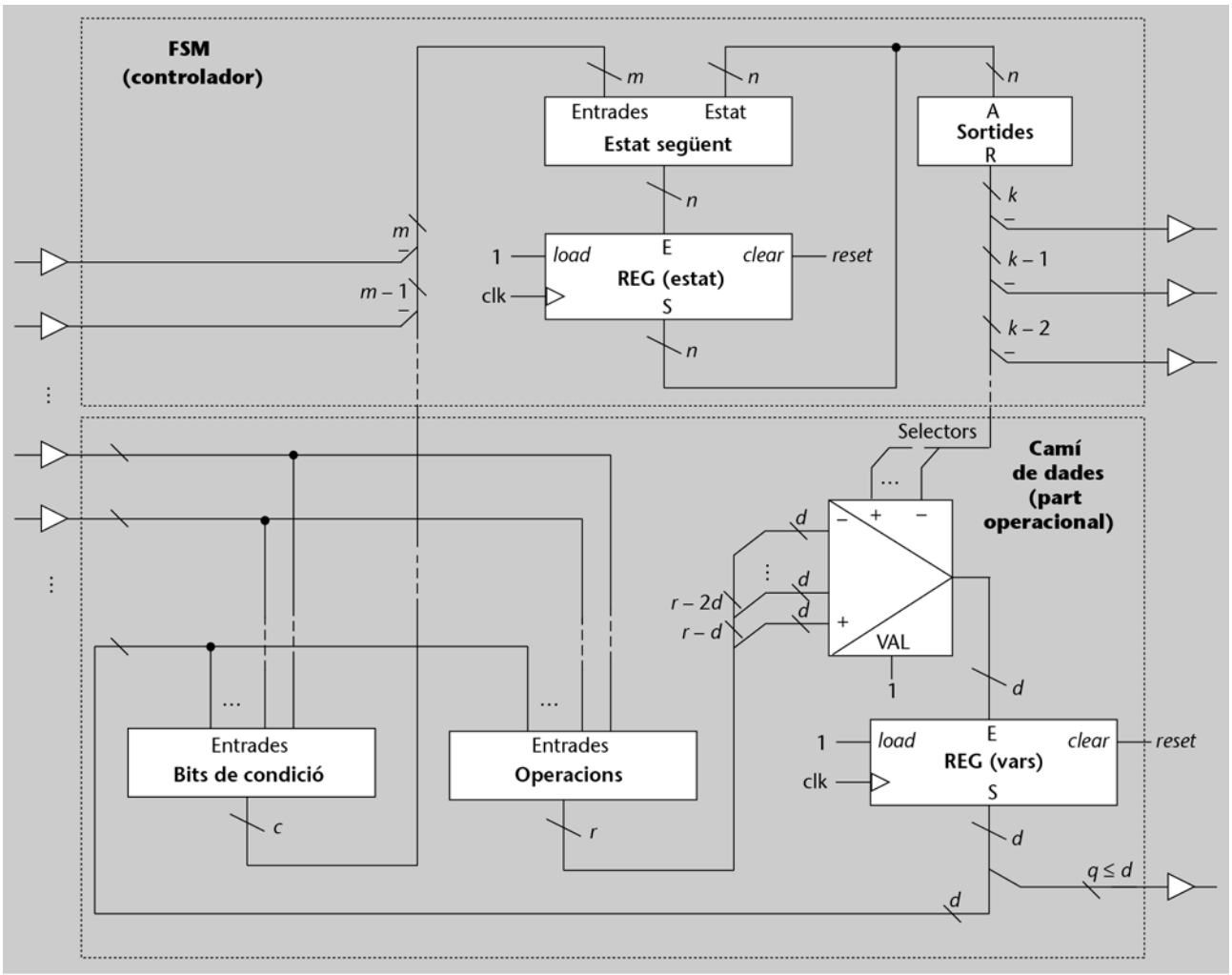
Estat actual		Entrades			Estat⁺
Símbol	*s*	*begin*	*C < B*	*C = B*	s^+
WAIT	0	0	x	x	0
WAIT	0	1	x	x	1
COUNT	1	x	0	1	0
COUNT	1	x	1	0	1

Estat	Sortides		
	sB	*sC*	*end*
0	1	0	1
1	0	1	0

Abans de passar a la implementació de les funcions de les taules de veritat anteriors cal veure com es construeix una d'aquestes EFSM. En general, l'arquitectura d'aquesta mena de circuits seqüencials separa la part que s'ocupa de fer les operacions amb les dades numèriques de la part que tracta amb els senyals d'1 bit i dels selectors.

En la figura 12 es pot veure l'organització per mòduls d'un circuit seqüencial per a una EFSM. S'hi distingeixen dues parts. La que apareix en el requadre superior és la de l'FSM, que rep, com a entrades, els senyals d'entrada d'un únic bit i també els resultats lògics de les operacions amb les dades (bits de condició) i que té, com a sortides, senyals d'un únic bit i també selectors per als re-

Figura 12. Arquitectura d'una EFSM



sultats a emmagatzemar als registres interns. La segona part és la que fa les operacions amb dades o **camí de dades**. Aquesta part rep, com a entrades, tots els senyals d'entrada de dades de l'EFSM i els selectors i, com a sortides, proporciona tant els bits de condició per a l'FSM de la part controladora com les dades numèriques que corresponen als registres lligats a les variables internes de l'EFSM. Addicionalment, les sortides de part d'aquests registres poden ser, també, sortides de l'EFSM.

En resum, doncs, l'EFSM té dos tipus d'entrades (els senyals de control d'1 bit i els senyals de dades, de múltiples bits) i dos tipus de sortides (senyals d'un únic bit, que són de control per a elements exteriors, i senyals de dades resultants d'alguna operació amb d'altres dades, internes o externes). El model de construcció que s'ha mostrat les organitza de manera que els bits de control es processen amb una FSM i les dades, amb una unitat operacional diferenciada. L'FSM rep, a més, senyals d'1 bit (condicions) de la part operacional i li proporciona bits de selecció d'operació per a les sortides. Aquesta part aprofita aquests bits per generar les sortides corresponents, entre les quals hi ha, també, el valor següent de les variables. Al mateix temps, part del contingut dels registres també es pot utilitzar com a sortides de l'EFSM.

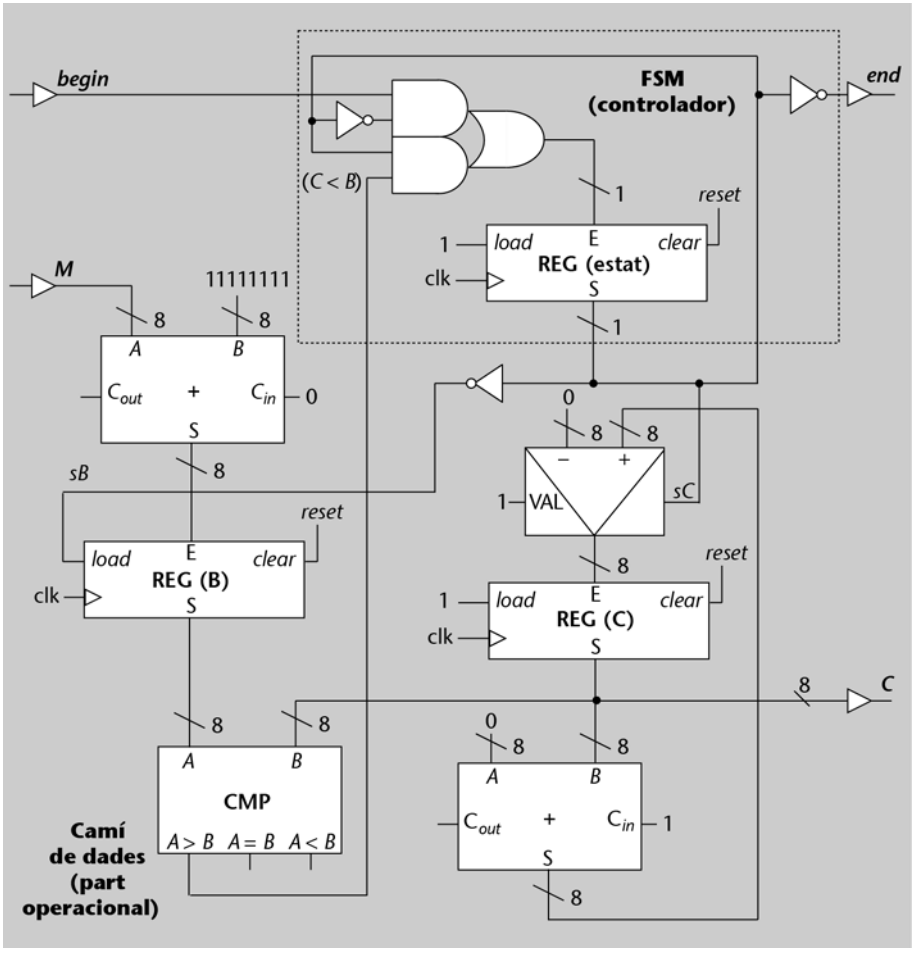
Aquesta arquitectura, que separa el circuit seqüencial en dues unitats –la de control i la de processament–, s'anomena **arquitectura de màquina d'estats amb camí de dades** o **FSMD**, que és l'acrònim de l'anglès *finite-state machine with datapath*.

Per a la construcció del comptador s'ha de seguir el mateix model. En la figura 13 es pot veure el circuit complet. La part superior es correspon amb la unitat de control, que calcula la funció d'excitació, que és $s^+ = s' \cdot begin + s \cdot (C < B)$, i les de sortida (*end*, cap a l'exterior, i *sB* i *sC*, que són selectors), que s'obtenen directament de l'estat o de l'estat complementat. S'hi ha mantingut el registre d'estat, com en la figura 12, però és un registre d'un únic bit que es pot implementar amb un biestable. S'hi ha fet una petita optimització: el bus d'entrada per al registre *B* no el proporciona la sortida d'un multiplexor controlat per *sB* sinó que és directament el valor $M - 1$, que es carrega o no, segons *sB*, que està connectat a l'entrada *load* del registre *B*. (En general, això sempre es podrà fer amb les variables que tinguin, d'una banda el canvi de valor i, de l'altra, el manteniment del contingut.)

En el circuit es pot observar que les variables de les EFSM proporcionen valors d'entrada i emmagatzemen valors de sortida: el registre *C* proporciona el valor del compte en el període de rellotge en curs i, en acabat, emmagatzema el valor de sortida que es calculi per a C^+, que depèn del selector *sC* i, en darrera instància, de l'estat actual. En conseqüència, el valor de *C* no s'actualitza fins que no es passa a l'estat següent.

Cal tenir en compte que, en les EFSM, l'estat complet queda format per l'estat de l'FSM de control i l'estat del camí de dades, que es defineix pel contingut dels registres de dades interns. Com que la funcionalitat de les EFSM queda fixada per la part controladora, és habitual fer l'equivalència entre els seus estats, deixant a part els dels camins de dades. Així, quan es parla d'estat d'una EFSM es fa referència a l'estat de la seva unitat de control.

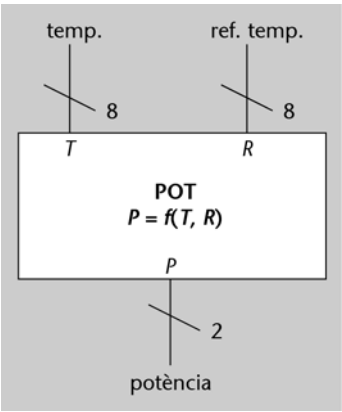
Figura 13. Circuit seqüencial corresponent al comptador de l'exemple



Activitats

3. Implementeu el circuit de l'EFSM del termòstat. Per a això, disposeu d'un mòdul, POT, que calcula $f(T, R)$.

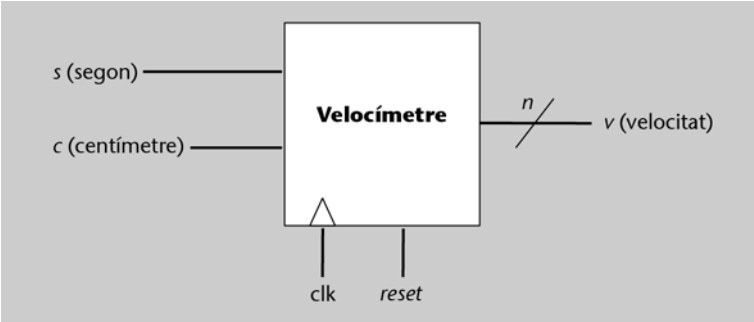
Figura 14. Esquema d'entrades i sortides del mòdul POT



4. En molts casos, del comptadors només interessa el senyal de final de compte. En aquest sentit, no cal que el compte es faci de 0 a $M - 1$, prenent la notació del que s'ha vist, sinó que també es pot fer al revés. D'aquesta manera n'hi ha prou amb un únic registre el valor del qual es vagi decrementant fins a 0. Modifiqueu l'EFSM del comptador de l'exemple perquè es comporti d'aquesta manera i dissenyeu el circuit resultant.

5. Un velocímetre consisteix en un mecanisme que compta unitats d'espai recorregut per unitats de temps. Per exemple, pot comptar centímetres per segon. Es tracta de dissenyar el graf d'estats d'un controlador d'un velocímetre que prengui, com a entrades, un senyal *s*, que es posa a 1 durant un cicle de rellotge a cada segon, i un senyal *c*, que és 1 en el període de rellotge en què s'ha recorregut un centímetre, i, com a sortida, el valor de la velocitat en centímetres per segon, *v*, que s'ha d'actualitzar a cada segon. Cal tenir present que *s* i *c* poden passar al mateix instant. El valor de la velocitat s'ha de mantenir durant tot un segon. En arrencar, durant el primer segon ha de ser 0.

Figura 15. Esquema d'entrades i sortides del mòdul del velocímetre



1.3. Màquines d'estats-programa

Les màquines d'estats esteses permeten integrar condicions i operacions en els arcs i nodes dels grafs corresponents. No obstant això, moltes operacions no són simples i impliquen l'execució d'un conjunt d'operacions elementals en seqüència, és a dir, són **programes**. Per exemple, una acció que depengui d'una mitjana calculada a partir de diverses entrades requereix una sèrie de sumes prèvies.

En aquest sentit, tot i que es pot mantenir aquesta seqüència d'operacions de manera explícita en el graf d'estats de la màquina corresponent, resulta molt més convenient que els programes quedin associats a un únic node. D'aquesta manera, cada node representa un estat amb un possible programa associat. Per aquest motiu, les màquines corresponents s'anomenen **màquines d'estats-programa** o **PSM** (de les sigles en anglès de *program-state machines*). Les PSM es poden veure com a representacions més compactes de les EFSM.

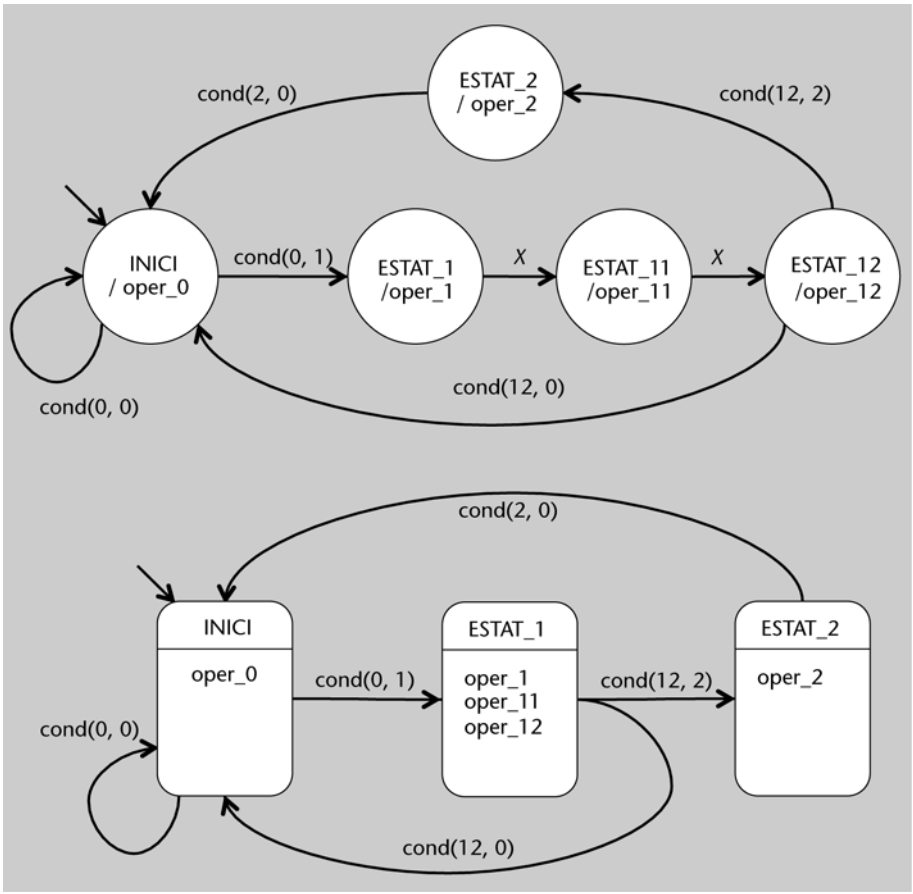
De fet, els programes de cada estat són un conjunt d'accions que cal executar en seqüència (en el model general de PSM, els programes no tenen cap restricció, però en aquest text se suposarà que són sèries d'accions que es duen a terme una rere l'altra, sense salts).

Els estats a què estan vinculats es poden mantenir durant l'execució del programa o canviar segons les entrades corresponents. De fet, en el primer cas es diu que les possibles transicions només es fan un cop acabada l'execució del programa (*transition on completion* o TOC, en anglès) i, en el segon cas, que es

fan de manera immediata en iniciar l'execució (*transition immediately* o TI, en anglès). En aquest text només es tractarà de les PSM amb TOC, ja que el model amb TI queda fora de l'abast d'aquesta assignatura.

En l'exemple de la figura 16, hi ha una EFSM amb una sèrie de nodes lligats a unes accions que es duen a terme en seqüència sempre que s'hi arriba per ESTAT_1. Cal tenir en compte que els arcs marcats amb *X* es corresponen amb transicions incondicionals, és a dir, en les quals no es té en compte cap bit de condició o entrada de control.

Figura 16. Exemple d'una EFSM (a dalt) i d'una PSM (a baix) equivalents



En el model de PSM amb TOC, aquestes seqüències d'estats es poden integrar en un únic estat-programa (ESTAT_1). Cada estat-programa té associada l'acció d'executar, en seqüència, les accions del programa corresponent. En el cas de l'ESTAT_1, es faria primer *oper*_1, després *oper*_11 i, finalment, *oper*_12. Cal tenir present que cadascuna d'aquestes accions pot implicar l'execució de diverses accions subordinades en paral·lel, a la manera en què es duen a terme en els nodes dels estats d'una EFSM.

Seguint amb l'exemple de la mitjana dels valors de diverses entrades, les accions en seqüència correspondrien a la suma d'un valor diferent en cada pas i, finalment, en la divisió pel nombre de valors que s'han sumat. Cadascuna d'aquestes accions es faria en paral·lel al càlcul de possibles sortides de l'EFSM. Per exemple, es pot suposar que la sortida *y* es manté en el darrer valor que

havia pres i que, en el darrer moment, es posa a 1 o a 0 segons si la mitjana supera o no un determinat llindar prefixat, T. Aquesta mena d'estat-programa, per a quatre valors d'entrada concrets (V_1, V_2, V_3 i V_4), podria tenir un programa associat com el que es presenta a continuació:

$$\{S^+ = V_1, y = M \geq T\};$$
$$\{S^+ = S + V_2, y = M \geq T\};$$
$$\{S^+ = S + V_3, y = M \geq T\};$$
$$\{S^+ = S + V_4, y = M \geq T\};$$
$$\{M^+ = S/4, y = S/4 \geq T\}$$

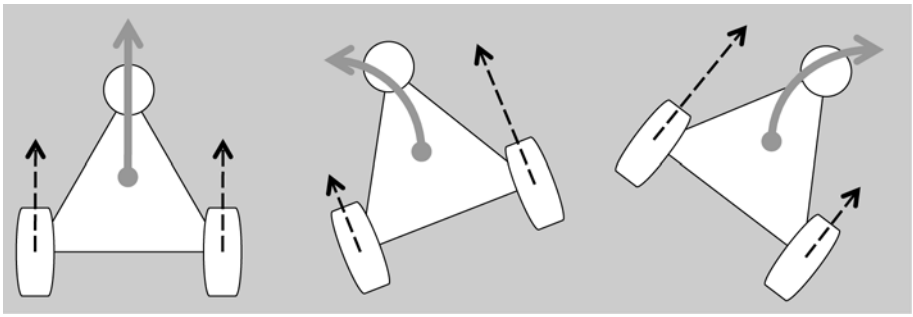
Aquest programa fa servir, a més, dues variables per a contenir la suma (S) i la mitjana (M) dels valors d'entrada. Cal tenir present que:

- les claus indiquen grups de càlculs que es fan en un únic pas, és a dir, inclouen totes les accions que es fan en un mateix període de temps en paral·lel;
- les comes separen les diferents accions o càlculs, i
- els punts i comes separen dos grups d'accions per a fer-se en seqüència o, el que seria el mateix, que en una EFSM es correspondrien a estats diferents.

Per a veure els avantatges de les PSM, es treballarà amb un exemple realista: el disseny d'un controlador de velocitat d'un servomotor per a un robot.

Habitualment, els robots senzills estan dotats amb un parell de servomotors que estan connectats a les rodes corresponents i es mouen de manera similar a la d'un tanc o qualsevol altre vehicle amb erugues: si es fan girar les dues rodes a la mateixa velocitat i en el mateix sentit, el vehicle es mou endavant o enrere; si les rodes giren a diferent velocitat o en diferent sentit, el vehicle descriu una corba segons el cas que toqui.

Figura 17. Moviment per parell motriu diferencial



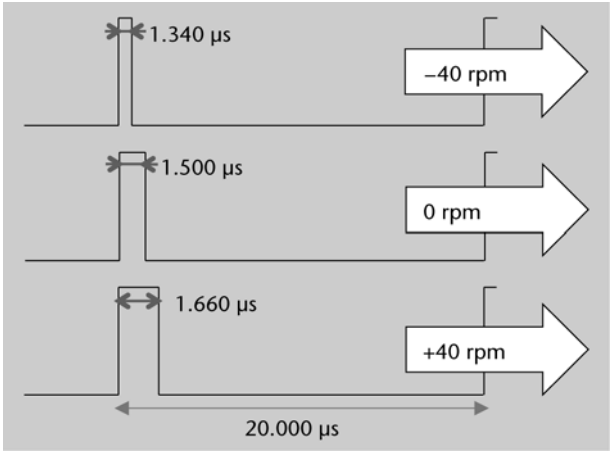
Com funcionen els servomotors?

Els **servomotors** són uns motors que tenen un senyal de control que, en funció de l'amplada dels polsos a 1 que tingui, fan girar el rotor fins a una certa posició. Convenientment adaptats, això té com a resultat que l'amplada dels polsos determina a quina velocitat, i en quin sentit, han de girar. Aquests polsos han de tenir una certa periodicitat

per a mantenir-los funcionant. De fet, el senyal de control descriu una forma d'ona que "transporta informació" en l'amplada d'aquests polsos periòdics i, per això mateix, es diu que és un senyal amb modulació d'amplitud de pols o PWM (de *pulse-width modulation*, en anglès). La modulació fa referència, justament, al canvi que experimenta l'amplada del pols (en aquest cas) segons la informació que s'hi transmet.

Per als servomotors, és habitual que les amplades dels polsos siguin entre 1.340 i 1.660 microsegons (μs), amb una periodicitat d'un pols cada 20.000 μs, tal com es mostra en la figura 18. En absència de pols, el motor resta immòbil, igual que amb polsos de 1.500 μs, que és el valor mitjà. Els polsos d'amplades superiors fan que el rotor es mogui en un sentit, i els d'amplades inferiors, en el sentit contrari.

Figura 18. Exemple de control d'un servomotor per amplada de pols



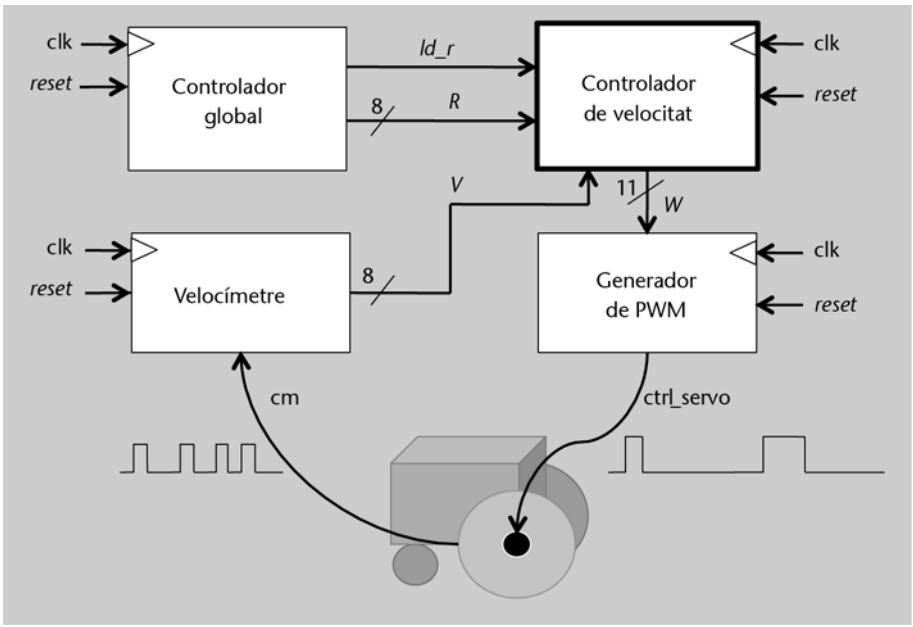
Malauradament, la relació entre l'amplada de pols i la velocitat en revolucions per minut (rpm) no és lineal (vegeu la figura 20) i pot canviar al llarg del temps.

El controlador de velocitat d'una de les rodes d'un vehicle que es mogui amb parell motriu diferencial és una part fonamental del control general del robot corresponent. En el cas de l'exemple, serà un dels mòduls del robot, que es relaciona amb altres mòduls del controlador complet tal com es mostra en la figura 19.

El mòdul del controlador de velocitat s'ocupa d'indicar quina és l'amplada de pols que s'ha de generar per al servomotor corresponent en funció tant del que li indiqui el controlador global com de la velocitat de gir que capti per mitjà d'un velocímetre.

Els senyals que relacionen els diversos mòduls es detallen tot seguit. L'explicació sempre és en referència al controlador de velocitat. El senyal d'entrada *ld_r* s'activa quan hi ha una nova velocitat de referència *R*. La velocitat de referència és un valor en el rang [–40, +40] i precisió de ½ rpm, per la qual cosa es representa en format de coma fixa de 7 bits per a la part entera i 1 bit per a la part fraccionària i en complement a 2. L'entrada *V*, que ve del velocímetre, indica la velocitat de gir de la roda, que es mesura amb precisió de ½ rpm, en el mateix format que *R*. Finalment, la sortida *W* indica l'amplada del pols per al generador del senyal modulat per amplada de pols. L'amplada de pols és un valor expressat en microsegons, que està en el rang [1.340, 1.660] i, per tant, necessita 11 bits, si s'expressa en format de nombre binari natural.

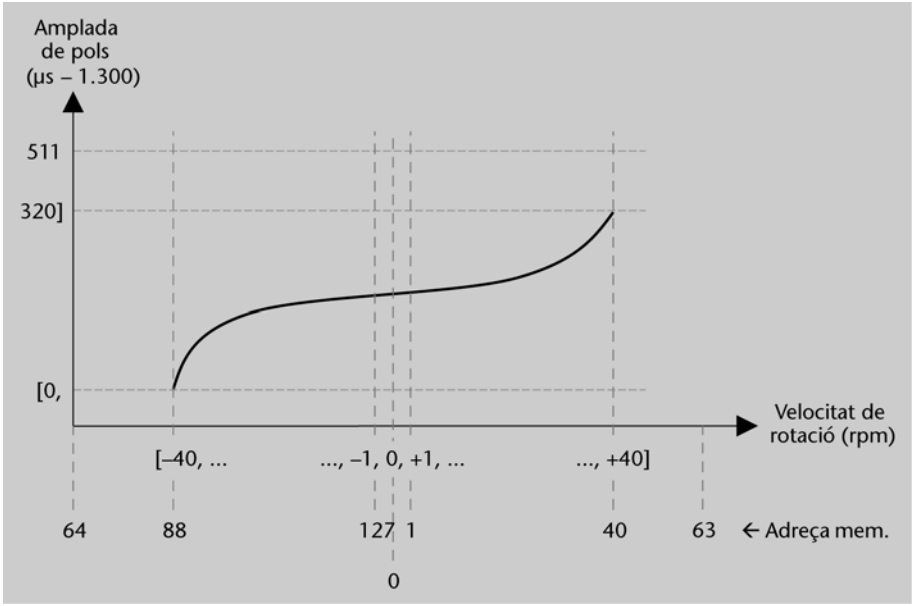
Figura 19. Esquema dels mòduls d'un controlador d'un robot



La relació no lineal entre la velocitat i l'amplada de pols s'emmagatzema en una memòria ROM del controlador de velocitat. En aquesta memòria s'han d'emmagatzemar els punts de la funció que relaciona l'amplada de pols amb la velocitat de rotació, tal com es mostra en la figura 20. Així, donada una velocitat presa com a adreça de la memòria, s'obté l'amplada de pols corresponent, presa com el contingut de la posició corresponent.

Per minimitzar les dimensions de la ROM i sabent que les amplades en microsegons són sempre en el rang [1.340, 1.660], cada posició de memòria només emmagatzemarà l'excés sobre 1.340 i, conseqüentment, serà un valor entre 0 i 320. Per tant, les paraules de memòria seran de 9 bits.

Figura 20. Representació de la funció no lineal de l'amplada de pols respecte de la velocitat de rotació



Per a reduir el nombre de posicions cal disminuir el nombre de punts que es guarden de la funció. Així, en lloc de guardar un valor d'amplada de pols per

a cada valor de velocitat diferent, se'n pot guardar un cada dues i interpolar les amplades per a les velocitats que quedin entremig. En aquest cas, això coincideix amb emmagatzemar els valors de les amplades de pols per a les velocitats enteres (és a dir, amb el bit fraccionari a 0) i, si és el cas, interpolar el valor de l'amplada de pols per als valors de velocitat amb el bit de ½ rpm a 1. D'aquesta manera, les velocitats que es prenen per a determinar les posicions de la memòria ROM són valors en el rang [–40, +40], que es poden codificar en binari amb 7 bits.

En resum, la memòria ROM serà de $2^7 \times 9 = 128 \times 9$ bits. Les adreces s'obtenen del valor de la velocitat de manera directa. És a dir, es tracten els 7 bits dels nombres com si fossin nombres naturals. Això fa que els valors positius es trobin en les adreces més petites i els negatius, després. De fet, si s'observa la figura 20 es pot comprovar que les velocitats positives es guarden de les posicions 0 a 40, i les negatives, de la posició 88 (que és igual a 128 – 40) a la posició 127 (que és igual a 128 – 1). Les posicions de memòria entre la 41 i la 87 no es fan servir.

Amb la informació de les entrades i de com s'obté la sortida ja es pot passar a dissenyar el model de comportament del controlador de velocitat, en aquest cas, amb una PSM.

Se suposa que la màquina s'engega a l'arribada de *ld_r* i només retorna a l'estat inicial amb un *reset*.

A l'estat inicial, INICI, espera que s'activi *ld_r*. Quan s'activa *ld_r* passa a l'estat de NOVA_R, en què comença l'execució d'un programa. Com que es tracta d'un model de PSM de tipus TOC, és a dir, que només fa transicions un cop que s'ha completat el programa associat, no es canvia d'estat fins que no n'acabi l'execució. El primer que fa aquest programa és carregar la nova referència *R* a dues variables, *U* i *T*. La primera serveix per a emmagatzemar *R*, i la segona per a contenir la velocitat de treball que, inicialment, ha de ser *R*. Després converteix aquest valor en un valor d'amplada de pols tot aprofitant la funció de transformació que hi ha emmagatzemada a la memòria ROM.

La conversió de velocitat a amplada de pols es fa amb una interpolació de dos punts de la funció. El valor de la velocitat es representa en nombres de 8 bits, un dels quals és fraccionari, i en Ca2. Però la memòria només conté el valor de la funció de transformació per a punts enters de 7 bits. Així doncs, si el bit fraccionari és 1, es considera que el valor de la funció serà la mitjana dels valors anterior i posterior:

$$A = (\mathrm{M}[T_7T_6T_5T_4T_3T_2T_1] + \mathrm{M}[T_7T_6T_5T_4T_3T_2T_1 + 1])/2$$

en què *A* és un valor entre 0 i 320 interpolat entre les posicions de memòria en les quals es troba el punt *T*. Els 7 bits més significatius, $T_7T_6T_5T_4T_3T_2T_1$,

constitueixen l'adreça de memòria en què hi ha el valor de la funció (la divisió per dos consisteix a desplaçar el resultat 1 bit a la dreta).

Si el bit fraccionari (T_0) és 0, el valor resultant és directament el del primer accés a memòria. Per simplicitat, en el programa sempre es fa la interpolació però, en lloc de sumar 1 a la segona adreça, s'hi suma T_0. D'aquesta manera, si és 1 es fa una interpolació i, si no, se suma dos cops el mateix valor i es divideix per 2, cosa que el deixa igual. Com que els valors de la memòria estan esbiaixats respecte de 1.340, se'ls ha de sumar aquesta quantitat per a obtenir l'amplada de pols en microsegons.

La darrera operació del programa de l'estat NOVA_R és l'activació d'un temporitzador per a esperar que l'acció de control tingui efecte. Per a fer-ho, n'hi ha prou de posar a 1 el senyal de *begin* corresponent i esperar un 1 per la sortida *end* del temporitzador, que ha d'estar preparat perquè el temps que transcorri entre l'activació i la finalització del període sigui suficient perquè el motor actuï.

Temporitzador

Un **temporitzador** és un comptador com els que s'han vist anteriorment, amb una entrada *M* per indicar-li quants cicles de rellotge ha d'estar comptant, una entrada (*begin*) per iniciar el compte i una sortida (*end*) que es posa a 1 quan ha arribat al final del compte. Si se sap el període del rellotge, *M* determina el temps que ha de transcórrer fins que no n'activi el senyal de finalització.

Aquest temporitzador és un submòdul (és a dir, un mòdul inclòs en un de més gran) que forma part del controlador de velocitat i, per tant, els senyals *begin* i *end* són interns i no visibles a l'exterior. La resta de la discussió se centrarà en el submòdul principal, que és el que es modelarà amb una PSM.

L'espera s'efectua en l'estat MESURANT. D'acord amb el diagrama de mòduls de la figura 19, en aquesta espera el velocímetre tindrà temps d'actualitzar els càlculs de la velocitat. En rebre l'activació del senyal *end*, que indica que el temporitzador ha arribat al final del compte, es passa a l'estat MOVENT.

De fet, l'estat MOVENT és el cor del controlador: calcula l'error entre la velocitat de treball, *T*, i la velocitat mesurada, *V*, i el suma a l'adreça de la de referència per a corregir el desviament entre la funció emmagatzemada i la realitat observada. Si no hi ha cap error, el valor amb què actualitzarà la variable de sortida *W* serà el mateix que ja tenia. Cal tenir en compte que l'estat de càrrega de les bateries i les condicions en què es troben les rodes associades tenen molta influència sobre els servomotors i que aquesta correcció és molt convenient.

El programa associat és igual al de NOVA_R tret que la velocitat que es converteix en amplada de pols de sortida es corregeix amb l'error mesurat. És a dir, la velocitat a què s'ha de moure el motor, *T*, és la de referència, *U*, més l'error calculat, que és la diferència entre la velocitat desitjada anterior, *T*, i la que s'ha mesurat en el moment actual, *V*:

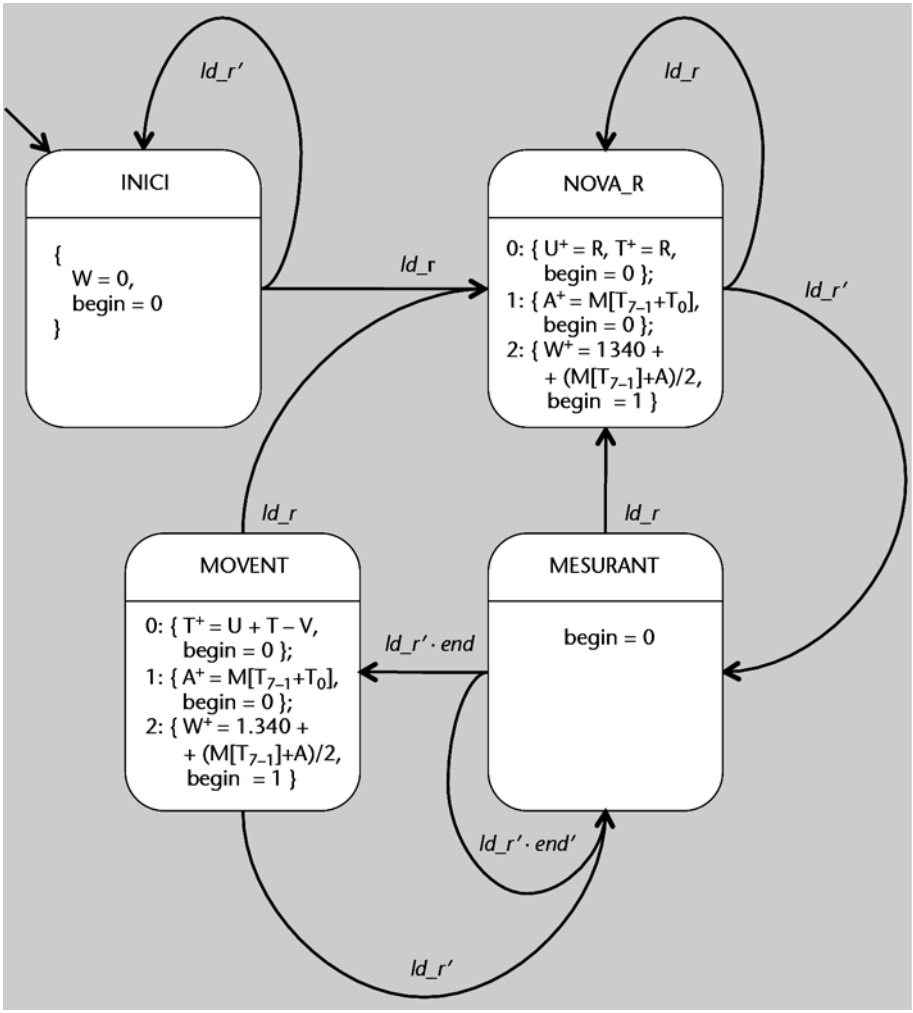
$$T^+ = U + (T - V)$$

Per exemple, si la velocitat de referència és de $U = 15$ cm/s, la velocitat de treball serà, inicialment, $T = 15$ cm/s. Si arriba un moment en què la velocitat mesurada és de $V = 12$ cm/s, significa que l'amplada de pols que es té per als 15 cm/s no és suficient i que cal augmentar-la una mica. En aquest cas s'augmentarà fins a la de la velocitat de $T^+ = 18$ cm/s. Si l'amplada resultés massa gran i, en la mesura següent, se'n mesurés una de 16 cm/s, el càlcul anterior faria que la propera velocitat de treball fos $T^+ = 15 + (18 - 16) = 17$ cm/s.

En aquest cas, els programes dels estats MOVENT i NOVA_R treballen amb les mateixes variables. Com que els programes executen les seves instruccions en seqüència, les variables actualitzen el seu valor en acabat d'una acció i el tenen disponible per a la següent. Així, el càlcul d'A^+ es fa amb el valor calculat per a T en la instrucció prèvia, és a dir, amb el valor de T^+.

En resum, un cop carregada una nova referència en l'estat-programa NOVA_R, la màquina anirà alternant entre els estats MOVENT i MESURANT mentre no se li indiqui de carregar una nova referència o se li faci una inicialització. El graf del PSM corresponent es pot veure en la figura 21. En el diagrama, les accions que es fan en un mateix període de rellotge s'han agrupat amb claus, i els diversos passos de cada programa s'han enumerat convenientment.

Figura 21. Diagrama d'estats del control adaptatiu de velocitat



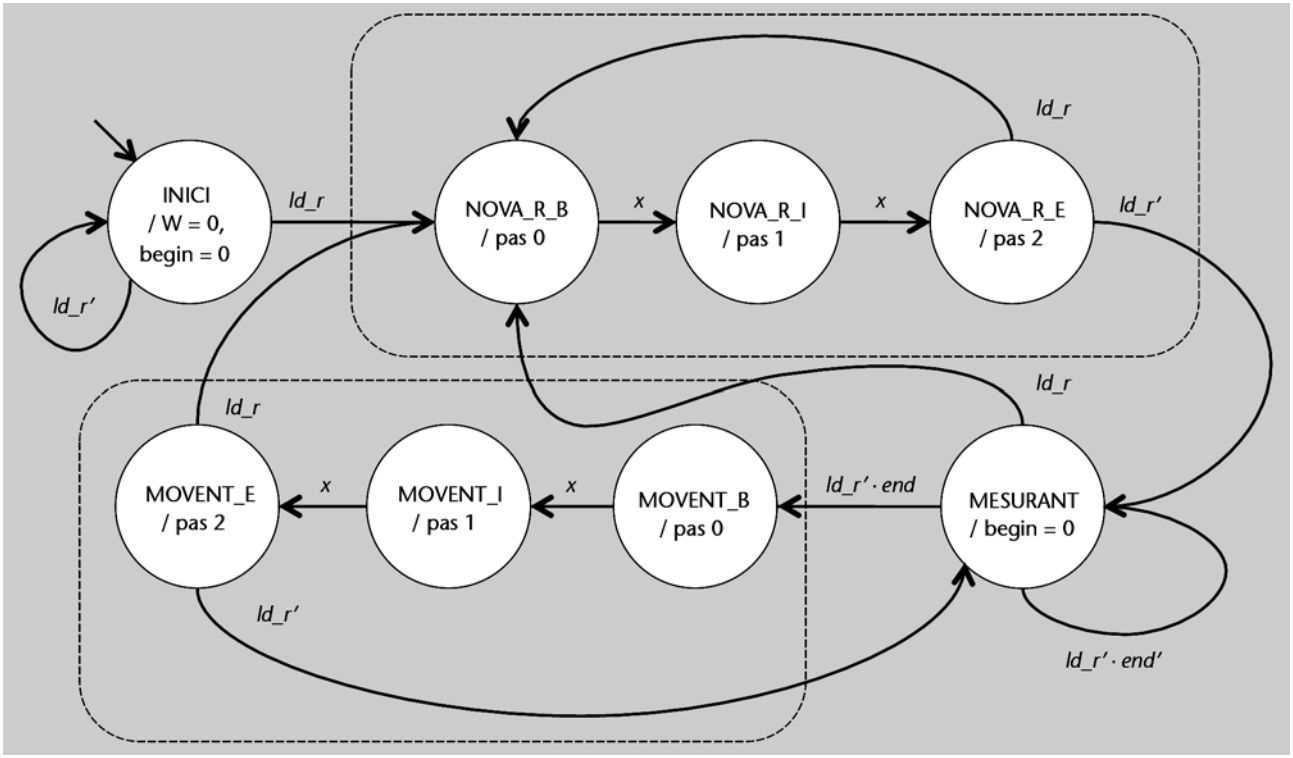
Les PSM es poden construir de dues maneres:

1) per expansió dels estats-programa en seqüències d'estats, de manera contrària a la que s'ha mostrat en la figura 16, i

2) per creació d'EFSM per a cada estat-programa, formant un conjunt d'EFSM interrelacionats.

En el primer cas, el nombre d'estats a tenir en compte per a la materialització pot ser molt gran, tal com es pot comprovar en la figura 22, en què es passa de quatre a vuit estats. En el graf corresponent, els estats-programa del cas d'exemple NOVA_R i MOVENT s'han dividit en tants subestats com passos tenen els programes. En aquest cas particular, se'ls han donat noms acabats en *_B*, de *principi* o *begin*; *_I*, d'*estat intermedi*, i *_E*, de *fi* o *end*. Per simplicitat, les accions associades a cada pas dels programes s'han substituït per una referència al número de pas que correspon.

Figura 22. EFSM de la PSM del controlador de velocitat



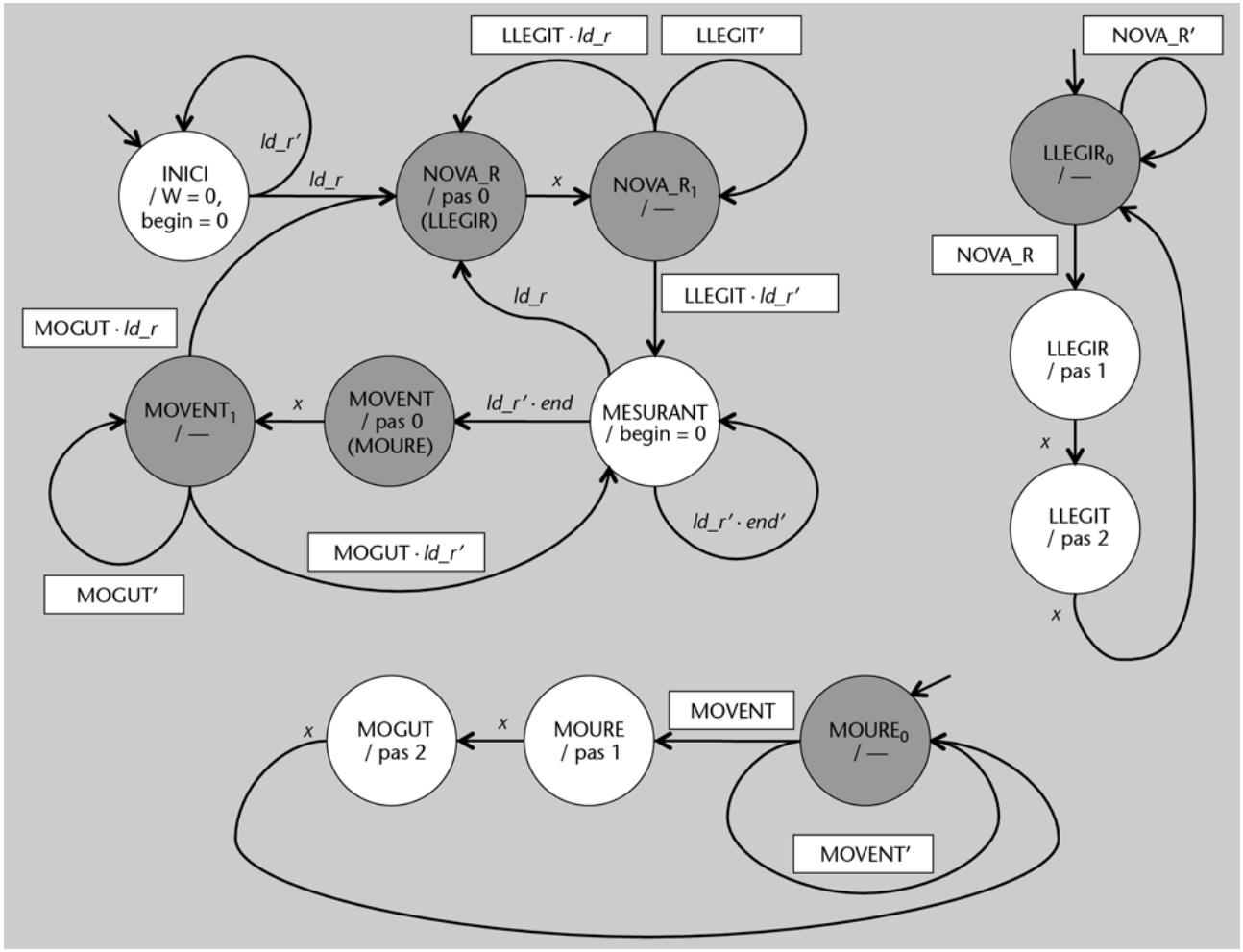
En el segon cas, el disseny de les EFSM és molt més simple a canvi, com es veurà, d'haver d'afegir estats addicionals i senyals per a coordinar-les.

En l'exemple de la figura 21, l'EFSM global és la formada pels estats INICI, NOVA_R, MESURANT i MOVENT, i n'hi ha dues per als programes dels estats NOVA_R i MOVENT. Tal com es pot veure en la figura 23, aquestes dues darreres EFSM estan formades pels estats $LLEGIR_0$, LLEGIR i LLEGIT, i per $MOURE_0$, MOURE i MOGUT, respectivament. També s'hi pot observar que els estats NOVA_R i MOVENT s'han desdoblat en dos, afegint els estats $NOVA_R_1$ i $MOVENT_1$ a l'EFSM global.

Els estats addicionals serveixen per a efectuar esperes. Així doncs, quan l'EFSM global arriba a l'estat NOVA_R o MOVENT els EFSM corresponents han de passar d'un estat d'espera ($LLEGIR_0$ o $MOURE_0$) als que porten a la seqüència d'estats dels programes que tenen associats. Al mateix temps, l'EFSM global ha de passar a un estat d'espera ($NOVA_R_1$ o $MOVENT_1$) del qual ha de sortir en el moment en què l'EFSM del programa que s'ha activat retorni a l'estat d'espera propi, cosa que indica que se n'ha finalitzat l'execució.

Per a veure aquest comportament d'una manera una mica més clara, es pot suposar que l'EFSM principal, el global, és a l'estat d'INICI i *ld_r* es posa 1, amb la qual cosa, l'estat següent serà NOVA_R. En aquest estat-programa cal dur a terme les accions dels passos 0, 1 i 2 del programa corresponent abans de comprovar si s'ha de carregar una nova referència o s'ha de passar a controlar el moviment. Per això, a NOVA_R s'executa el primer pas del programa i s'activa l'EFSM secundària corresponent, a la dreta de la figura 23. En aquesta EFSM, l'estat inicial $LLEGIR_0$ només s'abandona quan l'EFSM principal és a l'estat NOVA_R. Llavors, es passa d'un estat al següent per a dur a terme tots els passos del programa fins a acabar, moment en què retorna a $LLEGIR_0$. En tot aquest temps, l'EFSM principal es troba a l'estat $NOVA_R_1$, del qual surt en el moment en què l'EFSM secundària ha arribat al darrer estat de la seqüència, LLEGIT. Cal notar que els arcs de sortida de $NOVA_R_1$ són els mateixos que els de l'estat-programa NOVA_R original, però amb les condicions de sortida multiplicades pel fet que l'EFSM secundària sigui a l'estat LLEGIT.

Figura 23. Vista de la PSM del controlador de velocitat com a conjunt d'EFSM



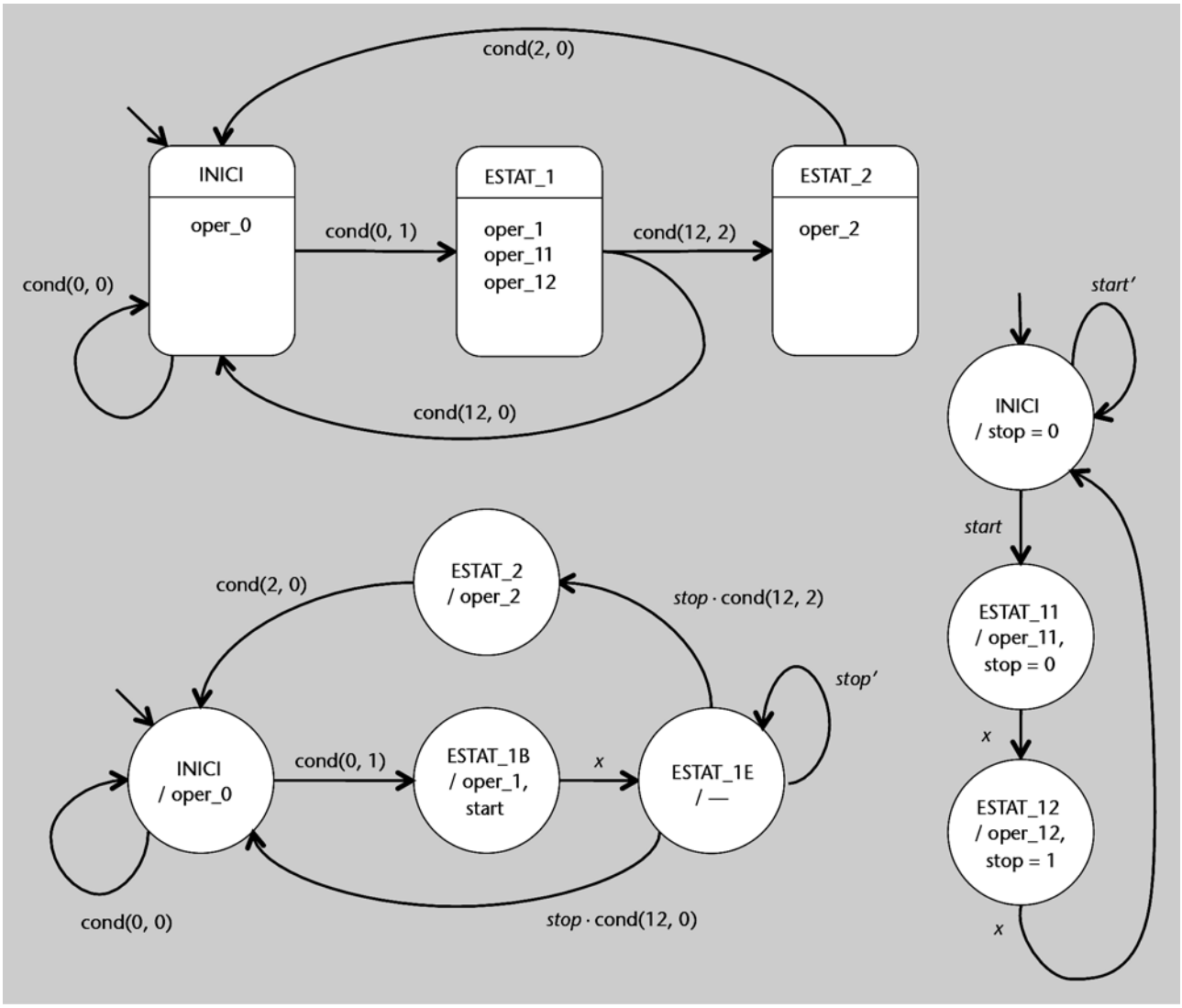
Com s'ha vist, les relacions entre les EFSM s'estableixen per mitjà d'estats d'espera i de referències d'estats externs a les entrades. Per exemple, que l'EFSM de la part inferior de la figura 23 passi de $MOURE_0$ a MOURE depèn d'una entrada que indiqui si l'EFSM principal és a l'estat MOVENT o no.

A més de l'afegiment dels estats d'espera, cal que l'EFSM principal tingui un senyal de sortida i un d'entrada per a cada EFSM secundària de manera que pugui dur a terme el programa corresponent i detectar quan s'ha acabat l'execució. Cal tenir present que, des de la perspectiva de les EFSM secundàries, els senyals són els mateixos però amb sentit diferent: les sortides de l'EFSM principal són les entrades i al revés.

Seguint amb l'exemple anterior, els senyals de sortida de l'EFSM principal es podrien denominar *iniLlegir* i *iniMoure* i servirien perquè les EFSM secundàries detectessin si la principal és a l'estat NOVA_R o a MOVENT, respectivament. Els senyals d'entrada serien *fiLlegir* i *fiMoure* per a indicar, a l'EFSM principal si les secundàries són als estats LLEGIT i MOGUT, respectivament.

En general, doncs, la implementació de PSM com a conjunt d'EFSM passa per una etapa de construcció de les EFSM de les seqüències d'accions per a cada estat-programa i la de l'EFSM global en la qual cal tenir en compte que s'hi han d'afegir estats d'espera i senyals per a la comunicació entre EFSM: un per a engegar el programa (*start*) i un altre, per a indicar-ne la finalització (*stop*).

Figura 24. Transformació d'una PSM en una jerarquia de dos nivells d'EFSM



En la figura 24 es veu aquest tipus d'organització amb un exemple senzill que segueix el de la figura 16. L'estat-programa ESTAT_1 es desdobla en dos estats: ESTAT_1B ("B" de *begin*) i ESTAT_1E ("E" d'*end*). En el primer, a més de fer la primera acció del programa, s'activa el senyal *start* perquè l'EFSM del programa corresponent passi a ESTAT_11. Després d'engegar-lo, la màquina principal passa a l'estat ESTAT_1E, en què espera que el senyal *stop* es posi a 1. Les condicions de sortida de l'estat-programa ESTAT_1 són les mateixes que les de l'ESTAT_1E, excepte perquè estan multiplicades lògicament per *stop*, ja que no es farà la transició fins que no es completi l'execució del programa. Per la seva banda, l'EFSM de control de programa, en arribar al darrer estat activarà el senyal *stop*.

La materialització d'aquests circuits és relativament senzilla, ja que n'hi ha prou de construir cada EFSM. Cal recordar que l'arquitectura de cada EFSM és d'FSMD, amb la unitat de control diferenciada de la unitat de procés.

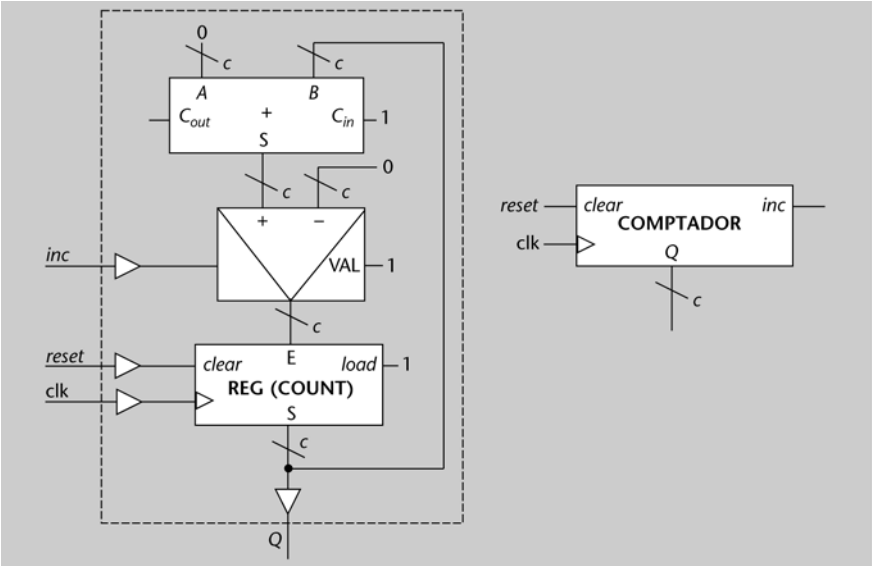
En els casos en què la part operacional sigui compartida, com en el de l'exemple del controlador de velocitat, es pot optar per integrar les unitats de control en una de sola. Aquesta part de control unificada consisteix en una FSM principal, que governa el conjunt i que activa un comptador cada cop que s'arriba a un estat-programa (amb més d'una acció en seqüència). El càlcul de l'estat següent d'aquesta FSM principal, amb el comptador activat, no ha de variar fins que s'arribi al final del compte per aquell estat. És a dir, mentre el comptador no hagi arribat al darrer valor a comptar, l'estat següent serà l'estat actual. El final del compte coincideix, evidentment, amb la compleció del programa associat.

Comptador amb senyal d'increment

Com ja s'ha vist, els comptadors són mòduls que tenen molts usos. Per a la construcció de PSM amb les EFSM principals i secundàries integrades es pot fer servir qualsevol dels comptadors que s'han vist amb anterioritat o bé un de més específic, un que no necessiti que se li doni el número fins al qual ha de comptar (el que s'havia denominat *M*). A canvi, haurà de tenir un senyal d'entrada específic que l'indiqui si ha d'incrementar el valor del compte o, al contrari, posar-lo a 0.

De fet, aquest comptador és com un comptador autònom en què, a cada estat, pot passar a l'estat inicial o al següent, segons el valor del senyal d'entrada d'increment, *inc*.

Figura 25. Esquema del circuit comptador amb senyal d'increment



Cal recordar que la distinció entre un tipus de mòdul comptador i un altre es pot fer tot veient les entrades i sortides que té.

En la figura 26 hi ha un esquema d'aquest model de construcció de PSM. En aquest cas, el circuit de càlcul de les sortides en genera una d'interna, *inc*, que permet activar el comptador. El compte es guarda en un registre específic (COUNT) del comptador que s'actualitza amb un zero, si *inc* = 0, o amb el valor anterior incrementat en una unitat, si *inc* = 1. Només en els estats-programa que tinguin més d'una acció en seqüència s'activarà aquesta sortida, que es retornarà a 0 quan se n'acabi l'execució.

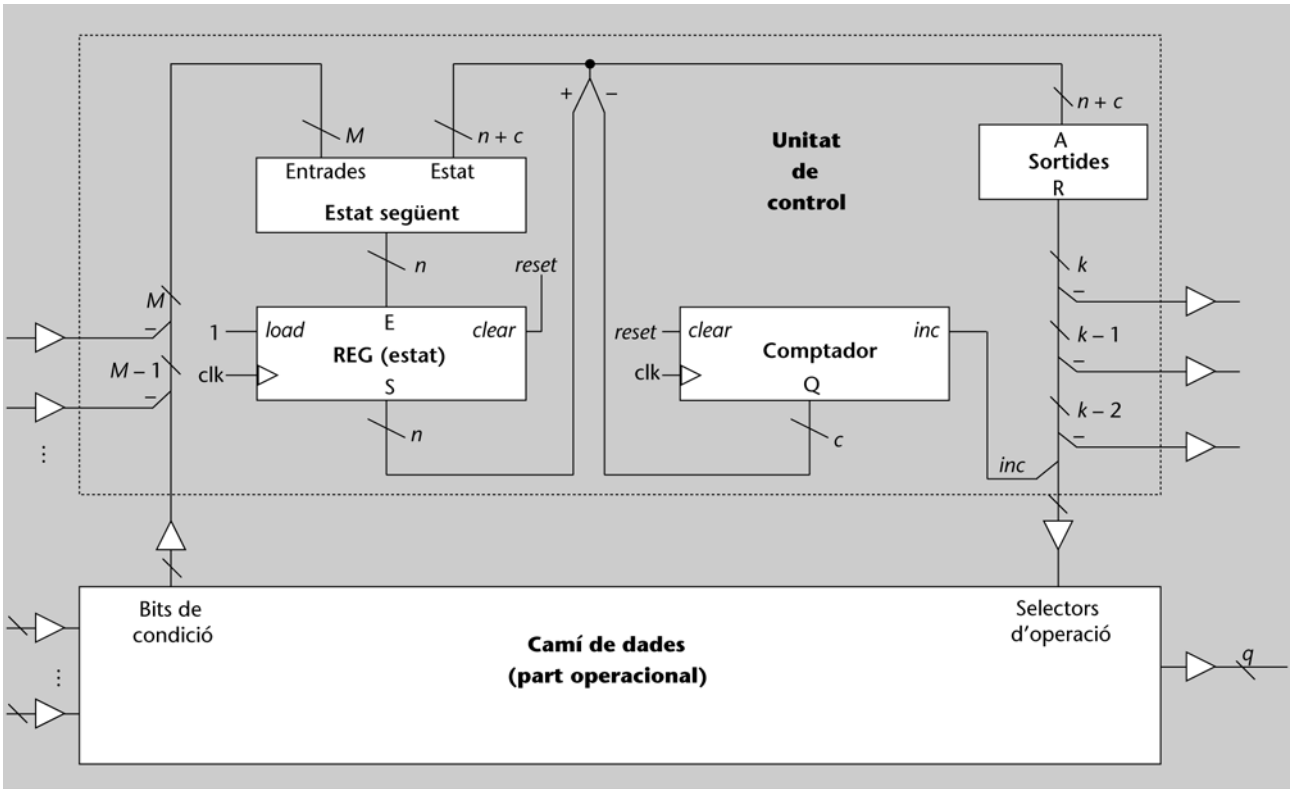
Així doncs, la unitat de control està constituïda per una màquina d'estats composta d'una FSM principal i diverses de secundàries que comparteixen un únic registre d'estat, que és el registre intern del comptador, COUNT. Això és possible perquè no estan mai actives al mateix temps. Consegüentment, la màquina d'estats que les engloba totes té un estat compost pel registre ESTAT i pel registre COUNT. Per aquest motiu, els busos cap als blocs combinacionals de càlcul d'estat següent i de càlcul de sortides són de $n + c$ bits.

Comptador dels estats-programa

El comptador dels estats-programa s'ocupa de fer el compte de la seqüència d'operacions que s'hi fan. Per això es denomina ***comptador de programa***. Com es veurà més endavant, els processadors també fan servir "comptadors de programa".

Com en el cas de qualsevol màquina d'estats, per a la materialització d'aquesta mena de PSM cal fer la taula de transicions i de sortides. Per al controlador de velocitat, la codificació dels estats s'ha fet seguint la numeració binària, deixant el codi 0 per a l'estat inicial.

Figura 26. Arquitectura d'una PSM amb TOC i camí de dades comú



Les entrades es corresponen amb els senyals de càrrega de nova referència (*ld_r*) i l'acabament del període d'un temporitzador (*end*). El valor de *count* fa

referència al contingut del registre COUNT del comptador de la unitat de control.

Estat actual		Entrades			Estat$^+$	Comentari
Símbol	S_{1-0}	*ld_r*	*end*	*count*	S_{1-0}	
INICI	00	0	x	x	00	
INICI	00	1	x	x	10	
MESURANT	01	0	0	x	01	
MESURANT	01	0	1	x	11	
MESURANT	01	1	x	x	10	
NOVA_R	10	x	x	00	10	1 pas del programa
NOVA_R	10	x	x	01	10	2 pas del programa
NOVA_R	10	0	x	10	01	3 pas del programa i sortida segons entrades
NOVA_R	10	1	x	10	10	
MOVENT	11	x	x	00	11	1 pas del programa
MOVENT	11	x	x	01	11	2 pas del programa
MOVENT	11	0	x	10	01	3 pas del programa i sortida segons entrades
MOVENT	11	1	x	10	10	

La taula de sortides associades a cada estat senzill i a cada pas d'un estat-programa és la que es mostra a continuació.

Estat actual		Pas de programa (*count*)	Sortides				
Símbol	S_{1-0}		*inc*	*begin*	*SelOpU*	*SelOpT*	*SelOpW*
INICI	00	x	0	0	0	00	0
MESURANT	01	x	0	0	0	00	0
NOVA_R	10	00	1	0	1	10	0
NOVA_R	10	01	1	0	0	00	0
NOVA_R	10	10	0	1	0	00	1
MOVENT	11	00	1	0	0	11	0
MOVENT	11	01	1	0	0	00	0
MOVENT	11	10	0	1	0	00	1

Val a dir que, per a les seqüències dels estats-programa, cal posar *inc* a 1 en totes els passos excepte en el darrer. Les columnes encapçalades amb *SelOp* són de selecció d'operació per a les variables afectades: *U*, *T* i *W*, d'esquerra a dreta en la taula. La variable *A* no hi apareix perquè es deixa que s'actualitzi sempre amb $M[T_{7-1} + T_0]$, que és l'únic valor que s'hi assigna en els dos programes.

Una cosa similar passa amb *U*, que només rep el valor de l'entrada *R*. En aquest cas, però, hi ha dues opcions: que *U* prengui un nou valor de *R* o que mantingui el que té. Amb vista a la implementació, s'ha codificat *SelOpU* de manera que coincideixi amb el senyal de càrrega del registre corresponent. La variable *T* té tres opcions: mantenir el contingut, carregar el de *R* o carregar el de $U + T - V$. Conseqüentment, *SelOpT* té tres valors possibles: 00, 10 i 11, respectivament. Amb aquesta codificació, el bit més significatiu serveix com a indicador de càrrega de nou valor per al registre corresponent.

Finalment, la variable *W*, que és la que proporciona la sortida, pot tenir els tres valors següents: 0 (INICI), *W* (MESURANT) i el càlcul de l'amplada (NOVA_R i MOVENT). Per simplificar-ne la implementació, atès que només es posa a 0 en l'estat inicial i que s'hi ha de posar immediatament (l'operació és $W = 0$, no pas $W^+ = 0$), se suposa que la inicialització del registre associat es farà havent arribat a l'estat INICI, juntament amb la de tots els altres registres de la màquina. Així, *SelOpW* només ha de discriminar entre si es manté el valor o si s'actualitza i es pot fer correspondre amb el senyal de càrrega del registre associat.

Finalment, cal fer una consideració quant als accessos a la memòria ROM: les adreces són diferents en el pas 1 i en el pas 2. La distinció es fa aprofitant el senyal de *SelOpW*: quan sigui 1, l'adreça serà T_{7-1} i, quan sigui 0, $T_{7-1} + T_0$.

A partir de les taules anteriors es pot generar el circuit. Cal tenir en compte que, pel que fa a les funcions de les sortides, la taula és incompleta, ja que hi ha casos que no es poden donar mai (el comptador a 11) i que es poden aprofitar per a obtenir expressions mínimes per a cada cas.

Seguint l'arquitectura que s'ha presentat, es pot construir el circuit seqüencial corresponent, tot implementant les funcions de les taules anteriors. El circuit està fet a partir de les expressions mínimes, en suma de productes, de les funcions.

L'esquema del circuit resultant es presenta en la figura 27, organitzat en dos blocs: el de control en la part superior, i el de processament de dades en la inferior. Com es pot comprovar, és convenient respectar l'organització de l'arquitectura d'FSMD i separar les unitats de control i de procés.

Abans d'acabar, cal fer notar que es tracta d'un controlador de velocitat bàsic, que només té en compte les situacions més habituals.

Portes NOT

Per simplicitat, els inversors o portes NOT en les entrades de les portes AND se substitueixen gràficament per boles. És una opció freqüent en els esquemes de circuits. Per exemple, les portes AND de la figura següent implementen la mateixa funció.

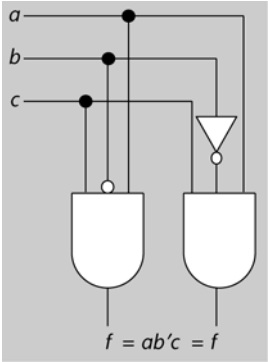
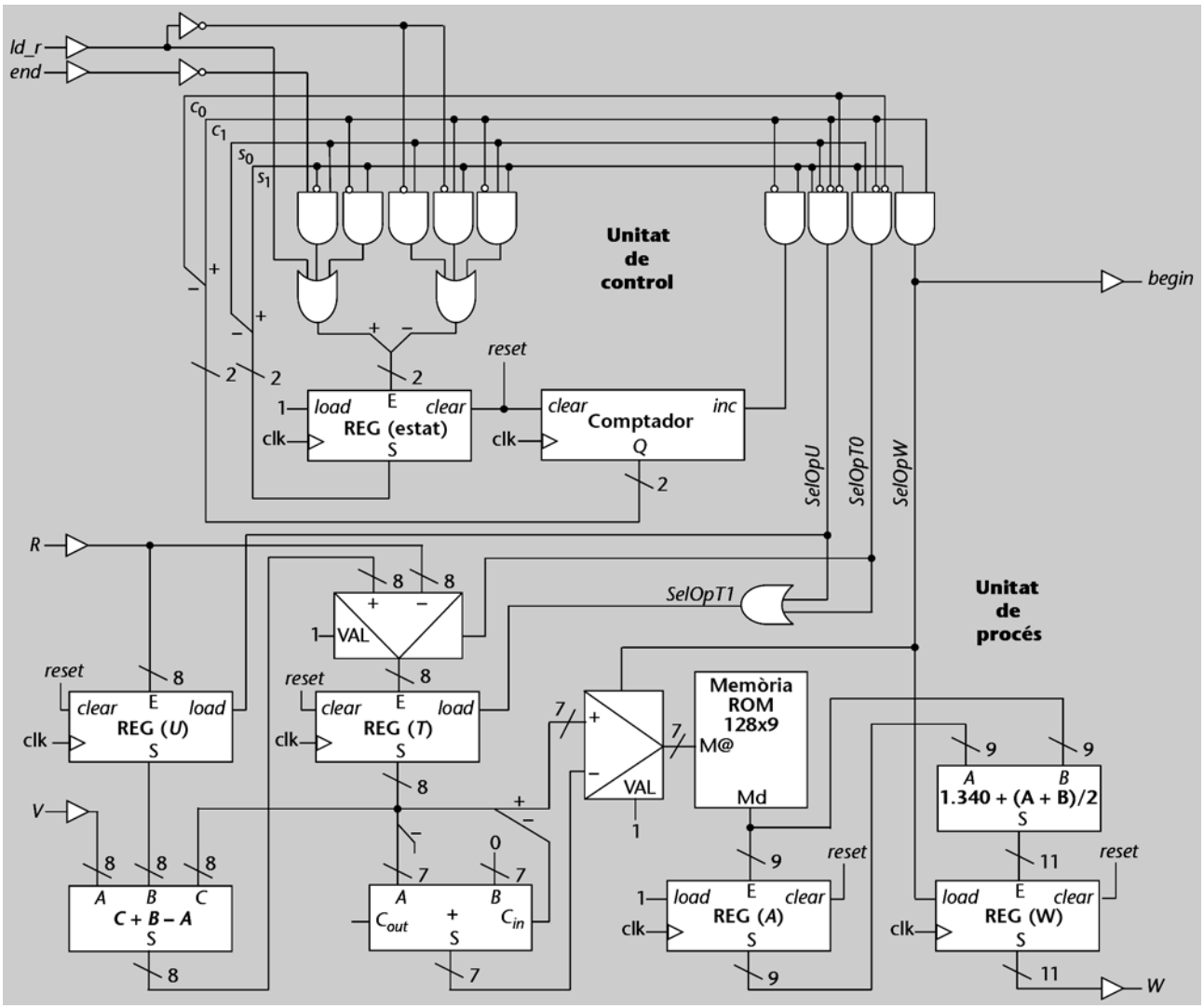


Figura 27. Circuit del control adaptatiu de velocitat



Activitats

6. Per completar el circuit del controlador de velocitat, dissenyeu els mòduls combinacionals corresponents als càlculs $C + B - A$ i $1.340 + (A + B) / 2$. Pareu atenció als diferents formats dels nombres.

7. El model del controlador de velocitat que s'ha vist rep dos senyals del controlador global: *ld_r* i *R*. El senyal *ld_r* actua exclusivament com a indicador de quan el contingut de *R* ha canviat i cal que el controlador de velocitat carregui el nou valor de referència. Per tant, des del punt de vista del controlador global, n'hi hauria prou d'efectuar un canvi a *R* perquè el controlador de velocitat fes la càrrega de la nova referència i, d'aquesta manera, no caldria fer servir *ld_r*. Com que el valor de la velocitat de referència es desa en la variable *U*, el controlador de velocitat pot determinar si hi ha canvis en el valor de l'entrada *R* o no. Modifiqueu el diagrama de la PSM per reflectir aquest canvi. Com canviaria el circuit?

1.4. Màquines d'estats algorísmiques

Les màquines d'estats que s'han vist fins ara encara presenten l'inconvenient d'haver de representar completament totes les possibles condicions de transició. Certament, es pot optar per suposar que les que no s'associen a cap arc de sortida són de manteniment en l'estat actual. Amb tot, encara continua essent

necessari posar de manera explícita els casos que impliquen transició cap a un estat diferent de l'actual.

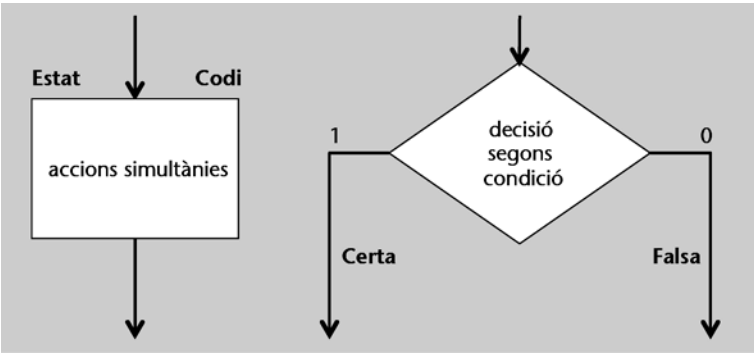
Vist des d'un altre punt de vista, els diagrames d'estats han d'anar acompanyats, necessàriament, de les condicions (entrades) i de les accions (sortides) que s'estableixen per a cada arc i node que tenen. Això fa que, quan el nombre d'entrades i sortides creix, resultin farragosos de tractar. Les PSM resolen el problema de les sortides quan es tracta de seqüències d'accions, però continuen requerint que, per a cada estat, s'especifiquin tots els casos possibles d'entrades i els estats següents per a cada cas.

De fet, en la majoria d'estats, hi ha poques sortides que canvien i les transicions cap als estats següents també depenen d'unes poques entrades. Els grafs de transicions d'estats es poden representar d'una manera més eficient si es té en compte aquest fet.

Les **màquines d'estats algorísmiques** o **ASM** (de l'anglès, *algorithmic state machines*) són màquines d'estats que relacionen les diverses accions, que estan lligades als estats, mitjançat condicions determinades en funció de les entrades que són significatives per a cada transició.

En els diagrames de representació de les ASM hi ha dos tipus de nodes bàsics: les caixes d'estat (rectangulars) i les de decisió (romboides). Les caixes d'estat s'etiqueten amb el nom de l'estat i el seu codi binari, habitualment en la part superior, tal com apareix en la figura 28. A dins es posen les accions que s'han de dur a terme en aquest estat. Són accions que es fan simultàniament, és a dir, en paral·lel. Les caixes de decisió tenen dues sortides, segons si la condició que s'expressa en el seu interior és certa (arc etiquetat amb el bit 1) o falsa (arc etiquetat amb el bit 0).

Figura 28. Tipus de nodes en una màquina d'estats algorísmica

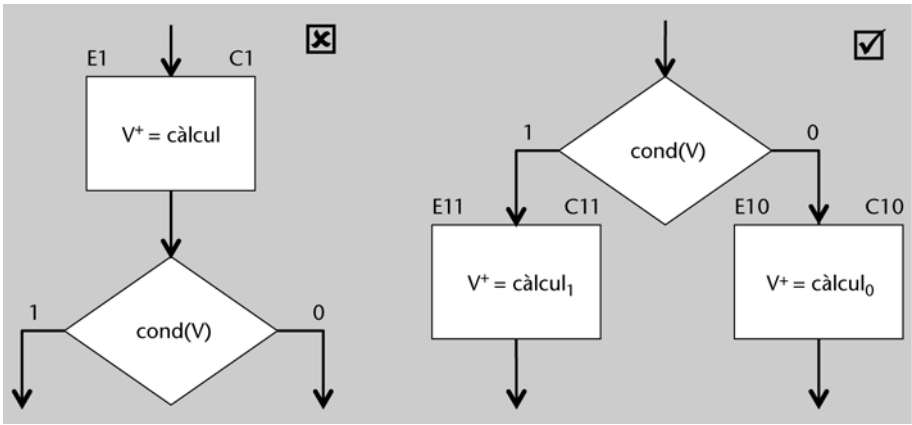


En comparació als grafs que representen EFSM, les caixes d'estats es corresponen als nodes, i les de decisió, als arcs.

Com amb qualsevol màquina d'estats, cal tenir en compte que les condicions es calculen a partir de l'estat mateix i dels valors emmagatzemats en aquest

moment en les variables de la mateixa màquina, però no pas dels continguts futurs. Per a evitar confusions, és recomanable que els diagrames dels ASM posin primer les caixes de decisió i després les d'estat, i no pas al revés, tal com s'il·lustra en la figura 29.

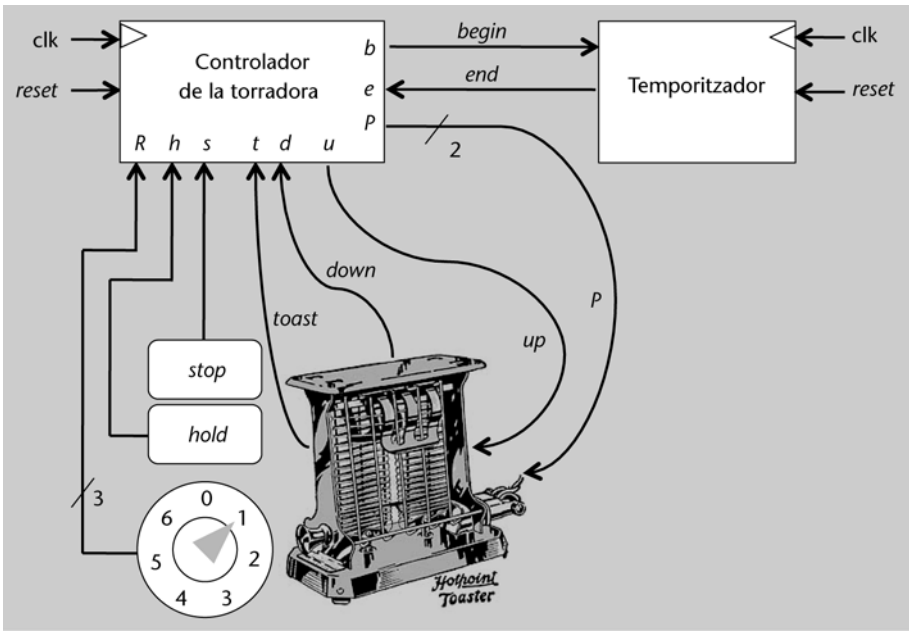
Figura 29. Formes desaconsellada i aconsellada de col·locar estats amb càlculs que afecten bits de condició.



Per a veure com es representa el funcionament d'un controlador amb una ASM s'estudiarà el cas per a una torradora de pa. Es tracta d'un dispositiu relativament senzill que té dos botons: un (*stop*) per a interrompre'n el funcionament i fer saltar la torrada, i un altre (*hold*) per a mantenir-la calenta a dins fins que no es premi el botó anterior. A més, té una roda que permet seleccionar sis graus diferents de torrat i que també pot estar apagada (posició 0). Per a engegar-la només cal posar el selector en una posició diferent de 0 i fer baixar la llesca de pa. Hi ha un sensor per a indicar quan s'abaixa el suport del pa (*down*) i un altre per a detectar la presència de la llesca (*toast*). Quant a les sortides del controlador, n'hi ha una per a fer pujar la torrada (*up*) i una altra per a indicar la potència dels elements calefactors (*P*). Per a tenir en compte el temps de cocció, hi ha un temporitzador que serveix per a fer una espera d'un període prefixat. Així, el procés de torrada durarà tants períodes de temps com el número de posició que indiqui el selector.

L'esquema de la torradora es mostra en la figura 30 (la torradora representada és un model del principi del segle XX que, evidentment, tenia un funcionament ben diferent del presentat). En la figura també hi ha el nom dels senyals per al controlador. Cal tenir present que la posició de la roda es llegeix per mitjà del senyal *R* de 3 bits, els valors possibles són en el rang [0, 6], i que la potència *P* dels elements calefactors es codifica de manera que 00 els apaga totalment, 10 els manté per donar un calor mínim de manteniment de la torrada calenta, i 11 els encén per irradiar calor suficient per a torrar. Amb $P = 11$, la llesca de pa es torrarà més o menys segons *R*. De fet, la posició de la roda indica la quantitat de períodes de temps en què es farà la torrada. A més valor, més lapses de temps i més torrada quedarà la llesca de pa. Aquest període de temps es controlarà amb un temporitzador, mentre que el nombre de períodes es comptarà internament, amb l'ASM corresponent, tot fent servir una variable auxiliar per al compte *C*.

Figura 30. Esquema de blocs d'una torradora



En aquest cas, l'elevat nombre d'entrades del controlador ajuda a mostrar els avantatges de la representació del seu comportament amb una ASM. En la figura 31 es pot observar el diagrama complet.

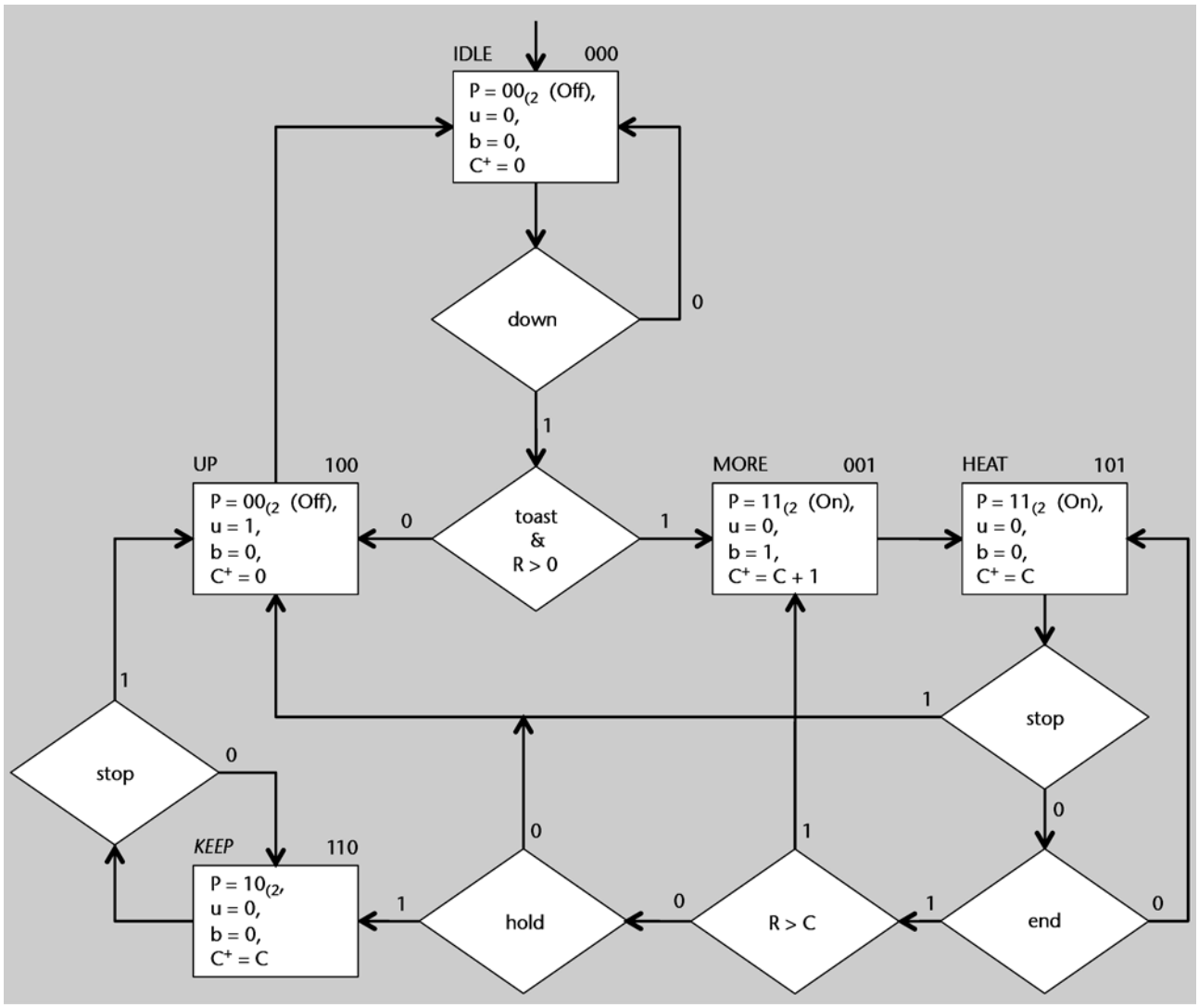
És recomanable que les accions i les condicions que es descriuen en les caixes del diagrama d'estats siguin el més generals possible. D'aquesta manera són més entenedores i no queden vinculades a una implementació concreta.

A tall d'exemple, en les caixes de decisió només hi ha la indicació textual de les condicions que s'han de complir. Per comparació, en les caixes d'estat, les accions es descriuen directament amb els canvis als senyals de control i variables, tal com apareixen simbolitzades en la part interior del mòdul del controlador de la torradora en la figura 30. Es pot comprovar que, en un cas una mica més complex, les accions serien més difícils d'interpretar. En canvi, les condicions resulten més entenedores i, a més, es poden traslladar fàcilment cap a expressions que, al seu torn, es poden materialitzar de manera força directa.

Cal tenir present la distinció entre variables i senyals de sortida: els valors dels senyals de sortida són els que es calculen en cada caixa d'estat, mentre que els valors de les variables es mantenen durant tot l'estat i només s'actualitzen amb el resultat de les expressions que hi ha en la caixa d'estat quan es passa a l'estat següent.

L'estat inicial de l'ASM és IDLE, en què els radiadors de la torradora estan apagats ($P = 00$), no es fa cap acció (*up* o *begin*), ni fer saltar la llesca de pa ($u = 0$) ni engegar el temporitzador ($b = 0$), i a més es posa a zero la variable C, cosa que tindrà efecte per a l'estat següent, tal com s'indica amb el símbol més en posició de superíndex.

Figura 31. Diagrama d'una ASM per al controlador d'una torradora



En l'estat IDLE s'espera que es posi una llesca de pa a la torradora. Per això en l'arc de sortida la condició que s'ha de satisfer és *down*. Si no s'hi ha posat res, la màquina es queda en el mateix estat. Si no, passa a comprovar que, efectivament, s'hi ha inserit una llesca de pa (*toast*) i que s'ha posat la roda en una posició que no sigui la d'apagat ($R > 0$). En funció del resultat d'aquesta segona condició, la màquina passarà a l'estat UP o a l'estat MORE. En el primer cas, es farà pujar el suport del pa (o bé no s'hi havia ficat res o bé hi havia una llesca però la roda era en la posició 0) i es retornarà, finalment, a l'estat inicial d'espera. En el segon cas, s'engegaran els calefactors per a iniciar la torrada. També es posarà el comptador intern d'intervals de temps a 1 i s'activarà un temporitzador perquè avisi del moment en què s'acabi el període de temps de l'interval actual.

És interessant observar com, en les ASM, les caixes d'estat només tenen un arc de sortida que es va dividint a mesura que travessa caixes de condició. Per exemple, l'arc de sortida d'IDLE té tres destinacions possibles: IDLE, MORE i UP, segons les condicions que se satisfacin. De manera ben diferent de com s'hauria de fer en els models de PSM i EFSM, en els quals un estat similar hauria de tenir tres arcs de sortida amb expressions de condició completes per a cada arc.

Tal com s'ha comentat, el nombre d'intervals de temps que han de transcórrer per a fer la torrada està determinat per la intensitat de la torrada, que es regula mitjançant la posició de la roda de control, que pot variar d'1 a 6. En l'estat MORE s'incrementa el comptador intern, C, que té la funció de comptar quants intervals de temps ha estat en funcionament.

D'aquest estat es passa al de torrar (HEAT), en què es queda fins que no es premi el botó de parada (*stop*) o no s'acabi el període de temps en curs (*end*). En el primer cas, passarà a UP per fer pujar la torrada tal com estigui en aquest moment i retornarà, finalment, a IDLE. En el segon cas, comprovarà quants períodes han de passar encara fins a aconseguir la intensitat desitjada de torrada. Si $R > C$ torna a MORE per incrementar el comptador intern i iniciar un nou període d'espera.

L'arc de sortida de HEAT encara té una derivació més. Si ja s'ha completat la torrada ($R > C$) cal fer pujar la llesca (anar a UP) a menys que el botó de manteniment (*hold*) estigui premut. En aquest cas, es passa a un estat d'espera (KEEP) que deixa els elements calefactors en una potència intermèdia que mantingui la torrada calenta. En prémer el botó d'aturada, es passa a UP per a fer saltar la torrada i apagar la torradora.

És important tenir en compte que s'ha seguit la recomanació exemplificada en la figura 29: la caixa de condició en funció d'una variable ($R > C$) precedeix una d'estat que la modifica (MORE).

Per a la materialització de l'ASM cal representar les expressions de totes les caixes amb fórmules lògiques. Per aquest motiu convé fer el mateix que s'ha fet amb les caixes d'estat: substituir els noms de les condicions per les expressions lògiques corresponents. En la taula següent, es mostra aquesta relació.

Condició	Expressió lògica
down	d
toast & $R > 0$	$t \cdot (R > 0)$
stop	s
end	e
$R > C$	$(R > C)$
hold	h

Les funcions que calculen l'estat següent es poden obtenir a partir de la taula de veritat de les funcions de transició. En el cas de les ASM, hi ha una altra opció en què l'estat següent es calcula a partir dels resultats de l'avaluació de les condicions que hi ha en les caixes de decisió. La idea del mètode és que la transició d'un estat a un altre es fa canviant només els bits que difereixen d'un codi

d'estat a l'altre, segons les avaluacions de les condicions de les caixes que els relacionen.

Amb aquest segon mètode, la construcció de les expressions lògiques per al càlcul de l'estat següent és directa des de les ASM. Perquè sigui eficient, és convenient que la codificació dels estats faci que dues caixes d'estat veïnes tinguin codis en què canvien el mínim nombre possible de bits.

Càlcul de l'estat següent a partir de les diferències entre codis d'estats

A mode il·lustratiu, se suposa l'exemple següent: una ASM és a l'estat $(s_3, s_2, s_1, s_0) = 0101$ i, amb les entrades $a = 1$ i $b = 1$, ha de passar a l'estat 0111. En la manera convencional, les funcions de l'estat següent, representades en suma de productes, haurien d'incloure el terme $s'_3 \cdot s_2 \cdot s'_1 \cdot s_0 \cdot a \cdot b$, és a dir:

$$s^+_3 = s^*_3$$
$$s^+_2 = s^*_2 + s'_3 \cdot s_2 \cdot s'_1 \cdot s_0 \cdot a \cdot b$$
$$s^+_1 = s^*_1 + s'_3 \cdot s_2 \cdot s'_1 \cdot s_0 \cdot a \cdot b$$
$$s^+_0 = s^*_0 + s'_3 \cdot s_2 \cdot s'_1 \cdot s_0 \cdot a \cdot b$$

en què s^*_i fa referència als altres termes producte de la funció corresponent, que són els altres 1 que té en la taula de veritat associada. En l'expressió de s^+_3 no s'afegeix el terme anterior perquè la funció no és 1 en aquest cas d'estat actual i condicions.

Si només es tinguessin en compte els bits que canvien, les funcions que calculen l'estat següent serien del tipus:

$$s^+_3 = s_3 \oplus c_3$$
$$s^+_2 = s_2 \oplus c_2$$
$$s^+_1 = s_1 \oplus c_1$$
$$s^+_0 = s_0 \oplus c_0$$

en què c_i fa referència a la funció de canvi de bit. Cal tenir en compte que:

$$s_i \oplus 0 = s_i$$
$$s_i \oplus 1 = s'_i$$

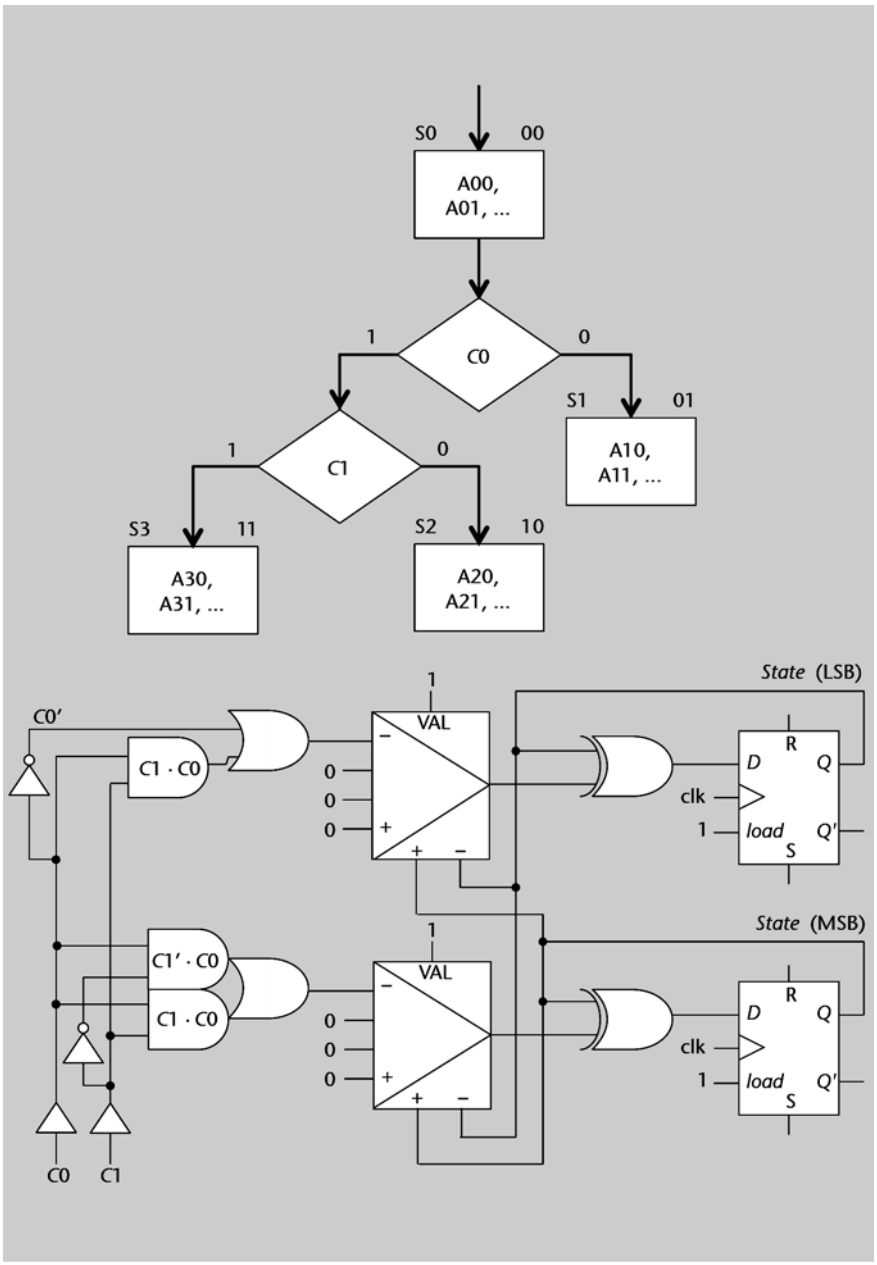
Continuant amb l'exemple, per a construir les expressions de les funcions de canvi c_i, s'ha de veure quins bits entre el codi d'estat origen i destinació canvien i afegir el terme corresponent a la suma de productes que toqui. En aquest cas, només canvia el bit que hi ha en la posició 1:

$$c_3 = c^*_3$$
$$c_2 = c^*_2$$
$$c_1 = c^*_1 + s'_3 \cdot s_2 \cdot s'_1 \cdot s_0 \cdot a \cdot b$$
$$c_0 = c^*_0$$

en què c^*_i representa l'expressió en sumes de productes per als altres canvis en el bit ièssim del codi d'estat. D'aquesta manera, sempre que els codis d'estats veïns siguin propers (és a dir, que no difereixin en massa bits), les funcions de càlcul d'estat següent tenen una representació molt directa i senzilla, amb pocs termes producte.

Per a veure-ho més clar, en la figura 32 hi ha una ASM i el circuit que fa el càlcul de l'estat següent. Com que tots els termes que se sumen a les funcions de canvi inclouen el producte amb l'estat actual, s'opta per fer servir multiplexors que, a més, ajuden a fer la suma final de tots els termes (en aquest cas, com que només hi ha un arc de sortida per a l'estat *S*0, els multiplexors només serveixen per a fer el producte per $s'_1 \cdot s'_0$). Per a cada estat, que el bit canviï o no, depèn del codi de l'estat de destinació. En l'exemple, només es pot passar de l'estat *S*0 (00) als *S*1 (01), *S*2 (10) i *S*3 (11). I els canvis consisteixen a posar a 1 el bit que toqui. Perquè canviï el bit menys significatiu (LSB) de l'estat, *C*0 ha de ser 0 (branca dreta de la primera caixa de decisió), o *C*1 i *C*0 han de ser 1 alhora (branques esquerres de l'esquema). Perquè canviï l'MSB de l'estat, s'ha de complir que $C1' \cdot C0$ o que $C1 \cdot C0$. És a dir, n'hi ha prou que *C*0 sigui 1. En la figura, el circuit no està minimitzat per a il·lustrar bé el model de construcció del càlcul de l'estat següent.

Figura 32. Model constructiu de la unitat de control d'una ASM



En el circuit de la figura 32 s'han deixat els dos multiplexors que serveixen per a fer el producte lògic entre els estats origen i les condicions lligades als arcs de sortida que porten a estats destinació que tenen un codi en què el bit corresponent és diferent del dels estats origen. Com que en l'exemple només s'han vist els arcs de sortida de l'estat *S*0, només s'aprofita l'entrada 0 dels multiplexors. Si les altres caixes d'estat tinguessin arcs de sortida, llavors les condicions corresponents serien les entrades dels multiplexors en les posicions 1, 2 o 3.

Les unitats de control que segueixen aquesta arquitectura es poden construir sense necessitat de calcular explícitament les funcions de càlcul de l'estat següent. Cosa molt avantatjosa per als casos en què el nombre d'entrades és relativament gran.

Per a aquest tipus de càlculs, la codificació dels estats s'ha de fer de manera que entre dos estats veïns (és a dir, que es pugui anar d'un a l'altre sense que n'hi hagi cap altre entremig) canviï el mínim nombre possible de bits.

Per a construir el circuit seqüencial de la torradora, que correspon a l'ASM de la figura 31, primer cal obtenir les equacions que calculen l'estat següent. Per a això, se segueix l'arquitectura que s'acaba d'explicar.

Per a fer-ho, n'hi ha prou a determinar les expressions per a les funcions de canvi, (c_2, c_1, c_0). Una manera de procedir és, estat per estat, mirar quins termes s'hi han d'afegir, segons l'estat destinació i els bits que canvien. Per exemple, des d'IDLE (000) es pot anar a IDLE (000), MORE (001) o UP (100). D'un estat a ell mateix no s'afegeix cap terme a la funció de canvi perquè, òbviament, no canvia cap bit del codi d'estat. De l'estat IDLE a MORE canvia el bit en posició 0, per tant, cal afegir el terme corresponent, $[s'_2 s'_1 s'_0] \cdot dtr$, a la funció de canvi c_0. A les altres funcions no cal afegir-hi res. Finalment, d'IDLE a UP només canvia el bit més significatiu i cal afegir el terme $[s'_2 s'_1 s'_0] \cdot d(t' + r')$ a la funció de canvi c_2. Com tots els termes, una part la constitueix la que és 1 per a l'estat origen, i l'altra, el producte de les expressions lògiques de les condicions que porten al de destinació. En aquest cas, que $d = 1$ i que $(t \cdot r = 0)$, és a dir, que es compleixi que $d \cdot (t \cdot r)' = 1$ o, el que és el mateix, que $d(t' + r') = 1$.

Al final del procés, les expressions resultants són les següents:

$$s^+_2 = s_2 \oplus ([s'_2 s'_1 s'_0] \cdot d(t' + r') + [s'_2 s'_1 s_0] \cdot 1 + [s_2 s'_1 s'_0] \cdot 1 + [s_2 s'_1 s_0] \cdot s'er)$$
$$s^+_1 = s_1 \oplus ([s_2 s'_1 s_0] \cdot s'ehr' + [s_2 s_1 s'_0] \cdot s)$$
$$s^+_0 = s_0 \oplus ([s'_2 s'_1 s'_0] \cdot dtr + [s_2 s'_1 s_0] \cdot (s + er'))$$

en què el senyal r representa el resultat de comprovar si $R > C$, que és equivalent a comprovar $R > 0$ en la caixa de decisió corresponent perquè només s'hi pot arribar de l'estat IDLE, en què C es posa a 0. Com que, de fet, C es posa a 0 en iniciar-se l'estat següent i el valor actual en IDLE podria no ser 0 si l'estat immediatament anterior hagués estat UP, també cal que es faci $C^+ = 0$ a UP.

La part operacional només ha de materialitzar els pocs càlculs que hi ha, que són:

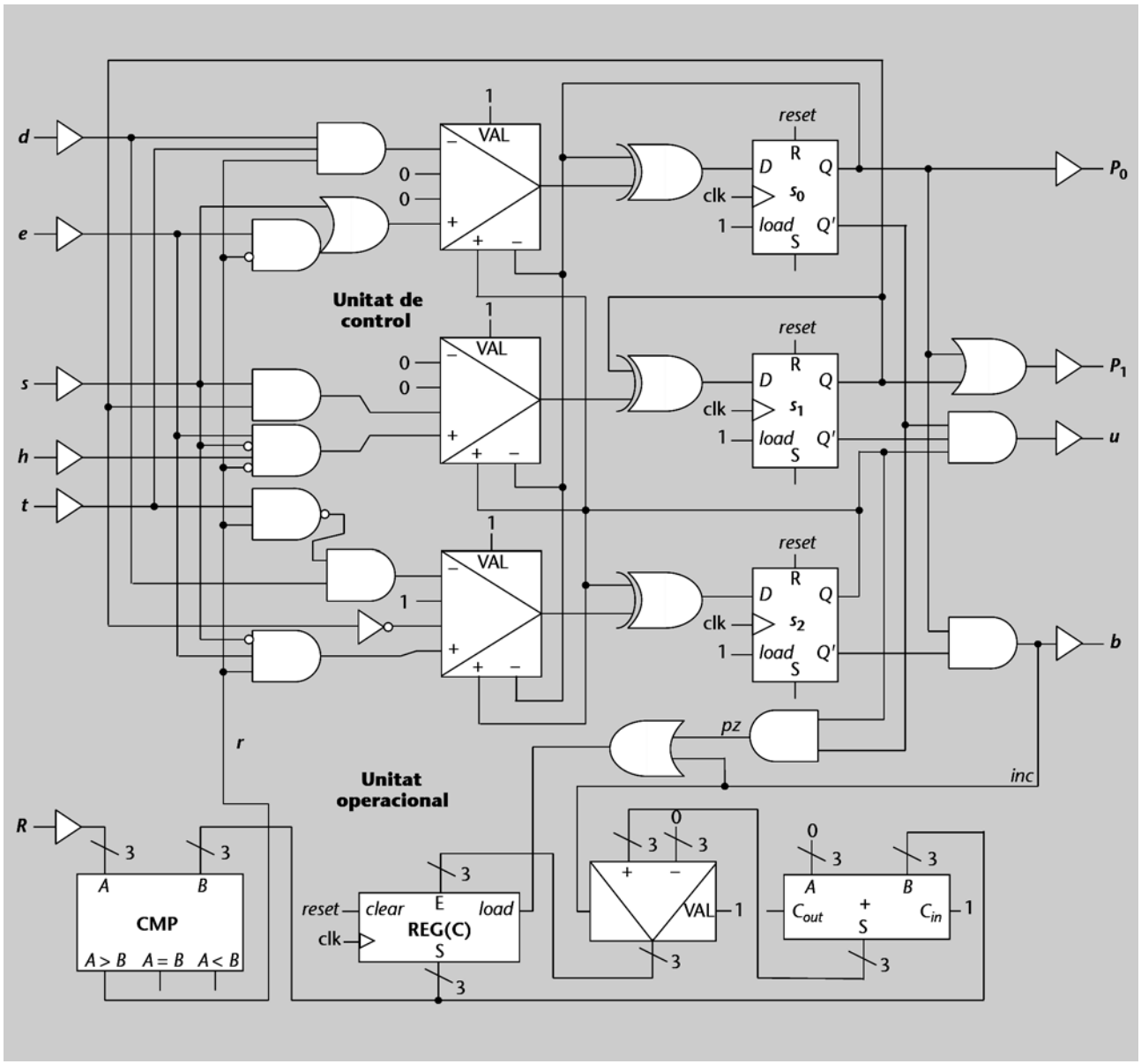
- Per a un registre de compte C, $C^+ = 0$ i $C^+ = C + 1$, que se seleccionaran amb els senyals *inc*, d'increment, i *pz*, de posar a 0.
- La comparació amb la referència R, $R > C$, que dóna el bit de condició r.

Les funcions de sortida (*inc*, *pz*, *u*, *b*, *P*) es calculen a partir de l'estat i són les següents:

$$inc = s'_2\, s'_1 s_0 \ \{= \text{MORE}\}$$
$$pz = s'_2\, s'_1 s'_0 + s_2 s'_1 s'_0 \ \{= \text{IDLE} + \text{UP}\}$$
$$u = s_2 s'_1 s'_0 \ \{= \text{UP}\}$$
$$b = s'_2\, s'_1 s_0 \ \{= \text{MORE}\}$$
$$p_1 = s'_2\, s'_1 s_0 + s_2 s'_1 s_0 + s_2 s_1 s'_0 \ \{= \text{MORE} + \text{HEAT} + \text{KEEP}\}$$
$$p_0 = s'_2\, s'_1 s_0 + s_2 s'_1 s_0 \ \{= \text{MORE} + \text{HEAT}\}$$

Amb tot, l'esquema del controlador de la torradora és el que es mostra en la figura 33.

Figura 33. Esquema del circuit del controlador d'una torradora



El circuit inclou una optimització basada a aprofitar com a *don't-cares* els codis d'estat que no es fan servir, que són: 010, 011 i 111. Així doncs, per a saber si la màquina és a l'estat 000 n'hi ha prou de mirar els dos bits dels extrems, ja que 010 no es pot donar mai. De manera semblant, això passa amb 001 i 101 respecte dels codis irrellevants 011 i 111, respectivament. En el cas dels estats 100 i 110, cal introduir una lògica mínima de diferenciació. Això val la pena a canvi de fer servir multiplexors de selecció d'estat d'ordre 2 en lloc dels d'ordre 3.

En la part inferior de la figura es veu la unitat de processament, amb el comparador per a calcular *r*, i també com s'han implementat els càlculs per a la variable *C*. En aquest cas, el multiplexor selecciona si el valor en l'estat següent de la variable, C^+, ha de ser 0 o $C + 1$, i la senyal de càrrega del registre corresponent s'utilitza per a determinar si ha de canviar, cas que es dóna quan *inc* + *pz*, o si ha de mantenir el contingut que ja tenia.

Activitats

8. Construïu la taula de veritat de les funcions de transició del controlador de la torradora.

9. Modifiqueu l'ASM de manera que, quan s'hi posi una llesca de pa, no s'expulsi automàticament si $R = 0$. En altres paraules, que passi a un estat d'espera (WAIT) fins que $R > 0$ o fins que es premi *stop*.

En general, la materialització d'una màquina d'estats es fa partint de la representació que sigui més adequada. Les FSM que treballen amb senyals binaris són més senzilles d'implementar, però és més dificultós veure'n la relació amb la part operacional. Les EFSM solucionen aquest problema integrant càlculs en el model de sistema. Segons com, però, és convenient empaquetar aquestes seqüències de càlcul (PSM) i simplificar les expressions de les transicions (ASM).

2. Màquines algorísmiques

Les màquines d'estats serveixen per a representar el comportament dels circuits de control d'una multitud de sistemes ben diversos. Però no són tan útils a l'hora de representar els càlculs que s'hi han de dur a terme, sobretot, si gran part de la funcionalitat del sistema consisteix en això mateix. De fet, el comportament dels sistemes que fan molts de càlculs o, en altres paraules, que fan molt de processament de la informació que reben es pot representar millor amb algorismes que no pas amb màquines d'estats.

En aquesta part es tractarà de la materialització de sistemes en què el processament de les dades té més rellevància que la gestió de les entrades i sortides d'aquests. Dit d'una altra manera, són sistemes que implementen algorismes o, si es vol, màquines algorísmiques.

2.1. Esquemes de càlcul

En dissenyar màquines d'estats amb càlculs associats no s'ha tractat la problemàtica lligada a la implementació dels que són complexos.

En general, en un sistema orientat al processament de la informació, les expressions dels càlculs poden ser formades per una gran quantitat d'operands i d'operadors. A més, entre els operadors es poden trobar els més comuns i simples de materialitzar, com els lògics, la suma i la resta, però també productes, divisions, potenciacions, arrels quadrades, funcions trigonomètriques, etc.

Per a materialitzar aquests càlculs complexos és necessari descompondre'ls en d'altres de més senzills. Una manera de fer-ho és transformant-los en un programa, a la manera que s'ha vist per a les màquines d'estats-programa.

Segons com, però, és convenient representar el càlcul de manera que sigui fàcil analitzar-lo per a obtenir una implementació eficient del camí de dades corresponent, cosa que no s'ha vist en el tema de les PSM.

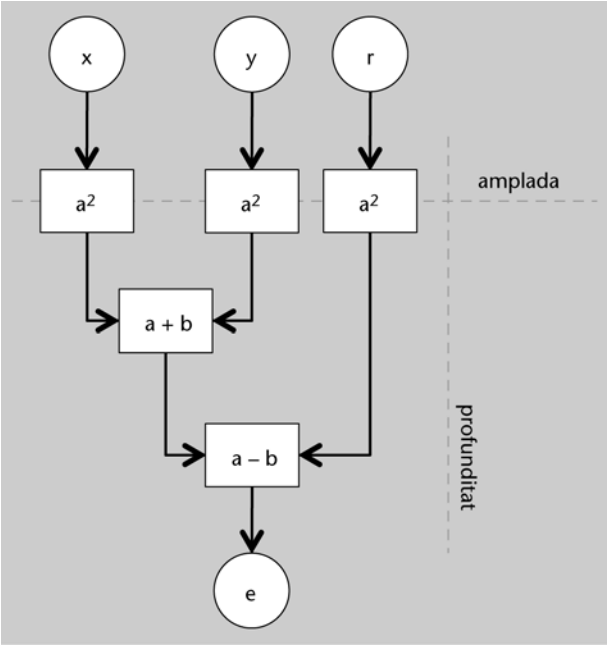
De fet, els esquemes de càlcul són representacions gràfiques de càlculs més o menys complexos que es manipulen amb vista a obtenir-ne una d'adequada per a les condicions del problema en el qual s'hagin de fer. De vegades interessarà que el càlcul es faci molt ràpid, de vegades que es faci amb el nombre mínim de recursos de càlcul possible.

En un esquema de càlcul hi ha dos tipus de nodes: els que fan referència als operands (cercles) i els que fan referència als operadors (rectangles). La relació

entre uns i altres es marca mitjançant arcs (fletxes) que indiquen el flux de les dades. Els operadors se solen descriure en termes d'operands genèrics, no vinculats a un càlcul específic (en els exemples que hi ha a continuació s'especifiquen amb operands identificats amb les lletres *a*, *b*, *c*...).

Per exemple, una expressió com $e = x^2 + y^2 - r^2$ es pot representar mitjançant l'esquema de càlcul de la figura 34.

Figura 34. Graf d'un esquema de càlcul



En l'esquema de càlcul anterior es pot veure la precedència entre les operacions, però també altres coses: l'**amplada del càlcul**, que fa referència al nombre de recursos de càlcul que es fan servir en una etapa determinada, i la **profunditat del càlcul**, que fa referència al nombre d'etapes que hi ha fins a obtenir el resultat final.

Els recursos de càlcul són els elements (habitualment, blocs lògics o mòduls combinacionals) que duen a terme les operacions. Les etapes en un càlcul queden constituïdes per totes les operacions que es poden fer concurrentment (en paral·lel) perquè els resultats són independents. En la figura 34, n'hi ha tres: una per al càlcul dels quadrats, una altra per a la suma, i la final per a la resta.

Pot passar que hi hagi operacions que es puguin resoldre en diverses etapes: en el cas de la figura 34, el càlcul de r^2 es pot fer en la primera o en la segona etapa, per exemple.

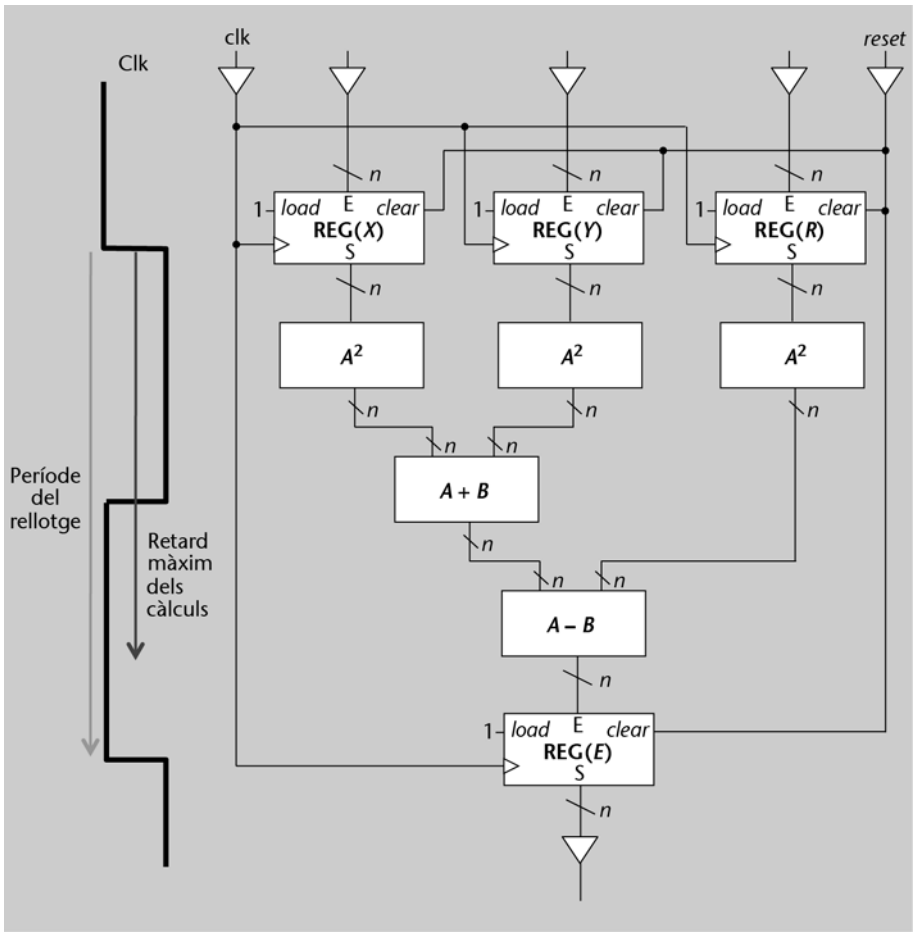
La profunditat mínima del càlcul és el nombre mínim d'etapes que pot tenir. Com més gran és, més complex és el càlcul. En particular, si se suposa que els operands s'emmagatzemen en registres, igual que el resultat, el càlcul es veu com una transferència entre registres. Per a l'exemple anterior, seria:

$$E^+ = X^2 + Y^2 - R^2$$

Seguint amb aquesta suposició, els càlculs molt profunds necessitaran un temps de cicle del rellotge molt gran (vegeu la figura 35). Com s'ha dit, el temps de processament depèn de la profunditat del càlcul, ja que, com més nivells d'operadors, més etapes de blocs lògics combinacionals han de "travessar" les dades per a arribar a produir el resultat final. De fet, es pot atribuir un temps de retard a cada operador, de manera similar a com es fa amb les portes lògiques, i determinar el retard que introdueix un determinat càlcul. Si, com s'ha suposat, les dades d'entrada provenen de registres i el resultat s'emmagatzema en un altre registre, el període de rellotge que els controla ha de ser més gran que el retard dels càlculs per a obtenir el resultat que s'hi ha de carregar.

Això pot anar en detriment del rendiment del circuit en cas que hi hagi càlculs molt complexos, i profunds, i d'altres de més senzills, ja que el període del rellotge s'ha d'ajustar al pitjor cas.

Figura 35. Circuit d'un esquema de càlcul amb cronograma lateral



D'altra banda, els càlculs que tinguin una gran amplitud s'han de dur a terme amb molts recursos que poden no aprofitar-se plenament quan s'avaluïn d'altres expressions, cosa que també té una influència negativa en la materialització dels circuits corresponents: són prou grans per a encabir la màxima amplitud de càlcul, encara que la major part del temps no la necessitin.

En els propers apartats es veurà com es resolen aquests problemes i com s'implementen els circuits corresponents.

2.2. Esquemes de càlcul segmentats

Els esquemes de càlcul permeten manipular, de manera gràfica, les operacions i els operands que intervenen en un càlcul amb vista a optimitzar algun aspecte d'interès. Habitualment, es tracta de minimitzar l'ús de recursos, el temps de processament, o d'arribar a un compromís entre els dos factors.

Per a reduir el temps de processament d'un càlcul concret cal ajustar-se a la seva profunditat mínima. Amb tot, aquest temps ha de ser inferior al període del rellotge que marca les càrregues de valors als biestables del circuit. Això fa que s'hagin de tenir en compte totes les operacions que es poden fer, i les que siguin massa complexes i necessitin un temps relativament més gran que les altres, separar-les en diverses parts o segments. És a dir, **segmentar** l'esquema de càlcul corresponent.

Entre segment i segment s'han d'emmagatzemar les dades intermèdies. Això fa que les implementacions dels esquemes segmentats necessitin registres auxiliars i, per tant, siguin més grans. A canvi, el circuit global pot treballar amb un rellotge més ràpid.

En la figura 36 hi ha una segmentació de l'esquema de càlcul que s'ha pres com a exemple en l'apartat anterior. En el primer segment es calculen els quadrats de totes les dades i, en el segon, es fan les operacions de suma i de resta. Per a fer aquest càlcul calen dos cicles de rellotge i, de manera directa, tres registres addicionals extres: un per a cada quadrat.

Si s'han d'encadenar dos o més càlculs parcials seguits, aquesta estructura té l'avantatge de poder-los començar en períodes de rellotge successius: mentre el segon segment fa les operacions sobre les dades obtingudes en la primera etapa, el primer segment ja fa les operacions amb les dades que venen amb posterioritat a les anteriors. Seguint amb el cas de la figura 36, un cop calculats els quadrats de les dades en un període de rellotge determinat, se'n poden calcular uns altres de nous amb d'altres dades en el període següent i, al mateix temps, fer la suma i la resta dels quadrats que s'havien calculat amb anterioritat.

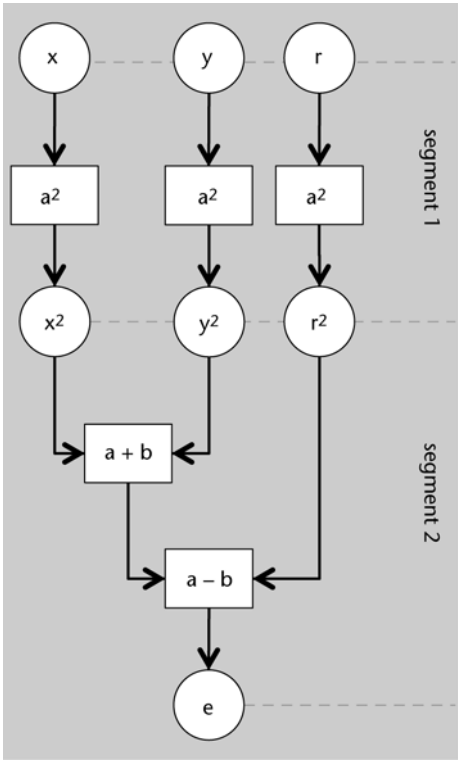
En molts casos, però, el que interessa és reduir al màxim el nombre de registres addicionals i es tornen a fer servir els que contenen les dades d'entrada, sempre que sigui possible (això sol impedir que es puguin prendre noves dades d'entrada a cada cicle de rellotge, tal com s'ha comentat just abans).

Seguint amb l'exemple, per a reduir el nombre de registres, els valors de x i de x^2 s'haurien de desar en un mateix registre i, de manera similar, també s'hauria de fer amb els de y i y^2 i els de r i r^2. Perquè això sigui possible, els formats de representació de les dades han de ser compatibles i, a més, no han d'interferir amb altres càlculs que es puguin fer en el sistema.

Estructures sistòliques

Els esquemes de càlcul segmentats amb emmagatzemaments diferenciats s'anomenen ***estructures sistòliques*** perquè la manera de fer els càlculs s'assembla a la manera de funcionar del cor: a cada sístole s'expulsa un broll de sang. Traduït: a cada pols de rellotge s'obté una dada resultant (d'uns càlculs sobre unes dades que han entrat uns quants cicles abans).

Figura 36. Esquema de càlcul amb segmentació



Com que els retards en proporcionar la sortida respecte del moment en què canvia l'entrada poden ser molt diferents segons el recurs de càlcul de què es tracti, la idea és que els segments introdueixin retards similars. En la segmentació de l'exemple, l'etapa de la suma i de la resta s'han mantingut en un mateix segment, tot suposant que les sumes són més ràpides que els productes. Cal tenir en compte que les multiplicacions inclouen les sumes dels resultats parcials.

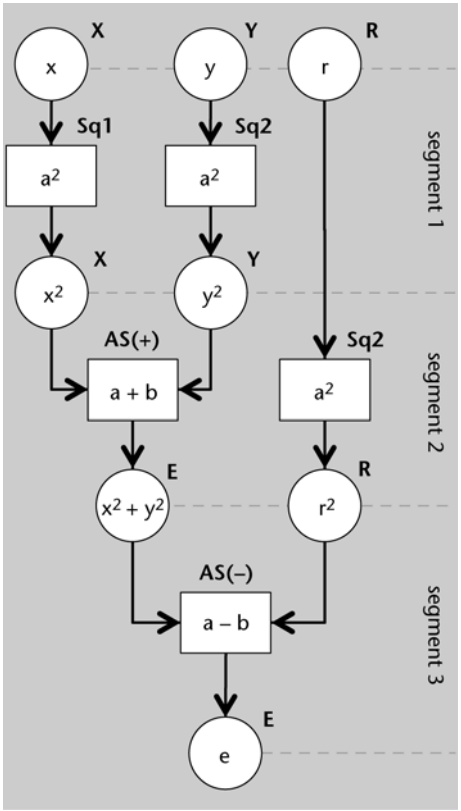
2.3. Esquemes de càlcul amb recursos compartits

La segmentació permet materialitzar esquemes de càlcul que treballen amb rellotges de freqüències més elevades i compatibilitzar els càlculs complexos amb d'altres de més simples en un mateix circuit.

El possible increment en nombre de registres per a emmagatzemar dades intermèdies es pot compensar amb la reducció de recursos de càlcul, ja que sempre n'hi ha que no es fan servir en tots els segments (això és cert si no es fan servir estructures sistòliques).

Així doncs, la idea és fer una segmentació que permeti fer servir un mateix recurs en diverses etapes del càlcul (o segments). Continuant amb l'exemple, el quadrat de *r* i de *y* o *x* es pot fer amb el mateix recurs. Si, a més, s'introdueix un nou segment, la resta i la suma es podrien fer amb un mateix recurs, que fos sumador/restador. L'esquema de càlcul corresponent es mostra en la figura 37.

Figura 37. Esquema de càlcul segmentat amb recursos compartits



L'esquema de càlcul de la figura 37 ja està etiquetat. És a dir, cada node del graf corresponent té un nom al costat que identifica el recurs al qual està associat. Per tant, en un esquema de càlcul, les etiquetes indiquen la correspondència entre nodes i recursos. En aquest cas, els recursos poden ser de memòria (registres) o de càlcul.

Els recursos de càlcul que poden fer diverses funcions incorporen, en l'etiqueta, la identificació de la funció que han de fer. Per exemple, AS(–) indica fer servir el mòdul sumador/restador (AS, de l'anglès *adder/subtracter*) com a restador. Es tracta, doncs, de recursos programables.

A l'extrem, es pot pensar a segmentar un esquema de càlcul de manera que faci servir un únic recurs de càlcul programable. Aquest únic recurs de càlcul hauria de poder fer totes les operacions de l'esquema.

2.4. Materialització d'esquemes de càlcul

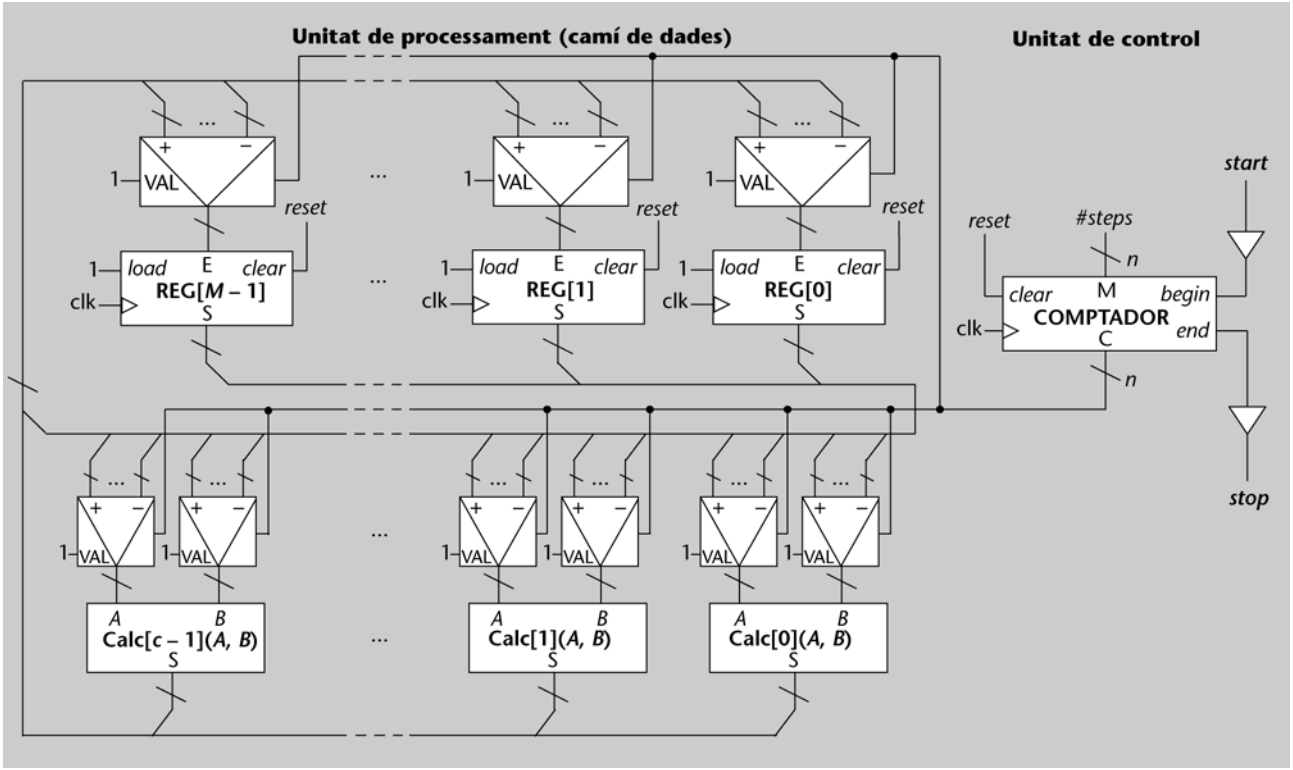
La materialització d'un esquema de càlcul consisteix a transformar-ne el graf etiquetat en el circuit corresponent. Els circuits s'organitzen de manera similar a la que s'ha vist per a les màquines d'estats: es divideixen en una unitat de processament, que és l'esquema de càlcul pròpiament dit, i una de control, que és un comptador senzill.

El model de construcció de la part de processament consisteix en una sèrie de registres l'entrada dels quals és determinada per un multiplexor que la selecciona segons el segment que es tracti. La tria del segment és feta segons un valor de control que coincideix amb el valor de sortida d'un comptador. Si el comptador és autònom, llavors, el processament es repeteix cada cert nombre de cicles. Si és controlat externament, com el de la figura 38, el processament es fa cada cop que es rep un pols d'inici (*start*) i, havent acabat, es posa l'indicador corresponent a 1 (*stop*). Cal tenir en compte que el comptador haurà de fer el compte des de 0 i fins a $M - 1$, en què M és el nombre de passos (#*steps*) o de segments de l'esquema de càlcul.

L'organització dels recursos de càlcul és la mateixa que la dels de memòria (registres): les entrades queden determinades per multiplexors que les seleccionen segons el segment que s'estigui duent a terme de l'esquema de càlcul.

En la figura 38 s'ha suposat que tots els recursos treballen amb dos operands per a simplificar una mica el dibuix. Pel mateix motiu, no s'hi mostren ni les entrades ni les sortides de dades. Habitualment, les entrades de dades es fan en el primer pas (pas número 0 o primer segment) i les sortides s'obtenen d'un subconjunt dels registres en el darrer pas, al mateix temps que es posa *stop* a 1.

Figura 38. Arquitectura general d'un esquema de càlcul

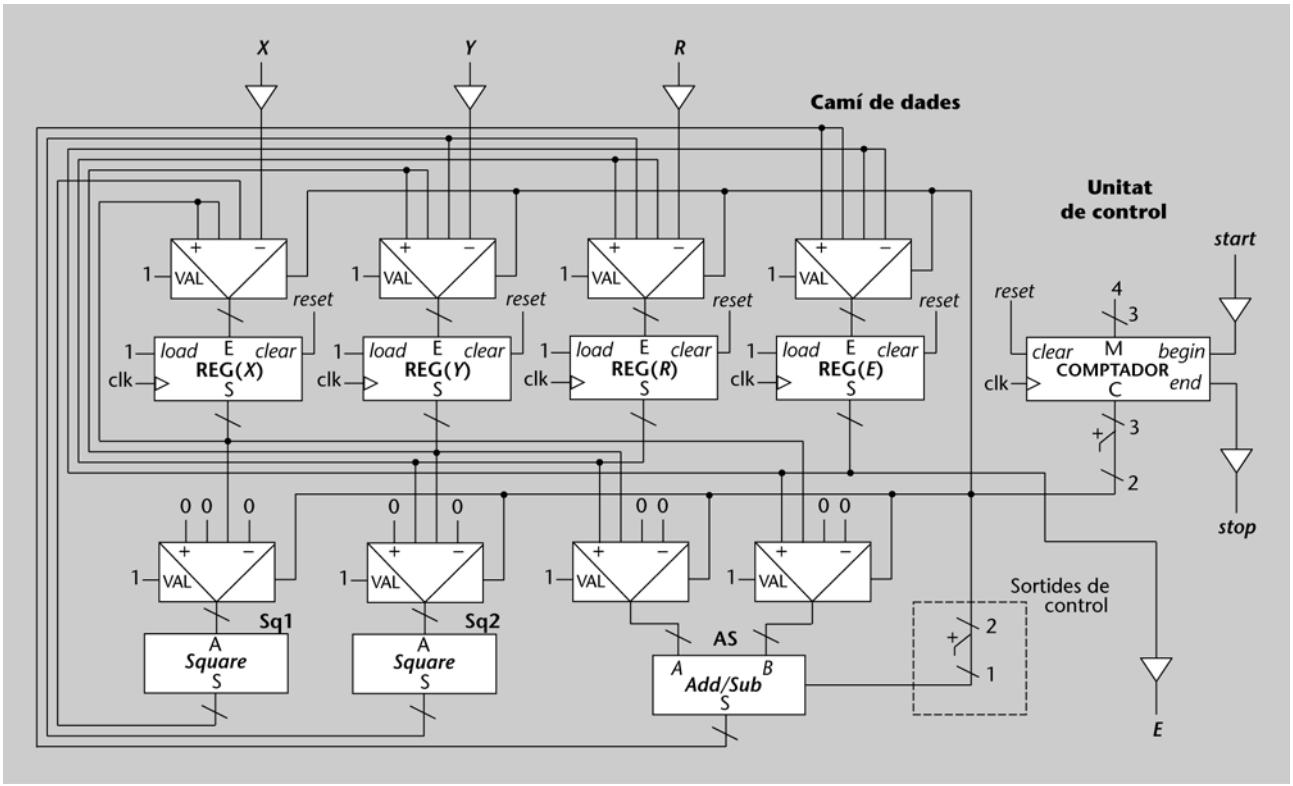


Hi ha dos busos de connexió entre els recursos: el que recull les dades dels recursos de càlcul i les porta als de memòria, i el que serveix per a la comunicació en sentit invers. En la figura només s'ha marcat que cada element pren alguns bits del bus o n'hi deixa, sense especificar quins.

Per a deixar més clara aquesta arquitectura, en la figura 39 hi ha el circuit corresponent al cas d'exemple que s'ha mostrat en la figura 37. Per a construir-lo, primer cal posar multiplexors en les entrades dels registres i dels recursos de càlcul. Han de tenir tantes entrades de dades com segments té l'esquema de càlcul. Les entrades dels multiplexors d'entrada als registres que no es facin servir s'han de connectar a les sortides dels mateixos registres perquè la càrrega no en canviï el contingut. Les dels multiplexors per als recursos de càlcul que no es facin servir es poden deixar a 0. Els busos de distribució de dades s'han de posar a prop: n'hi ha d'haver tants com registres per als multiplexors que seleccionen entrades per als elements de càlcul i tants com recursos de càlcul per als de memòria. Les connexions, llavors, són prou directes tot observant l'esquema de càlcul, segment per segment.

En l'esquema del circuit s'han deixat els multiplexors de quatre entrades per a respectar la forma de construcció que s'ha explicat. Amb tot, amb vista a la seva materialització, es podria simplificar considerablement.

Figura 39. Circuit corresponent a un esquema de càlcul per a $x^2 + y^2 - r^2$



En aquest cas, l'esquema de càlcul té dues sortides: *stop*, que indica que el càlcul ja s'ha acabat, i *E*, que conté el resultat del càlcul complet. De fet, el registre *E* manté el valor del darrer càlcul fins a l'etapa número 2, en què s'hi desa $x^2 + y^2$. En el darrer segment (etapa número 3) serà quan es calculi el valor final ($x^2 + y^2 - r^2$) i s'emmagatzemi a *E*. Per tant, en fer servir els esquemes de càlcul com a submòduls d'algun circuit més gran, cal tenir present que el resultat només serà vàlid quan *stop* = 1.

En aquest tipus d'implementacions, l'estat s'associa directament al camí que segueixen les dades fins a obtenir el resultat d'aquest "recorregut": el valor del comptador selecciona els operands de cada operació i també el registre de destinació. Amb tot, sempre hi pot haver una mica de lògica de control per a calcular altres sortides, com la programació d'alguns recursos de càlcul o la gestió de bancs de registres o memòries. En la figura 39 es pot veure un petit requadre en què es calcula la sortida de control per al recurs programable de suma/resta.

Camí de dades

La denominació de **camí de dades** en la part operacional és molt més rellevant en els esquemes de càlcul, ja que, de fet, cadascun dels seus segments equival al camí que han de seguir les dades per a obtenir un resultat concret.

Activitats

10. Dissenyeu l'esquema de càlcul per a:

$$a \cdot t^2/2 + v \cdot t + s$$

en què a és l'acceleració; v, la velocitat; s, l'espai inicial, i t, el temps.

Es pot suposar que es disposa de recursos de càlcul per a sumar, multiplicar, dividir per 2 i fer el quadrat d'un nombre. Tots els nombres tenen una amplada de 16 bits amb un format en coma fixa i 8 bits per a la part fraccionària.

Els **esquemes de càlcul** són representacions de seqüències d'operacions encaminades a obtenir un determinat resultat. Segons els criteris de disseny amb què es facin, necessiten més o menys recursos. Generalment es poden resoldre de manera que el càlcul es faci al més ràpidament possible, costi el que costi, o amb el mínim cost possible, trigui el que trigui. Els primers esquemes seran tan paral·lels com sigui possible, amb molts recursos de càlcul treballant al mateix temps. Els altres, tan seqüencials com sigui possible, amb el mínim nombre de recursos possible.

2.5. Representació de màquines algorísmiques

Els esquemes de càlcul són útils per a dissenyar màquines de càlcul, però no serveixen per a representar càlculs condicionals o iteratius. Un càlcul condicional només s'efectua si es compleixen unes determinades condicions. Un d'iteratiu necessita repetir una porció de les operacions per a obtenir el resultat.

Si en un càlcul s'introdueixen aquestes opcions es transforma en un algorisme, i la màquina de càlcul corresponent, en una **màquina algorísmica**.

La representació dels algorismes es pot fer de moltes maneres. Gràficament es poden fer servir **diagrames de flux** amb nodes de diversos tipus: un de terminal, un de processament, un de procés predefinit (s'explica què és a continuació) i un de decisió.

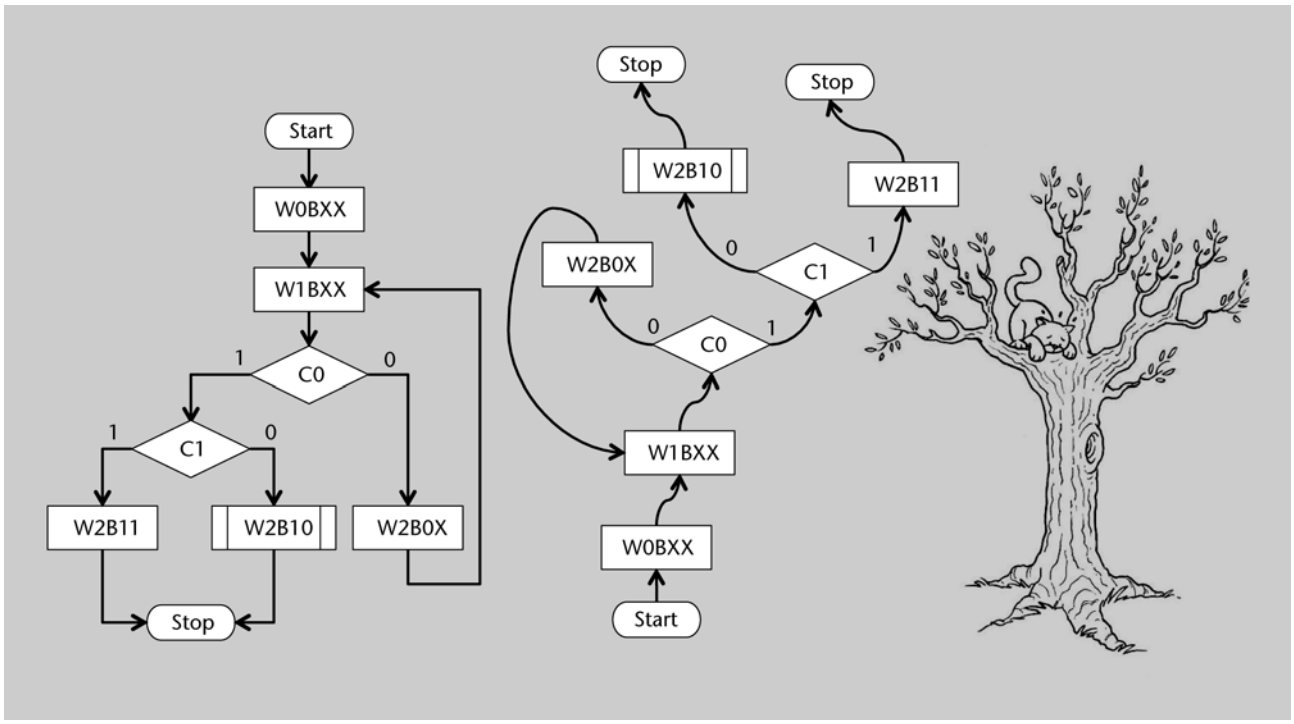
Com es mostra en la figura 40, els nodes terminals tenen la forma de càpsula i serveixen per a marcar el començament i l'acabament de l'algorisme. Els de

processament i procés predefinit s'utilitzen per a encabir-hi les operacions. Tots dos són rectangles, però els darrers tenen una doble línia a cada costat. Les operacions que es posen en els primers s'executen simultàniament. Els segons es fan servir per als esquemes de càlcul. Els nodes de decisió són romboides. S'utilitzen per a introduir branques d'execució noves segons si un bit de condició és 1 (CERT) o 0 (FALS).

De fet, un algorisme es pot veure com una mena d'arbre que té l'arrel en un node terminal (l'inicial) i un tronc del qual neixen diverses branques a cada node de decisió. La longitud de cada branca depèn dels nodes de processament/procés predefinit que hi hagi. Les fulles sempre són nodes terminals de finalització (a diferència dels arbres de debò, algunes branques es poden tornar a unir).

La figura 40 és una il·lustració d'un diagrama de flux, amb nodes de tots els tipus i l'analogia amb els arbres.

Figura 40. Representació d'una màquina algorísmica amb un diagrama de flux, en sentit de lectura a l'esquerra i en forma d'arbre a la dreta



Les representacions dels diagrames de flux són adequades per al disseny d'algorismes que finalment es materialitzin en forma de circuits, ja que es focalitzen en el processament (càlculs en sèrie, condicionals i iteratius) i no pas en el control.

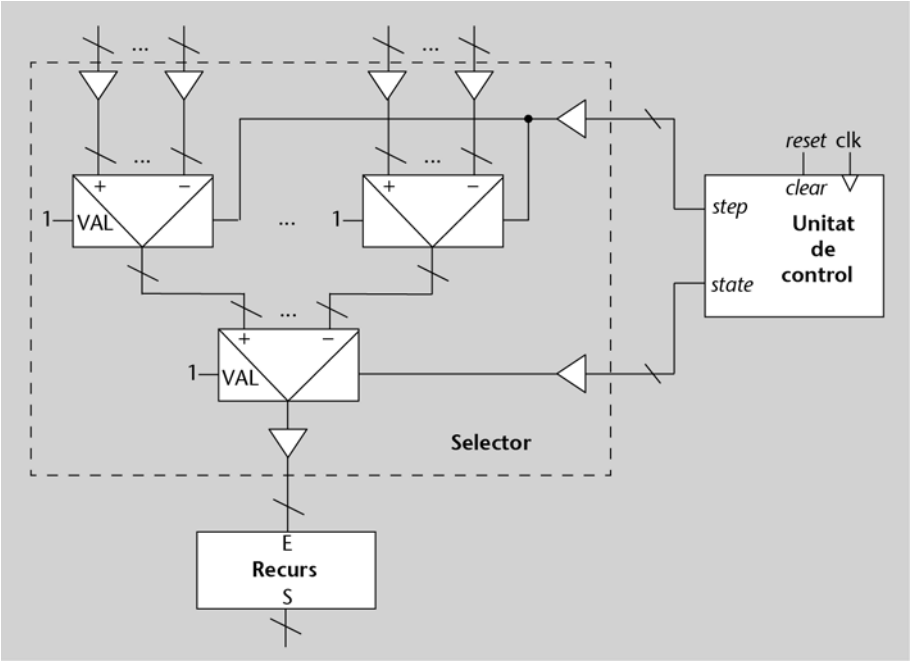
2.6. Materialització de màquines algorísmiques

La implementació dels algorismes segueix un esquema similar al dels esquemes de càlcul, amb un pes més important per a la unitat de processament que per a la unitat de control, que és més senzilla.

L'arquitectura de la unitat de procés és, doncs, igual que la que s'ha presentat per als esquemes de càlcul, excepte en dues coses: que hi ha recursos per a calcular els bits de condició i que els multiplexors que seleccionen les entrades per als recursos són substituïts per mòduls "selectors". Aquests mòduls tenen dos nivells de multiplexors: el primer nivell és el que rep l'entrada del comptador, i el segon el que selecciona l'entrada segons l'estat.

En la figura 41 hi ha un diagrama de blocs de com s'organitza aquest mòdul selector. Com ja passava amb els esquemes de càlcul, aquests mòduls es poden simplificar moltíssim en la majoria d'implementacions. Cal tenir en compte que hi ha molts casos *don't-care* i entrades repetides en els multiplexors.

Figura 41. Mòdul de connexions en l'entrada dels recursos



La unitat de control de les màquines algorísmiques també té un comptador, que serveix per a comptar les etapes o els segments dels esquemes de càlcul corresponents, si n'hi ha. Però també ha de tenir un registre per a emmagatzemar l'estat en què es troba (node de processament o de procés predefinit) i lògica de càlcul de l'estat següent.

De fet, es tracta d'una part de control que s'assembla a la que s'ha vist per a les PSM, en la secció 1.3. Per a fer servir el model de construcció que s'hi presenta caldria transformar el diagrama de flux en una PSM. Aquesta transformació implica agrupar les diverses caixes d'estat en seqüència en un únic estat-programa i convertir els camins que uneixen les caixes d'estats en arcs de sortida de les d'origen amb expressions de condició formades pel producte de les expressions de les caixes de condició, complementades o no, segons si es tracta de la sortida 0 o de la 1 de cada caixa.

També hi ha l'opció d'obtenir la unitat de control directament del diagrama de flux, tal com es veurà a continuació. En aquest sentit, cal tenir present la

similitud entre els diagrames de flux i les ASM, particularment en el fet que els nodes de processament o de procés predefinit són equivalents a les caixes d'estat, i els de decisió a les de decisió.

Per a evitar haver de tenir lògica de càlcul de l'estat següent es pot optar per una codificació de tipus *one-hot bit*. En aquesta codificació, cada estat té associat un biestable que es posa a 1 quan la màquina d'estats es troba, justament, en aquest estat (se suposa que, en un algorisme orientat a fer càlculs, el nombre de biestables que es farà servir serà relativament petit en comparació amb la quantitat de registres de treball del circuit global).

Codificación *one-hot bit*

La codificació *one-hot bit* és la que destina 1 bit a codificar cada símbol del conjunt que codifica. En altres paraules, codifica *n* símbols en *n* bits i a cada símbol correspon un codi en què només el bit associat és a 1, és a dir, el símbol número *i* s'associa al codi amb el bit en posició *i* a 1. Per exemple, si hi ha dos símbols, s_0 i s_1, la codificació de bit únic seria 01 i 10, respectivament.

Amb aquesta mena d'unitats de control, les màquines algorísmiques s'engeguen en rebre un pols a 1 per l'entrada *start* i generen un pols a 1 per la sortida *stop* en el darrer cicle de rellotge en què s'executa l'algorisme associat. En altres paraules, la idea és que reben un 1 per l'entrada *start*, i que aquest 1 es va desplaçant pels biestables de la unitat de control corresponent fins a arribar a la sortida. De fet, cada desplaçament n'és un estat diferent. Cada biestable representa, doncs, un estat o, si es vol, un node de processament o de procés definit.

En la figura 42 hi ha el circuit de la unitat de control del diagrama de flux que s'ha vist en la figura 40. En l'esquema els arcs són marcats amb busos de línia gruixuda, per a facilitar la comparació entre el circuit i el diagrama. A més, els nodes d'estat s'hi identifiquen amb el mateix nom amb què apareixen en el diagrama de flux corresponent. Si aquests nodes tenen més d'una entrada, s'hi ha de posar una porta OR que les uneixi, tal com es pot observar en el circuit amb la porta OR1. La porta OR1 s'ocupa de "deixar passar" l'1 que pot venir o del node de processament inicial (W0BXX) o d'un node posterior (W2B0X).

Els nodes de decisió són elements que fan el mateix que un demultiplexor: transmeten l'entrada a la sortida seleccionada. En la figura n'hi ha dos: DMUX0 i DMUX1. Per exemple, DMUX0 "fa passar" l'1 que ve de W1BXX cap a W2B0X o cap a l'altre node de decisió segons la condició *C*0.

Els nodes que es corresponen amb un procés predefinit (amb un esquema de càlcul, per exemple) activen un comptador. Aquest comptador és compartit per tots els nodes de procés predefinit, ja que només n'hi pot haver un d'actiu alhora. En l'exemple de la figura 40 n'hi ha un, que és W2B10.

Com que la màquina de control ha d'estar a W2B10 mentre el comptador no arribi al màxim, el biestable corresponent es posa a 1 tant per l'arc d'entrada que ve del node de decisió de *C*1 com pel fet que s'hagi activat i el compte no s'hagi acabat (AND1).

En el cas general, l'entrada que activa el comptador (*inc*) és formada per la suma lògica de tots els senyals dels biestables corresponents als nodes de procés predefinit. I l'entrada *A* del comparador és proporcionada per un multiple-

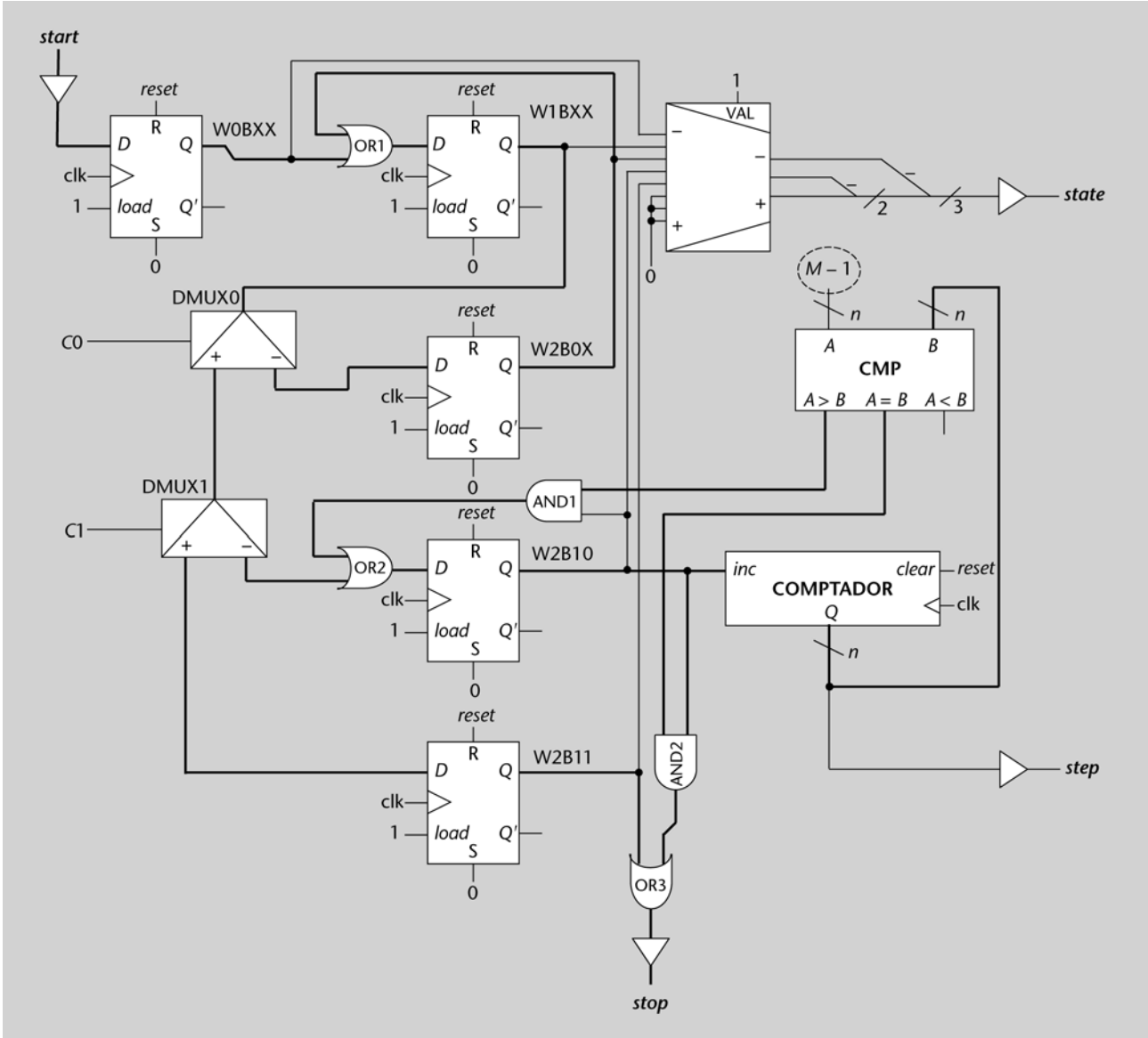
xor que selecciona els diferents valors de $M - 1$ (en què M representa el nombre de cicles a comptar) segons l'estat. En aquest cas, però, no cal ni la porta OR ni el multiplexor perquè només hi ha un procés predefinit, que és W2B10.

L'arc de sortida dels nodes que es corresponen amb processos predefinits és la sortida d'una porta (AND2) que comprova que l'estat és actiu i el compte ja ha acabat.

La sortida *stop* és formada per la suma lògica de tots els arcs que porten al node terminal d'acabament.

És important observar que aquesta unitat té, com a senyals d'entrada, *start* i els bits de condició de la unitat operacional, $C0$ i $C1$. I, com a senyals de sortida, *stop* i els codis d'estat (*state*) i de segment (*step*) per a governar els selectors de la unitat operacional.

Figura 42. Una arquitectura de la unitat de control d'una màquina algorísmica



Si el nombre de nodes d'un diagrama de flux és molt gran, la unitat de control també ho és. Com es veurà més endavant, una possible solució per a la implementació de màquines algorísmiques complexes és l'ús de memòries ROM que emmagatzemin les taules de veritat per a les funcions de càlcul de l'estat següent.

2.7. Cas d'estudi

A l'hora de dissenyar una màquina algorísmica cal partir d'un diagrama de flux amb el mínim nombre de nodes possible: com menys nodes, més petita pot ser la unitat de processament i més simple pot ser la unitat de control.

El cas d'estudi que es presentarà consisteix a construir un multiplicador seqüencial. La idea és reproduir el mateix procediment que s'utilitza amb les multiplicacions a mà: s'obté un producte del multiplicand per cada dígit del multiplicador i se sumen els resultats parcials de manera decalada, segons la posició del dígit del multiplicador. En binari, les coses són més senzilles, ja que el producte del multiplicand pel bit del multiplicador que toqui es converteix en 0 o en el mateix multiplicand. Així, una multiplicació de nombres binaris es converteix en una suma de multiplicands decalats segons la posició de cada 1 del multiplicador.

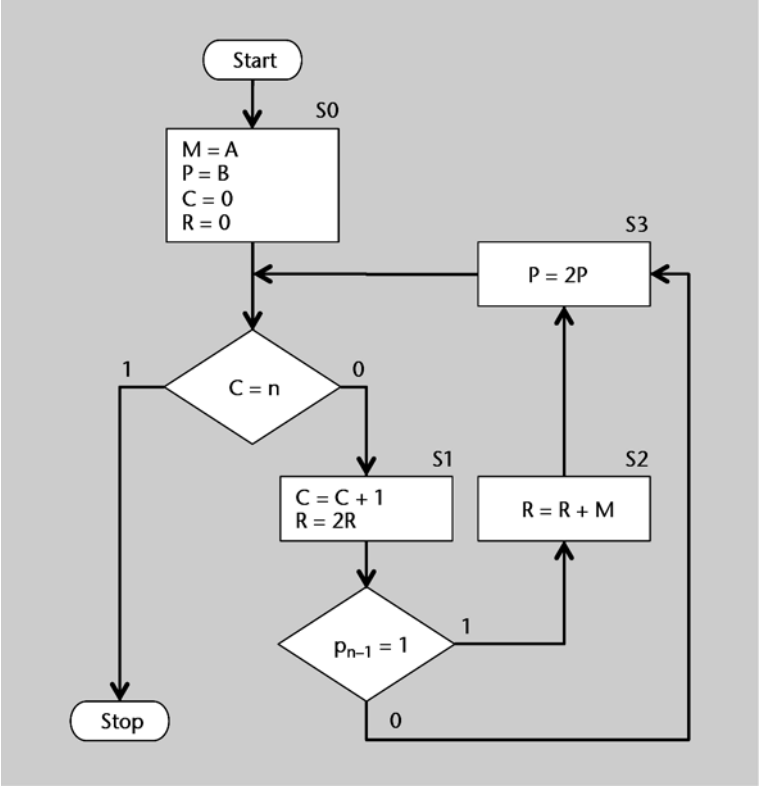
Per a veure-ho més clar, tot seguit es calcula el producte de 1011 (multiplicand) per 1010 (multiplicador):

$$
\begin{array}{cccccccc}
 & & & & 1 & 0 & 1 & 1 \\
 & & & \mathrm{x} & 1 & 0 & 1 & 0 \\
\hline
 & & & & 0 & 0 & 0 & 0 \\
 & & & 1 & 0 & 1 & 1 & \\
 & & 0 & 0 & 0 & 0 & & \\
 & 1 & 0 & 1 & 1 & & & \\
\hline
0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline\hline
\end{array}
$$

A fi de materialitzar un circuit que faci la multiplicació amb aquest procediment cal, primer, obtenir un diagrama de flux de les operacions i de les decisions que s'hi prenen.

En la figura 43 hi ha l'algorisme corresponent: el multiplicand és A, el multiplicador, B i el resultat és R. Internament, A i B s'emmagatzemen en les variables M i P, respectivament. L'indicador de la posició del dígit de B amb què toca fer un determinat producte s'obté d'un comptador, C. El comptador serveix, a més, de control del nombre d'iteracions de l'algorisme, que és el nombre de bits dels operands A i B (n).

Figura 43. Diagrama de flux d'un multiplicador binari



En els diagrames de flux per a les màquines algorísmiques, els diferents càlculs amb variables no s'expressen de la manera:

$$V^{+} = \text{expressió}$$

sinó que es fa de la manera:

$$V = \text{expressió}$$

perquè són representacions d'algorismes que treballen, majoritàriament, amb dades emmagatzemades en variables. Amb tot, cal tenir present que això es fa per a simplificar la representació: les variables actualitzen el valor en acabat de l'estat on es calculen. Se suposarà que no hi ha senyals de sortida que no siguin variables.

Val a dir que el resultat *R* haurà de tenir una amplada de 2*n* bits i que el comptador en té prou amb el nombre enter més proper per la dreta a $\log_2 n$, que s'identificarà en el circuit amb *m*, és a dir que el comptador serà de *m* bits, amb *m* essent prou gran per a representar qualsevol nombre més petit que *n*.

L'algorisme de multiplicació ha de començar (*S*0) carregant el multiplicand a *M* i el multiplicador a *P*, com també inicialitzant-ne les variables de treball. En aquest cas, *R* també serà la que contindrà el resultat de sortida.

Com que ha de fer tants productes com bits tingui el multiplicador *P*, hi ha un node de decisió que serveix per a controlar el nombre d'iteracions: quan *C* sigui *n* ja s'hauran fet tots els productes parcials i a *R* hi haurà el resultat.

A cada iteració, s'incrementa el comptador C i es desplaça el resultat previ cap a l'esquerra ($S1$). Això últim es justifica perquè el resultat final s'obté fent les sumes parcials dels productes del multiplicand M pels bits del multiplicador d'esquerra a dreta. Així doncs, abans de sumar a R un nou producte, cal multiplicar-lo per 2, perquè el resultat parcial que conté és de la suma parcial prèvia, que correspon als productes de M pels bits més significatius de P.

El procés a $S2$ consisteix, simplement, a fer la suma del producte d'1 bit a 1 de P per M amb el resultat parcial previ a R. Amb independència d'aquest procés, a $S3$ es fa un desplaçament a l'esquerra de P per a obtenir el multiplicador següent.

Els desplaçaments de P i de R estalvien l'ús de multiplexors o d'una altra mena de selectors, ja que els bits amb què s'ha de treballar sempre estan en posicions fixes.

Per a veure-ho més clar, tot seguit hi ha la multiplicació de l'exemple anterior, representada d'acord amb les operacions de l'algorisme que s'ha presentat (els desplaçaments de P no s'hi mostren, però es poden deduir del subíndex del bit que es fa servir per al producte, que aquí sí que canvia).

				1	0	1	1	M
			x	1	0	1	0	P
0	0	0	0	0	0	0	0	$R = 2 \cdot R$
			+	1	0	1	1	$M \cdot p_{n-1}$
0	0	0	0	1	0	1	1	$R = R + M \cdot p_{n-1}$
0	0	0	1	0	1	1	0	$R = 2 \cdot R$
			+	0	0	0	0	$M \cdot p_{n-2}$
0	0	0	1	0	1	1	0	$R = R + M \cdot p_{n-2}$
0	0	1	0	1	1	0	0	$R = 2 \cdot R$
			+	1	0	1	1	$M \cdot p_{n-3}$
0	0	1	1	0	1	1	1	$R = R + M \cdot p_{n-3}$
0	1	1	0	1	1	1	0	$R = 2 \cdot R$
			+	0	0	0	0	$M \cdot p_{n-4}$
0	1	1	0	1	1	1	0	$R = R + M \cdot p_{n-4}$
0	1	1	0	1	1	1	0	

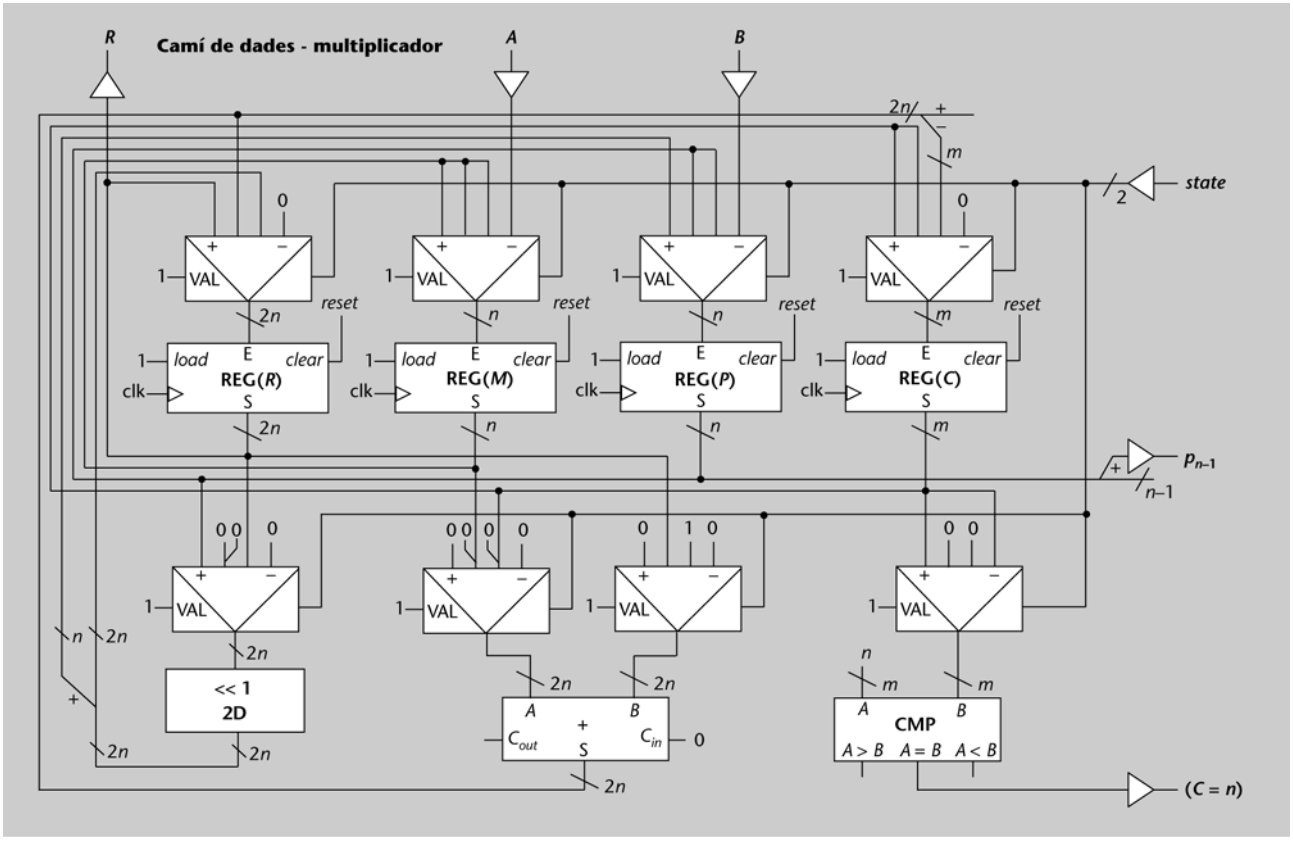
La unitat de processament seguirà l'arquitectura que s'ha presentat, però sense necessitat de selectors de dos nivells, ja que no hi ha processos predefinits: tot són processaments que es poden fer en paral·lel. Això també simplificarà la unitat de control.

Per a construir la unitat de processament cal primer tenir clars els recursos de memòria i de càlcul que ha de tenir. Quant als primers, n'hi ha prou amb un

registre per a cada variable de l'algorisme: M i P de n bits, R de $2n$ bits i C de m bits. Quant als segons, n'hi ha d'haver un per operació diferent, com a mínim. En aquest cas, tant les sumes com els decaladors poden ser compartits perquè es fan servir en estats diferents, cosa que és positiva perquè es redueix el nombre de recursos de càlcul. En aquest cas, només cal un sumador per a calcular $C + 1$ i $R + M$ perquè són sumes que es fan a estats diferents ($S1$ i $S2$, respectivament) i només cal un decalador que es pot aprofitar per a fer el càlcul de $2R$ a $S1$ i $2P$ a $S3$.

El fet que el decalador i el sumador siguin compartits implica que han de treballar amb dades de tants bits com el màxim de bits dels operands que puguin rebre. En aquest cas, com que R és de $2n$ bits, cal ajustar la resta d'operands a aquesta amplada: en el cas del decalador, s'afegeixen n bits a 0 a la dreta de P, en l'entrada més significativa, per a arribar a $2n$ bits; i en el cas del sumador, els operands C i M, a les entrades 1 i 2 del multiplexor de l'esquerra, es passen a $2n$ bits tot afegint-los 0 per l'esquerra, per a no canviar-ne el valor (en la figura no s'han posat les amplades dels busos per a mantenir-ne la llegibilitat).

Figura 44. Unitat de processament d'un multiplicador en sèrie



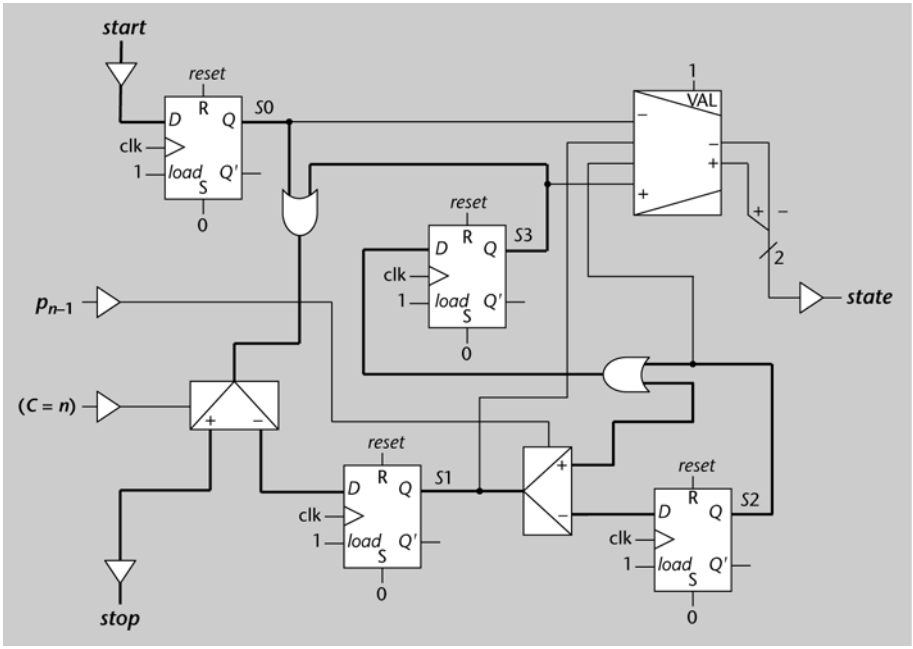
La unitat de processament que es mostra en la figura 44 segueix fidelment el model de construcció, però es pot simplificar pel que fa a l'ús dels multiplexors.

La unitat de control segueix el model de construcció presentat en l'apartat anterior, amb una codificació dels estats de tipus *one-hot bit* que permet el trasllat

directe de l'estructura del diagrama de flux a la del circuit corresponent. Amb tot, té un aspecte diferent del vist en la figura 42 perquè no hi ha cap comptador. Les entrades al codificador de l'estat han de seguir el mateix ordre que les entrades dels multiplexors dels selectors de la unitat de processament.

En la figura 45 hi ha l'esquema del circuit de control corresponent al multiplicador. Les línies de connexió que es corresponen amb els arcs del diagrama de flux corresponent es destaquen en negreta.

Figura 45. Unitat de control d'un multiplicador en sèrie



En aquest cas, atès que el nombre d'estats (quatre) i d'entrades (dues) és petit, també es pot construir una unitat de control a partir de la taula de transicions. Per a fer-ho, és convenient veure el diagrama de flux com un graf d'estats en què els nodes de processament són estats i els nodes de decisió, condicions de les transicions, de manera similar a com es tracta en les ASM.

Activitat

11. Refeu els esquemes dels circuits de la unitat de processament i de la de control, si convé, per reduir al màxim el nombre de multiplexors i la quantitat d'entrades de cadascun en la unitat de processament.

3. Arquitectura bàsica d'un computador

Un computador és una màquina capaç de processar informació. Amb aquesta definició, certament som en un món ple de computadors, alguns dels quals veiem com a tals, com els portàtils i els personals. Però la gran majoria no els percebem com a ordinadors perquè es troben encastats (i amagats) en altres objectes: dispositius mòbils, sistemes de cinema a casa, electrodomèstics, cotxes, avions, etc. Segons la variabilitat dels algorismes que poden executar per a processar la informació es pot anar des dels més senzills, constituïts per màquines algorísmiques especialitzades, fins als més complexos.

En aquest apartat es tractarà de com estan construïts els computadors i com funcionen, començant el recorregut per la transformació de les màquines algorísmiques en computadors i acabant per les arquitectures dels computadors més complets.

En primer lloc es veuran les màquines algorísmiques que són capaces d'interpretar seqüències de codis d'accions i operacions, és a dir, que poden executar programes diferents canviant exclusivament el contingut de la memòria en què s'emmagatzemen. De fet, d'aquest tipus de màquines se'n diu **processador** perquè processa dades d'acord amb un programa determinat.

A mesura que s'augmenta el repertori de coses que poden fer els programes, és a dir, el conjunt d'instruccions diferents que inclouen, les màquines algorísmiques que les poden entendre també es fan més complexes. En el segon subapartat es veurà com es pot construir un processador capaç d'executar programes relativament complexos i que s'organitza en dos grans blocs: el de memòria i el de processament. S'explicarà com s'interpreten les instruccions i la relació entre aquests dos blocs.

El tercer subapartat es dedica a explicar arquitectures de processadors (**microarquitectures**) més complexes que la del processador senzill i com s'aconsegueix que puguin executar més instruccions per unitat de temps. També s'explicaran els diversos tipus de processadors que hi pot haver segons les característiques que tenen.

Com que els processadors s'ocupen de processar informació però no d'obtenir-ne ni de proporcionar-ne, s'han d'acompanyar d'altres components que duguin a terme aquestes funcions, és a dir, que s'ocupin de l'entrada i sortida de la informació. El conjunt se l'anomena **computador**. La quarta i última secció explica com estan construïts els computadors tant en general com els que s'orienten a aplicacions específiques.

3.1. Màquines algorísmiques generalitzables

Les màquines algorísmiques es construeixen per a executar un únic algorisme de la manera més eficient possible. És a dir, duen a terme la funcionalitat indicada en l'algorisme tot minimitzant-ne el cost. En principi, el cost d'execució es determina segons factors com el temps, el consum d'energia i la grandària dels circuits.

L'evolució de la microelectrònica ha permès que, en un mateix espai, cada cop s'hi puguin posar circuits més complexos. Per tant, cada cop es poden construir màquines amb una funcionalitat més gran. Però també cal que els sistemes resultants tinguin un cost raonable respecte del rendiment que se n'obté. La batalla per l'eficiència està, doncs, en el temps i en el consum d'energia: es volen obtenir resultats cada cop més ràpidament i consumint poca energia.

Funcionalitat

En aquest context, la **funcionalitat** fa referència al conjunt de funcions que pot dur a terme un sistema. La funcionalitat d'un algorisme és formada per totes les funcions que es poden diferenciar de manera externa. Per exemple, la funcionalitat d'un algorisme de control d'un giny (*gadget*) d'àudio portàtil inclou funcions per a fer sonar un tema, deixar-lo en pausa, passar al següent, etc.

Per exemple, es vol que els "telèfons mòbils intel·ligents" permetin veure pel·lícules d'alta qualitat amb bateries de llarga durada i poc voluminoses que facilitin la portabilitat dels dispositius.

Una altra part del cost té a veure amb el desenvolupament i manteniment posterior dels productes governats per màquines algorísmiques: com més complexa és la funcionalitat d'un sistema, més difícil és implementar la màquina corresponent. A més, qualsevol canvi en la funcionalitat (és a dir, en l'algorisme) implica construir una màquina algorísmica nova.

Amb l'estat actual de la tecnologia, el que pesa més a l'hora de materialitzar un producte és la funcionalitat (perquè n'incrementa el valor afegit) i la facilitat de desenvolupament (perquè redueix el temps de posada al mercat) i manteniment posterior (perquè en facilita l'actualització). De fet, la tecnologia permet executar de manera eficient gran part de la funcionalitat genèrica dels algorismes. Només unes poques funcions són realment crítiques i cal durles a terme amb algorismes específics.

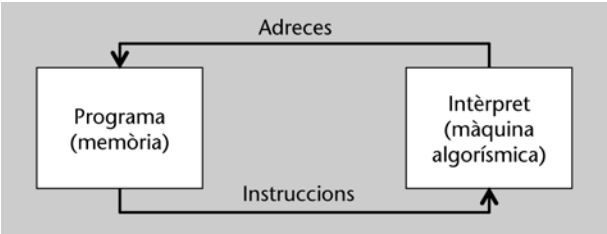
Funció crítica

Es diu que **una funció és crítica** quan té un pes molt elevat en el cost d'execució de l'algorisme que la inclou. Per exemple, perquè les altres funcions en depenen o perquè té molta influència en els paràmetres que determinen el cost o l'eficiència finals de l'algorisme global.

Així doncs, és convenient fer servir màquines més genèriques, que puguin executar conjunts de funcions diferents segons l'aplicació a què es destinin o la pròpia evolució del producte en el qual estiguin encastades. Per a fer-ho, els algorismes haurien d'estar emmagatzemats en mòduls de memòria, de manera que les màquines durien a terme els algorismes que, en un moment determinat, hi estiguessin guardats.

Una **màquina algorísmica generalitzable** seria la capaç d'interpretar les instruccions del programa (els passos de l'algorisme) que hi hagués en la memòria corresponent.

Figura 46. Esquema de la relació entre la memòria de programa i la màquina algorísmica d'interpretació



La màquina algorísmica que interpretés les instruccions emmagatzemades en la memòria corresponent seria, de fet, la unitat encarregada de fer el processament del programa en memòria. Per a poder construir aquesta màquina és necessari determinar quines instruccions ha d'interpretar i com es codifiquen en binari. A tall d'exemple, en l'apartat següent es descriu una màquina d'aquest tipus.

3.1.1. Exemple de màquina algorísmica general

Les màquines algorísmiques que interpreten programes emmagatzemats en la memòria són genèriques perquè són capaces d'executar qualsevol programa, encara que no ho poden fer amb l'eficiència de les màquines específiques, és clar. Es converteixen, doncs, en màquines de processament de programes o **processadors**. En aquest apartat en veurem un de molt petit: el Femtoproc.

La idea és detallar com es pot construir un d'aquests intèrprets aprofitant el que s'ha vist en les seccions anteriors i, amb això, il·lustrar el funcionament intern dels processadors.

El Femtoproc serà una petita màquina algorísmica que interpreti un repertori mínim d'instruccions: una per a sumar (`ADD`), un parell per a fer operacions lògiques bit a bit (`NOT` i `AND`) i, finalment, una de salt condicional (`JZ`), que efectuarà un salt en la seqüència d'instruccions a executar si la darrera operació ha donat com a resultat un 0.

A tall de curiositat, aquest repertori permet fer qualsevol programa, per sofisticat que sigui (el nombre d'instruccions, però, podria ser enorme, això sí). Cal tenir en compte que la suma també permet fer restes, si es treballa amb nombres en complement a 2, que amb la NOT i la AND es pot construir qualsevol funció lògica i que el salt condicional permet alterar la seqüència d'instruccions per a prendre decisions i es pot utilitzar com a salt incondicional (forçant la condició) per a fer iteracions.

Les instruccions es codifiquen amb 8 bits (1 byte) segons el format següent:

Instrucció	Bits							
	7	6	5	4	3	2	1	0
ADD	0	0	$operand_1$			$operand_0$		
AND	0	1	$operand_1$			$operand_0$		
NOT	1	0	$operand_1$			$operand_0$		
JZ	1	1	adreça de salt					

Els operands de les operacions ADD, AND i NOT s'obtenen d'un banc de registres que, atesa la codificació (s'utilitzen 3 bits per a identificar els operands), ha de tenir vuit registres: *R*7, *R*6, *R*5, *R*4, *R*3, *R*2, *R*1 i *R*0. Tots els registres són de 8 bits.

Per facilitar la programació, *R*0 i *R*1 són constants, és a dir, són registres en què no es pot escriure res (de fet, són falsos registres). *R*1 té el valor 01h (unitat), de manera que sigui possible construir qualsevol altre nombre i, especialment, que es puguin fer increments i decrements. *R*0, en canvi, té el valor 80h = $10000000_{(2}$ per a poder esbrinar el signe dels nombres enters.

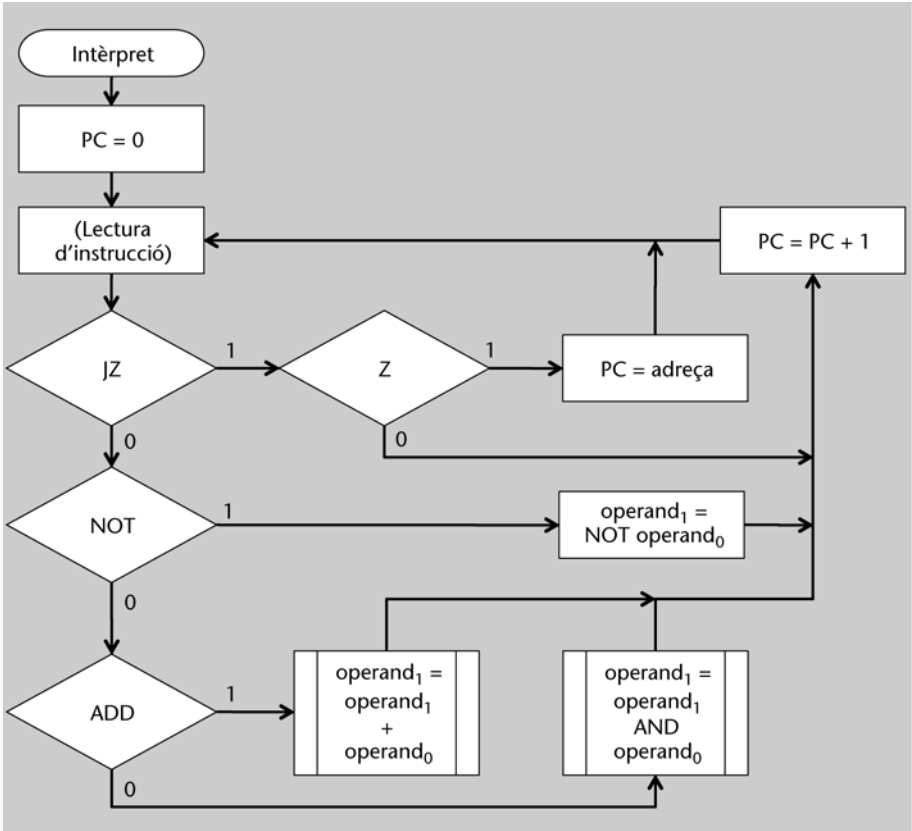
El resultat de les operacions s'emmagatzemarà en l'$operand_1$. Si és *R*0 o *R*1, el resultat es perdrà, però el bit que identifica si l'última operació ha estat 0 o no (*Z*) sí que s'actualitzarà. *Z* identifica un biestable que emmagatzema aquesta condició.

Com que la instrucció `JZ` pot expressar adreces de 6 bits, els programes s'emmagatzemaran en una ROM de 2^6 = 64 posicions, de 8 bits cadascuna.

Amb aquesta informació, ja és possible fer l'algorisme d'interpretació d'instruccions i materialitzar-lo amb la màquina algorísmica corresponent.

En aquest algorisme es fa servir una variable que fa de comptador de programa o *program counter* (PC), en anglès. Es tracta d'una variable que conté la posició de memòria de la instrucció a executar. Com es veu en el diagrama de flux de la figura 47, l'algorisme consisteix en un bucle infinit que comença per la lectura de la instrucció número 1, que es troba en l'adreça 0.

Figura 47. Diagrama de flux de la màquina d'interpretació de {`ADD`, `AND`, `NOT`, `JZ`}



El diagrama de flux anterior es pot detallar una mica més, sabent la codificació de les instruccions que interpreta el Femtoproc. En la taula següent es pot veure l'equivalència entre les descripcions dels nodes del diagrama de flux i les operacions que hi estan vinculades.

Procés o condició	Operacions	Efecte
(Lectura d'instrucció)	$Q = M[PC]$	Obtenció del codi de la instrucció que hi ha en la posició *PC* de memòria.
Salt?	$q_7 \cdot q_6$	Càlcul del bit que indica si la instrucció és JZ.
Zero?	z'	Indicació que la darrera operació ha tingut, com a resultat, el valor 0.
PC = adreça	$PC = Q_{5-0}$	Assignació del valor de l'adreça de salt a *PC*.
$PC = PC + 1$	$PC = PC + 1$	Assignació del valor de $PC + 1$ a *PC*.
NOT?	$q_7 \cdot q'_6$	Càlcul del bit que indica si la instrucció és NOT.
$operand_1$ = NOT $operand_0$	$BR[Q_{5-3}]$ = NOT $BR[Q_{2-0}]$	Complement dels bits del registre en posició Q_{2-0} del banc de registres (BR) i assignació al registre en posició Q_{5-3}.
Suma?	$q'_7 \cdot q'_6$	Determinació que la instrucció sigui ADD.
$operand_1 = operand_1 + operand_0$	0: $B = BR[Q_{2-0}]$; 1: $BR[Q_{5-3}] = BR[Q_{5-3}] + B$;	Execució de l'esquema de càlcul que obté la suma dels registres Q_{2-0} i Q_{5-3} del BR i assignació al registre en posició Q_{5-3}.
$operand_1 = operand_1$ AND $operand_0$	0: $B = BR[Q_{2-0}]$; 1: $BR[Q_{5-3}] = BR[Q_{5-3}]$ AND B;	Execució de l'esquema de càlcul que obté l'AND entre els bits dels registres Q_{2-0} i Q_{5-3} del BR i assignació al registre en posició Q_{5-3}.

Com es pot veure en la taula anterior, el node de processament corresponent a la lectura de la instrucció no és necessari, ja que el codi d'operació de la instrucció, *Q*, s'obté directament de la sortida de dades de la memòria del programa. Com es veurà, en màquines més complexes, la codificació de les instruccions fa que sigui necessari emmagatzemar-ne el codi en una variable per a descodificar-les de manera progressiva.

També hi ha dos esquemes de càlcul, un per a la suma i l'altre per al producte lògic (AND), que forçosament han d'estar segmentats si es fa servir un banc de registres amb una entrada i una única sortida de dades, ja que és necessari disposar d'un registre auxiliar (*B*) que emmagatzemi temporalment un dels operands.

La materialització de la màquina algorísmica corresponent es mostra a continuació només a tall d'exemple. Com es veurà tot seguit, el disseny d'aquestes màquines és més un art que una ciència, ja que cal transformar els diagrames de flux corresponents fins a arribar als de partida per a la implementació.

Implementació del Femtoproc

Si es partís del diagrama de flux anterior s'obtindria una màquina que funcionaria correctament, però seria ineficient. En concret, si s'assignés un estat per node de processament o node de procés predefinit (els esquemes de càlcul corresponents necessiten, forçosament, dos estats de compte), la màquina tindria vuit estats: PC = 0, PC = adreça, PC = PC + 1, NOT, ADD (x2) i AND (x2).

Simplificant estats, l'estat inicial PC = 0 només ha de posar el registre *PC* a 0, cosa que ja es fa amb la inicialització inicial, per tant, es pot suprimir. Ja són set.

Mirant les operacions dels esquemes de càlcul es pot veure que la primera operació és igual. Per tant, es poden separar en dos nodes de processament i fer que el primer sigui compartit. Ja només en queden sis.

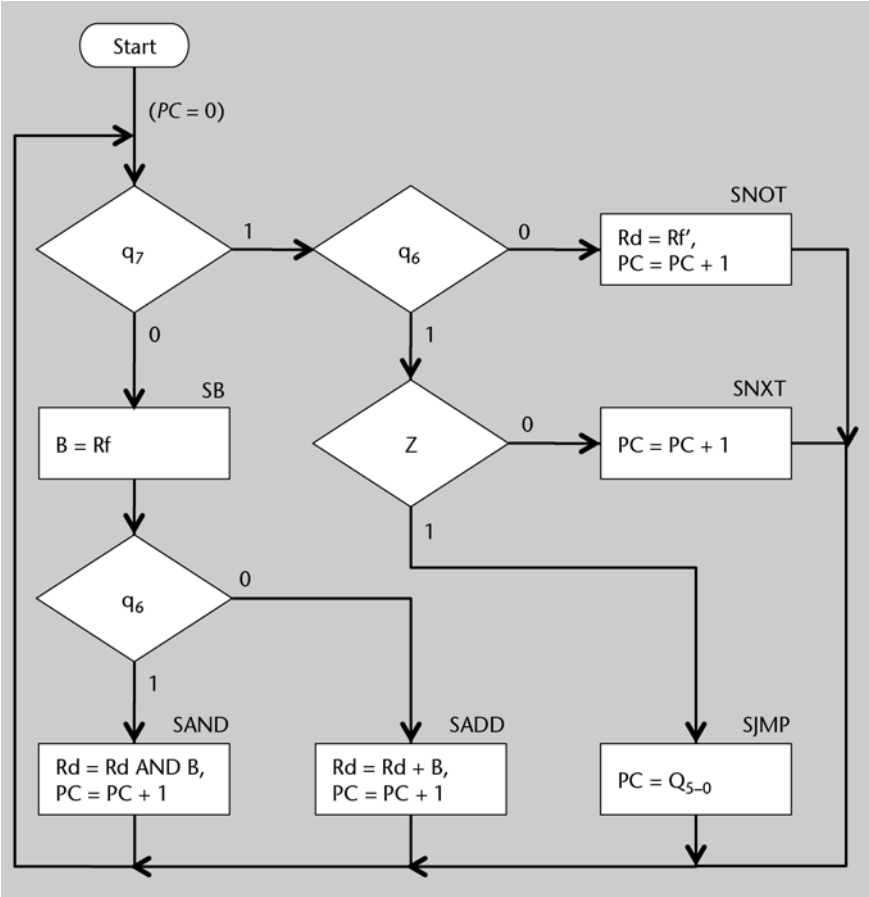
En la figura 48 es pot comprovar aquesta simplificació: s'ha reduït el nombre de nodes i minimitzat els esquemes de càlcul. En la figura es fa servir Rd (de **registre de destinació**) per a representar el registre en posició Q_{5-3} del banc de registres (BR), que substitueix el símbol $operand_1$ i Rf (de **registre font**) per BR[Q_{2-0}]. Els nodes de processament porten en la part dreta superior el nom que n'identifica l'estat corresponent.

Els nodes de decisió s'han alterat per a prendre's d'acord amb els bits del codi d'instrucció. Així, per a determinar si és JZ l'expressió és $q_7 \cdot q_6$, i per a NOT és $q_7 \cdot q'_6$, cosa que es pot transformar en esbrinar primer si $q_7 = 1$ i, llavors, segons q_6 se sap si és JZ o NOT. Una cosa similar es fa per discernir entre ADD i AND, que tenen un punt en comú: que el valor del primer operand s'ha d'emmagatzemar en el registre *B*. El segon pas sí que és diferent.

El camí de dades necessita els recursos que s'enumeren a continuació.

- **De memòria**: un per al comptador de programa (*PC*), un biestable per a emmagatzemar la indicació de si la darrera operació ha estat zero o no (*Z*), un registre auxiliar per a un dels operands (*B*) i un banc de registres (BR) amb vuit registres de 8 bits, dels quals, els dos primers amb valors fixats.
- **De càlcul**: dos sumadors (un per a incrementar el *PC* i l'altre per a la suma de dades), un producte lògic de busos, un complementador i la memòria ROM que conté el programa (es podria dir que és el circuit que relaciona el valor del *PC* amb el de la instrucció a executar, $Q = M[PC]$).
- **De connexió**: busos, un multiplexor de selecció del proper valor per al *PC*, que pot ser $PC + 1$ o Q_{5-0}, un altre del resultat a carregar al BR i un altre per a triar quin registre es vol llegir, si Rd o Rf.

Figura 48. Diagrama de flux del Femtoproc adaptat per a la materialització



La codificació dels estats serà de tipus *one-hot bit*, de manera que la unitat de control es pugui implementar directament a partir del diagrama de flux, tal com s'ha vist.

La taula que relaciona els estats amb les operacions del camí de dades és la següent. Tots els senyals denominats amb *ld_* es connecten a les entrades de càrrega de contingut dels registres associats. Els controls dels multiplexors són senyals que es denominen amb *selOp* previ al nom del recurs de memòria al qual proporcionen l'entrada de dades. En el

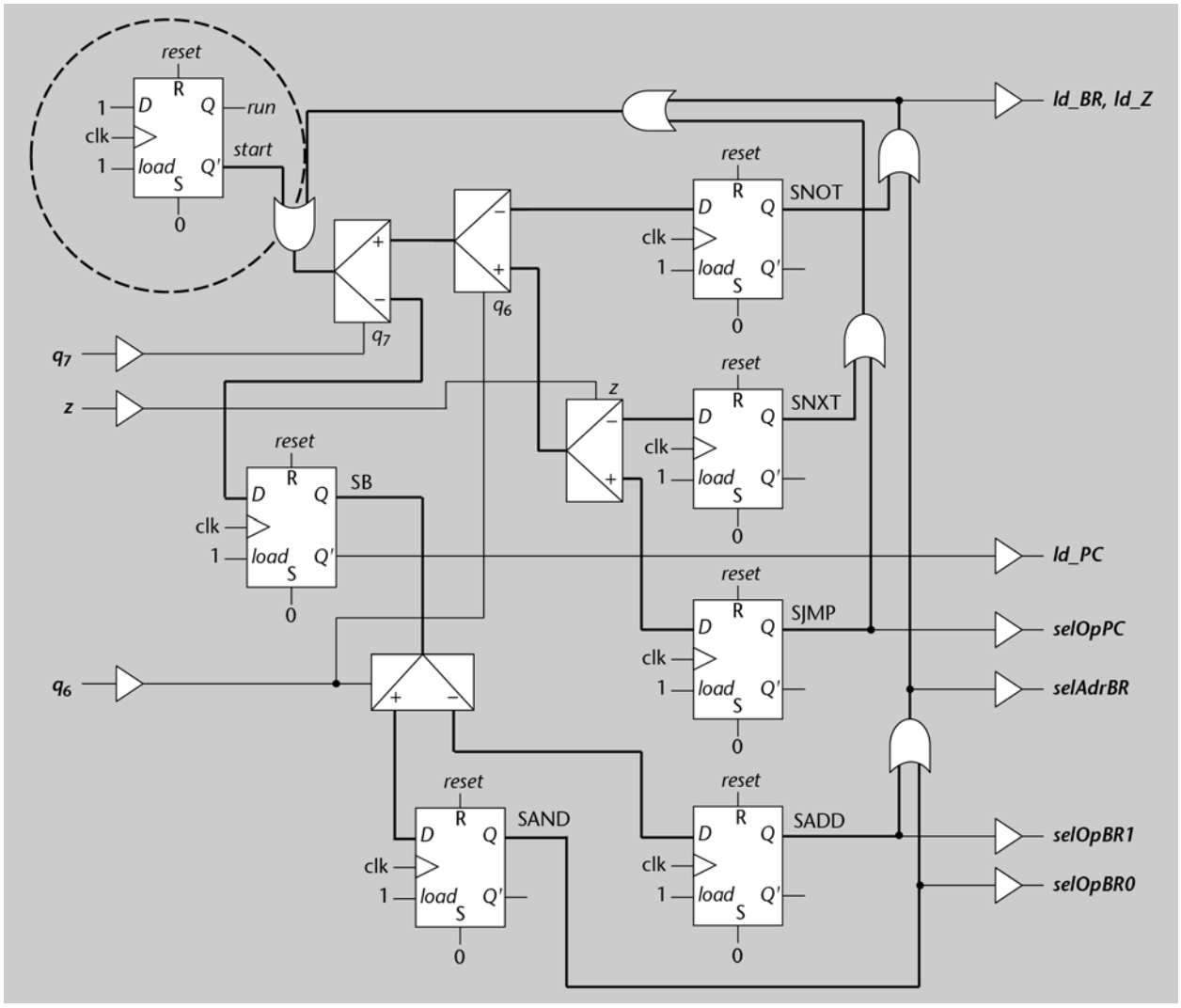
cas del banc de registres, *selOpBR* fa referència a la selecció del resultat a escriure, *ld_BR*, al senyal que n'activa l'escriptura, i *selAdrBR* a l'adreça del registre el contingut del qual serà presentat a *Dout* (0 per a Rf i 1 per a Rd). El senyal *selOpPC* és 1 per a seleccionar l'adreça de salt, que és Q_{5-0}.

Estat	*ld_PC*	*selOpPC*	*ld_B*	*selOpBR*	*ld_BR*	*selAdrBR*	*ld_Z*
SNXT	1	0	x	xx	0	x	0
SJMP	1	1	x	xx	0	x	0
SB	0	x	1	xx	0	0	x
SNOT	1	0	x	00	1	0	1
SAND	1	0	x	01	1	1	1
SADD	1	0	x	10	1	1	1

Amb vista a la implementació de la unitat de control, la majoria de sortides es poden obtenir directament d'un únic bit del codi d'estat, llevat dels senyals *ld_BR*, *selAdrBR* i *ld_Z*, que s'han de construir amb sumes lògiques d'aquests bits.

En la figura 49 hi ha el circuit de control del Femtoproc. És important tenir present que, a diferència de les unitats de control vistes en l'apartat 2.6, no hi ha senyal d'arrencada (*start*) ni senyal de sortida (*stop*): aquest circuit funciona de manera autònoma executant un bucle infinit (és com s'anomena, tot i que sempre es pot reiniciar amb *reset* o tallar l'alimentació de corrent). El circuit que hi ha encerclat proporciona el primer 1, just després d'un *reset*: El biestable es posa a 0 i, conseqüentment, la sortida negada es posa a 1. En el flanc següent de rellotge, carregarà un 1 i *start* passarà a ser 0.

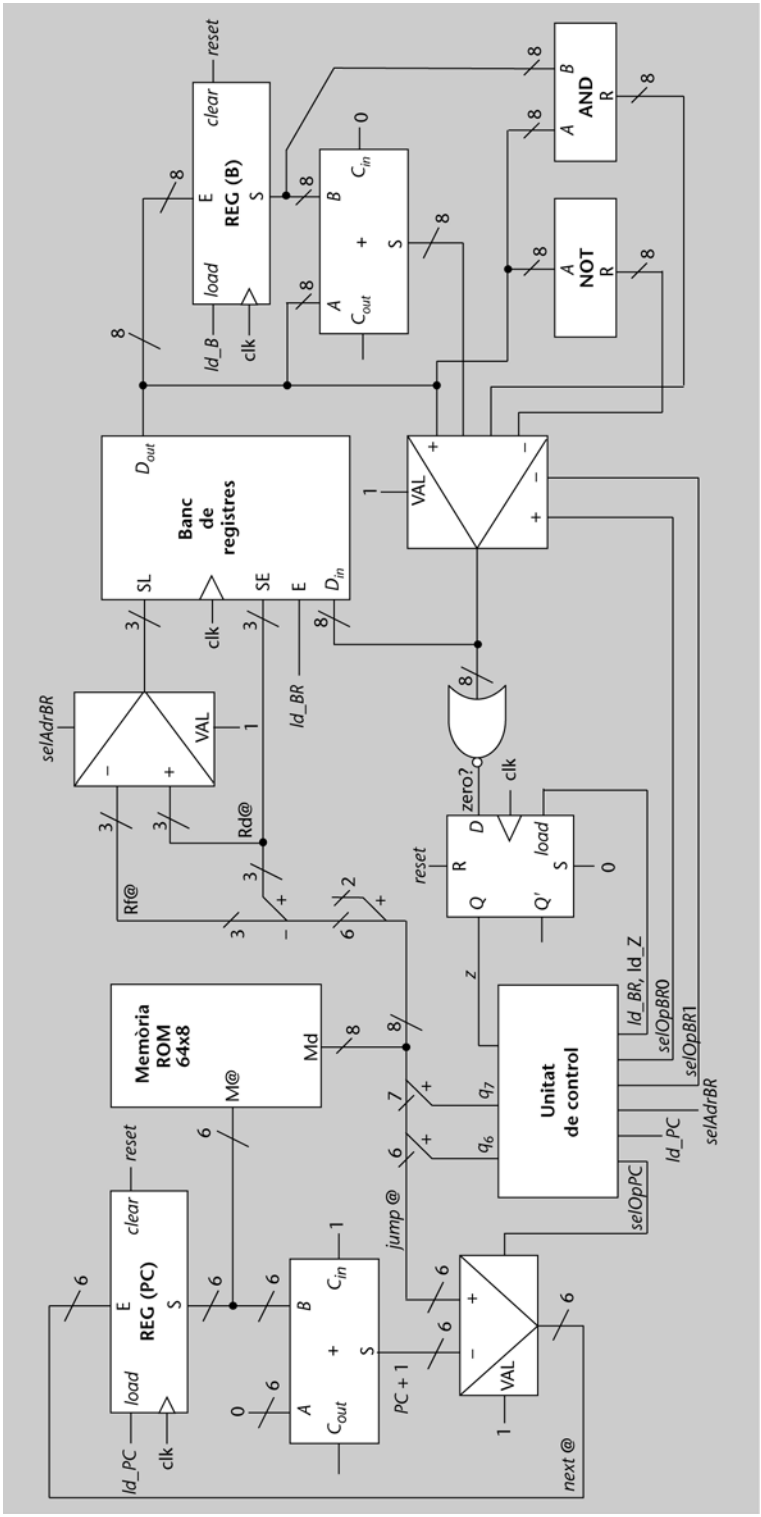
Figura 49. Unitat de control del Femtoproc, amb codificació d'estats de tipus *one-hot bit*



La unitat de control aprofita les sumes lògiques que implementen els arcs de tornada al node de decisió de q_7 per generar els senyals *selAdrBR* i *ld_BR* (o *ld_Z*).

Finalment, doncs, l'esquema del circuit complet del Femtoproc es pot veure en la figura 50. Si s'observa el dibuix en el sentit de lectura del text que conté, a l'esquerra de la unitat de control hi ha el circuit per a actualitzar el *PC*, que depèn de *selOpPC*. A la dreta hi ha la part d'operacions amb dades del banc de registres BR: complement, producte lògic i suma aritmètica. En aquests dos últims casos, es pren un operand del registre *B*, que haurà emmagatzemat el contingut de Rf en el cicle de rellotge anterior (amb *ld_B* = 1). El multiplexor de busos que hi ha sota de BR és el que tria quina operació s'ha de fer en la instrucció en curs i, per tant, (*selOpBR*1, *selOpBR*0) decideix quin valor s'ha d'emmagatzemar a BR[Rd] i que s'ha de comparar amb zero per a actualitzar el biestable corresponent, que genera el senyal *z* per a la unitat de control. Per aquest motiu, *ld_BR* = *ld_Z*.

Figura 50. Esquema complet del Femtoproc, amb la unitat de control com a mòdul



Com s'ha comentat, amb aquesta màquina d'interpretació es podria executar qualsevol programa construït amb les instruccions que pot descodificar. En l'exemple següent es mostra un programa per a calcular el màxim comú divisor de dos nombres.

En aquest cas, se suposa que els registres *R6* i *R7* s'han substituït pels valors d'entrada del programa (és a dir, són registres només de lectura) i que el resultat queda a *R5*. Cal tenir present que, inicialment, tots els registres, excepte *R0*, *R1*, *R6* i *R7*, estan a 0.

El programa calcula el màxim comú divisor per restes successives, segons l'expressió següent:

$$\mathrm{MCD}(a,b) = \begin{cases} \mathrm{MCD}(a-b,\, b), \text{ si } a > b \\ \mathrm{MCD}(a,\, b-a), \text{ si } b > a \\ a, \text{ si } b = 0 \\ b, \text{ si } a = 0 \end{cases}$$

La taula següent mostra el contingut de la ROM del Femtoproc per al càlcul del MCD. Les primeres entrades, en què en la columna "Instrucció" posa "configuració" són de caire explicatiu i no pas de contingut. Bàsicament, estableixen quins registres es faran servir d'entrada de dades des de l'exterior i quins de sortida. En la resta de files hi ha l'adreça i el contingut de la memòria. L'adreça s'hi expressa amb nombres en hexadecimal, per això se'ls posa una *h* al final. En cas que sigui una adreça de salt, també s'hi posa una etiqueta que les identifiqui perquè el flux de control del programa sigui més fàcil de seguir. Les instruccions es representen amb els símbols que es corresponen amb les operacions a fer (ADD, AND, NOT o JZ) i els que representen els operands, que poden ser registres o adreces. El nom dels registres comença per *R* i porta un nombre que n'identifica la posició en el banc de registres BR.

Adreça	Instrucció	Codificació	Comentari
---	configuració	–	*R0* = 1000 0000 = 80h, bit de signe.
---	configuració	–	*R1* = 0000 0001 = 01h, bit unitari.
---	configuració	–	*R5*, registre de sortida.
---	configuració	–	*R6*, registre d'entrada.
---	configuració	–	*R7*, registre d'entrada.
00h	ADD R2, R7	00 010 111	Passa un dels valors a un registre de treball.
01h	JZ 12h (end)	11 010010	Si és 0, l'MCD és l'altre nombre.
02h	ADD R3, R6	00 011 110	Passa l'altre valor a un segon registre auxiliar.
03h	JZ 12h (end)	11 010010	Si és 0, l'MCD és el primer nombre.
sub,04h	NOT R4, R2	10 100 010	*R4* = NOT(*R2*)
05h	ADD R4, R1	00 100 001	*R4* = –*R2* = Ca2(*R2*) = NOT(*R2*) + 1
06h	ADD R4, R3	00 100 011	*R4* = *R3* – *R2*
07h	AND R0, R4	01 000 100	Comprova el bit de signe, el resultat no es desa, perquè *R0* és només de lectura.
08h	JZ 0Dh (pos)	11 001101	Si és positiu (si *R3* > *R2*) passa a *pos*.
09h	NOT R2, R4	10 010 100	Si és negatiu, es canvia de signe el resultat.
0Ah	ADD R2, R1	00 010 001	Si es deixa a *R2*: *R2* = –(*R3* – *R2*)
0Bh	AND R0, R1	01 000 001	Força que el resultat sigui 0.
0Ch	JZ 04h (sub)	11 000100	Torna a fer una altra resta.
pos,0Dh	NOT R3, R4		
0Eh	NOT R3, R3		*R3* = *R4* = *R3* – *R2*
0Fh	JZ 12h (end)		
10h	AND R0, R1		
11h	JZ 04h (sub)		Torna a fer una altra resta.
end,12h	ADD R5, R2		

Adreça	Instrucció	Codificació	Comentari
13h	`ADD R5, R3`		*R5* = *R2* + *R3*, però un dels dos és zero.
14h	`AND R0, R1`		En espera, fins que se li faci *reset*.
hlt,15h	`JZ 15h (hlt)`		

Tot i que la màquina que s'ha vist en aquest exemple seria plenament operativa, només serviria per a executar programes molt simples: caldria incrementar el repertori d'instruccions, disposar de més registres per a dades i que la memòria de programa fos més gran.

Activitats

12. Completeu la columna de la codificació en la taula anterior.

13. Localitzeu, en el programa anterior, la seqüència d'instruccions que permet copiar el valor d'un registre a un altre.

14. Amb quina seqüència d'instruccions es poden desplaçar els bits d'un registre cap a l'esquerra?

3.2. Màquina elemental

S'anomena **llenguatge de màquina** el llenguatge en què es codifiquen els programes que interpreta una màquina determinada. El llenguatge de màquina de la màquina algorísmica que s'ha comentat en el subapartat anterior permet, en teoria, executar qualsevol algorisme que estigui descrit en un **llenguatge d'alt nivell** d'abstracció com C++ o C, un cop traduït. De totes maneres, excepte per als programes més senzills, la conversió en programes en llenguatge de màquina no seria factible per diversos motius:

- L'espai de dades és molt limitat per a encabir-hi les que es fan servir habitualment en un programa d'alt nivell. Cal tenir en compte que, per exemple, una paraula s'emmagatzema com una sèrie de caràcters i, segons la longitud, podria ocupar fàcilment tot el banc de registres, encara que s'ampliés significativament. De fet, la paraula "Femtoproc" no es pot guardar en el Femtoproc perquè ocupa més de sis caràcters (o bytes).

- La memòria de programa té poca capacitat si es té en compte que les instruccions d'alt nivell s'han de dur a terme amb instruccions molt simples del repertori del llenguatge màquina. Per exemple, una instrucció d'alt nivell de repetició d'un bloc de programa s'hauria de convertir en un programa en llenguatge de màquina que avalués la condició de repetició i passés a l'execució del bloc. Així doncs, una cosa com:

  `mentre (c < f) fer B;`

  s'hauria de convertir a un programa que llegís el contingut de les variables *c* i *f*, les comparés i, segons el resultat de la comparació, saltés a la primera instrucció del bloc *B* o a la instrucció d'alt nivell següent.

- El repertori d'instruccions és massa reduït per a poder dur a terme operacions comunes dels programes d'alt nivell de manera eficient. Per exemple, seria molt costós de traduir una suma d'una sèrie de nombres, ja que s'hau-

rien de tenir tantes instruccions com nombres a sumar. Un cas més simple és el de la resta. Una cosa com $R3 = R3 - R2$ necessita tres instruccions en llenguatge de màquina: el complement de $R2$, l'increment en 1 i, finalment, la suma de $R3$ amb $-R2$.

En una màquina elemental destinada a executar programes cal resoldre bé aquests problemes.

El repertori d'instruccions ha de ser més complet, cosa que implica que l'algorisme d'interpretació sigui més complex i que la mateixa màquina sigui més complexa, ja que hi ha d'haver més recursos per a poder dur a terme totes les operacions del repertori. Així doncs, caldria tenir, com a mínim, una instrucció de llenguatge de màquina per a cadascuna de les operacions aritmètiques, lògiques i de moviment més habituals. A més, ha d'incloure una varietat més elevada de salts, incloent-n'hi un d'incondicional. D'aquesta manera, la traducció de programes de nivell d'abstracció més alt a programes en llenguatge de màquina és més directa i el codi executable, més compacte (això sí, cal tenir present que la màquina interpretadora serà més gran i complexa).

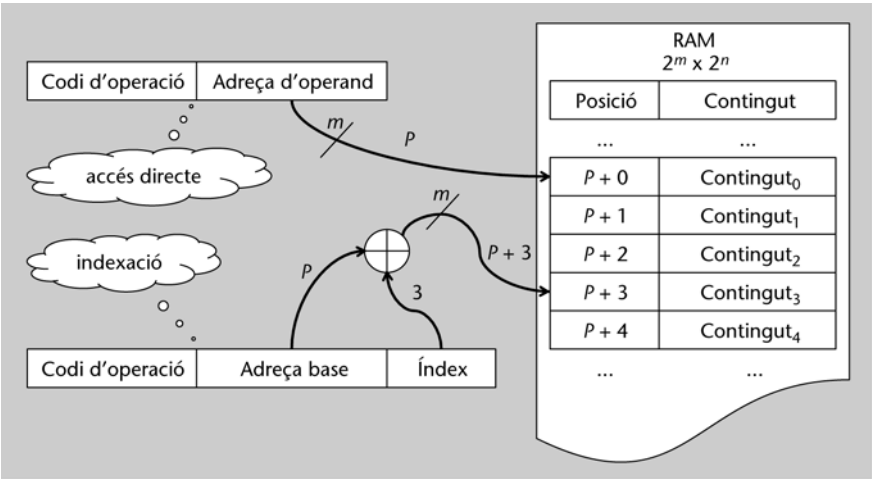
La memòria de programes ha de tenir molta més capacitat i, conseqüentment, les adreces han de ser de més bits. En aquest sentit, s'ha de tenir en compte que la codificació de les instruccions també es fa amb més bits, ja que el repertori s'ha ampliat.

L'espai reservat a les dades també ha de créixer. D'una banda, el nombre de registres de la màquina es pot ampliar, però resulta complex decidir fins quin punt. La solució, a més d'aquesta ampliació, consisteix a utilitzar una memòria per a les dades.

Accés a dades en la memòria

Per a l'accés a les dades en la memòria es pot fer servir un accés directe amb la seva adreça. Convé que hi hagi, a més, mecanismes d'accés indexat, en què sigui possible accedir a una sèrie de dades mitjançant una adreça base i un índex per a fer referència a una dada concreta de la sèrie. D'aquesta manera, la manipulació de les dades per part dels programes en llenguatge de màquina és més eficient. En la figura 51 hi ha un exemple de cadascun, amb un codi d'instrucció que inclou el codi de l'operació a fer, l'adreça de l'operand (P) i, per a l'accés indexat, un índex que val, en l'exemple, 3.

Figura 51. Il·lustració dels accessos directe i indexat a l'operand d'una instrucció



Prenent com a referència el cas del Femtoproc, la memòria de programació és de tipus ROM. Les memòries per a dades, en canvi, han de permetre tant lectures com escriptures. En aquest sentit, si s'hagués de millorar aquesta màquina, s'hi haurien d'incorporar memòries per a instruccions i per a dades diferenciades.

D'aquesta manera de construir les màquines, amb una memòria per a les instruccions i una altra per a les dades, se'n diu **arquitectura Hardvard**.

Arquitectura Hardvard

L'arquitectura Hardvard es denomina així perquè reflecteix el model de construcció d'una de les primeres màquines computadores que es va fer: l'IBM Automatic Sequence Controlled Calculator (ASCC), denominat *Mark I*, en la universitat de Hardvard pels volts del 1944. Aquesta màquina tenia l'emmagatzematge de les instruccions (en una cinta perforada) físicament separat del de les dades (en una mena de comptadors electromecànics).

De vegades, però, pot resultar més senzill tenir tant les instruccions com les dades en la mateixa memòria. D'aquesta manera, la màquina només ha de gestionar els accessos a una única memòria i, addicionalment, pot executar tant programes petits que facin servir moltes dades com d'altres de molt grans que en tractin poques.

Les màquines construïdes de manera que facin servir una mateixa memòria per a dades i instruccions segueixen l'**arquitectura de Von Neumann**.

Arquitectura de Von Neumann

L'arquitectura de Von Neumann es descriu a *First Draft of a Report on the EDVAC*, datat a 30 de juny de 1945. Aquest informe recollia la idea de construir computadors amb un programa emmagatzemat en la memòria en lloc de fer-ho com a màquines per a algorismes específics. De fet, descriu l'estat de l'art a l'època i proposa la construcció d'una màquina anomenada *electronic discrete variable automatic computer* ('computador electrònic automàtic de variables discretes'), que es basa, sobretot, en el treball fet entorn d'un altre computador primerenc: l'ENIAC o *electronic numerical integrator and computer* ('integrador numèric i calculador electrònic').

Amb independència de l'arquitectura que se segueixi per a materialitzar aquestes màquines, és necessari que puguin interpretar un repertori d'instruccions suficient per a permetre una traducció eficient dels programes en llenguatges d'alt nivell i que tinguin una capacitat de memòria prou gran per a encabir-hi tant les instruccions com les dades dels programes que han d'executar.

3.2.1. Màquines algorísmiques d'unitats de control

Una màquina d'aquest estil encara es pot veure com una màquina algorísmica d'interpretació, però l'algorisme que interpreta les instruccions del llenguatge màquina és força més complex que el que s'ha vist anteriorment. En línies generals, l'intèrpret treballa de manera similar, però les fases del cicle d'execució de les instruccions s'allarguen fins a trigar diversos períodes de rellotge. És a dir, el nombre de passos per a arribar a executar una única instrucció d'un programa en llenguatge màquina és elevat.

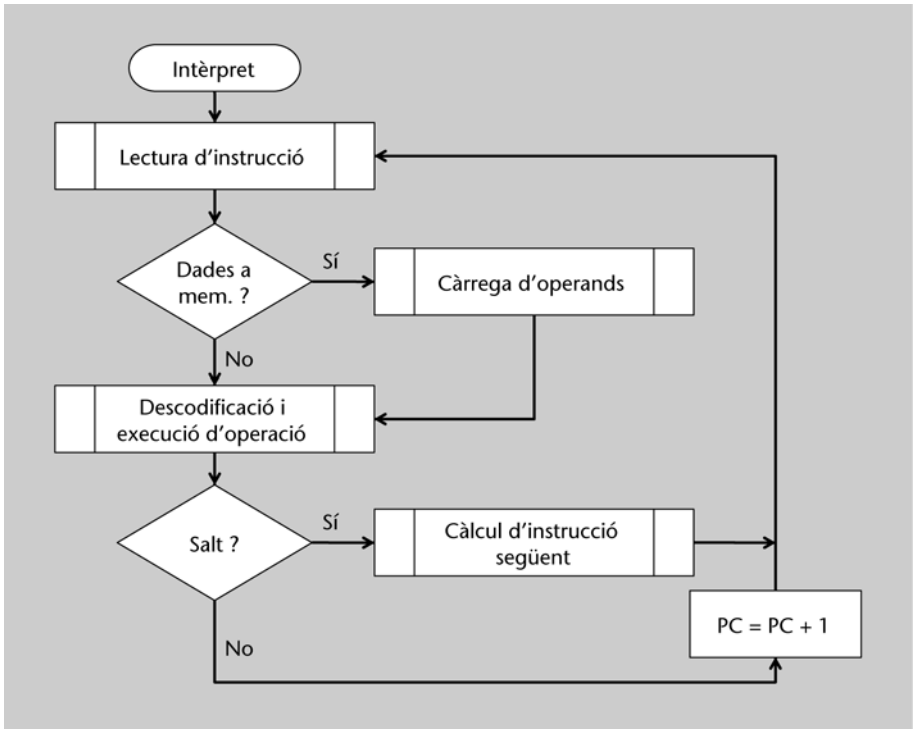
Cal tenir present que la lectura d'una sola instrucció pot necessitar de diverses passes, atès que l'amplada en bits n'ha de permetre codificar un repertori gran, cosa que fa que pugui ocupar més d'una paraula de memòria.

A més, en cas que es tracti d'instruccions que treballin amb dades a memòria, cal fer-ne la lectura, que pot no ser directa i, conseqüentment, allargar-se una

sèrie de passos que s'inclouen en la fase de càrrega d'operands. Com que també és convenient que el ventall d'operacions que puguin dur a terme els computadors sigui prou gran, la descodificació d'aquestes per a poder-les executar i la seva mateixa execució sol requerir una sèrie de passos que depèn del tipus d'operació i, com a conseqüència, hi ha una diferència de temps entre l'execució d'unes i altres instruccions. L'ampliació de la varietat de salts també complica de manera similar el càlcul de la instrucció següent.

La figura 52 resumeix el flux de control d'una unitat de processament que implementa un d'aquests repertoris d'instruccions. Cada etapa o fase s'ha de resoldre amb una sèrie de passos que s'allarguen durant uns quants cicles de rellotge. Cal tenir en compte que es tracta de nodes de procés predefinit i, per tant, que es corresponen amb esquemes de càlcul o, fins i tot, màquines algorísmiques completes.

Figura 52. Diagrama de flux del cicle d'execució d'instruccions



La implementació d'aquestes màquines algorísmiques d'interpretació es podria fer de manera similar a la que s'ha vist, a partir dels diagrames de flux corresponents, però tenint en compte que la seqüenciació dels passos a seguir és molt complexa. Conseqüentment, la unitat de control té un nombre d'estats enorme que fa poc pràctica la implementació com a circuit combinacional basat en blocs lògics i registres.

En la màquina algorísmica anterior, corresponent al cas del Femtoproc, l'execució de les instruccions dura entre un i dos cicles de rellotge. Conseqüentment, les seqüències de passos són força simples. En màquines que interpretin instruccions més complexes, les seqüències s'allarguen. Si s'ajunta amb el fet que la quantitat de combinacions d'entrades possibles per a la part controla-

dora creix exponencialment, ja que els repertoris d'instruccions són més grans (més bits per a codificar les operacions corresponents) i, a més, es fan servir molts més bits de condició de la part operativa (indicació de zero, arrossegament o *carry*, sobreeiximent, etc.), el nombre potencial d'estats creix de la mateixa manera.

Per aquest motiu, és convenient modelar la part de control d'aquestes màquines algorísmiques amb d'altres màquines algorísmiques encarregades d'interpretar les primeres. En aquest cas, també seran màquines algorísmiques d'interpretació d'instruccions, però d'un repertori més limitat. Per exemple, una instrucció per a cada tipus de node en un diagrama de flux.

Per a distingir aquesta màquina de la màquina que interpreta el repertori d'instruccions del processador, es parla de la màquina que interpreta les microinstruccions de la unitat de control.

Les **microinstruccions** són les operacions que es fan amb els recursos de càlcul d'una unitat de processament en un cicle de rellotge. Cada una de les ordres que es dóna als elements de la unitat de processament es denomina **microordre**. El programa d'interpretació de les instruccions del llenguatge de la màquina s'anomena, òbviament, **microprograma**.

Els processadors que interpreten repertoris d'instruccions complexos solen estar microprogramats, és a dir, la unitat de control corresponent és una màquina algorísmica implementada amb un intèrpret de microprogrames i el microprograma corresponent.

3.2.2. Màquines algorísmiques microprogramades

Com s'ha comentat, el repertori de microinstruccions sol ser molt reduït. Per exemple, en pot tenir de dos tipus, amb correspondència amb els nodes de processament i decisió de les màquines algorísmiques:

1) Les de **processament** són les que duen a terme alguna operació amb el camí de dades, és a dir, les que efectuen un conjunt de microordres en un únic cicle de rellotge. Una microinstrucció comuna d'aquest primer tipus és la que aplega les microordres necessàries per a carregar en un registre de destinació el resultat d'una operació feta amb dades provinents de diversos registres font.

2) Les de **decisió** serveixen per a fer salts condicionals. Per exemple, una microinstrucció de salt condicional activaria les microordres per a seleccionar quina condició s'ha de complir i la de càrrega del comptador de programa.

També hi ha l'opció de tenir un repertori basat en un tipus únic de microinstrucció que consisteixi en un salt condicional i un argument variable que indiqui quines microordres s'han d'executar en el cicle corresponent.

En la taula següent hi ha un exemple d'un possible format d'un repertori com el del primer cas. Cal tenir present que el nombre de bits ha de ser prou gran per a encabir-hi totes les possibles microordres del camí de dades i les adreces de la memòria de microprogramació, que ha de tenir capacitat suficient per a emmagatzemar el microprograma d'interpretació del repertori d'instruccions.

Micro-instrucció	**Bits (1 per microordre)**															
	15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
EXEC	0	selectors d'entrada, càrregues de registres, operació de l'ALU, etc														
JMPIF	1	selectors de condició				–	–	adreça de salt (microinstrucció següent)								

El camí de dades se suposa organitzat d'una manera similar a la d'un esquema de càlcul amb diversos registres i un únic recurs de càlcul programable, per a la qual cosa cal proporcionar l'operació que es fa en una microinstrucció concreta.

L'arquitectura del camí de dades que s'ha vist i que es basa en l'ús d'un únic recurs de càlcul programable és força comuna a la majoria de processadors. Com que aquest recurs fa tant operacions aritmètiques (suma, resta, canvi de signe, increments, etc.) com lògiques (suma i producte lògic, complement, desplaçaments de bits, etc.), s'hi fa referència com la unitat aritmeticològica (o ALU, de l'anglès, *arithmetic logic unit*) del camí de dades del processador.

La màquina algorísmica que interpreti un repertori de microinstruccions com l'anterior permetria executar un microprograma de control d'una altra màquina algorísmica que seria, finalment, el processador del repertori d'instruccions d'un llenguatge de màquina concret. Atès que es tracta d'una màquina algorísmica molt senzilla que s'ocupa d'executar les microinstruccions en una seqüència determinada, se sol anomenar **seqüenciador**.

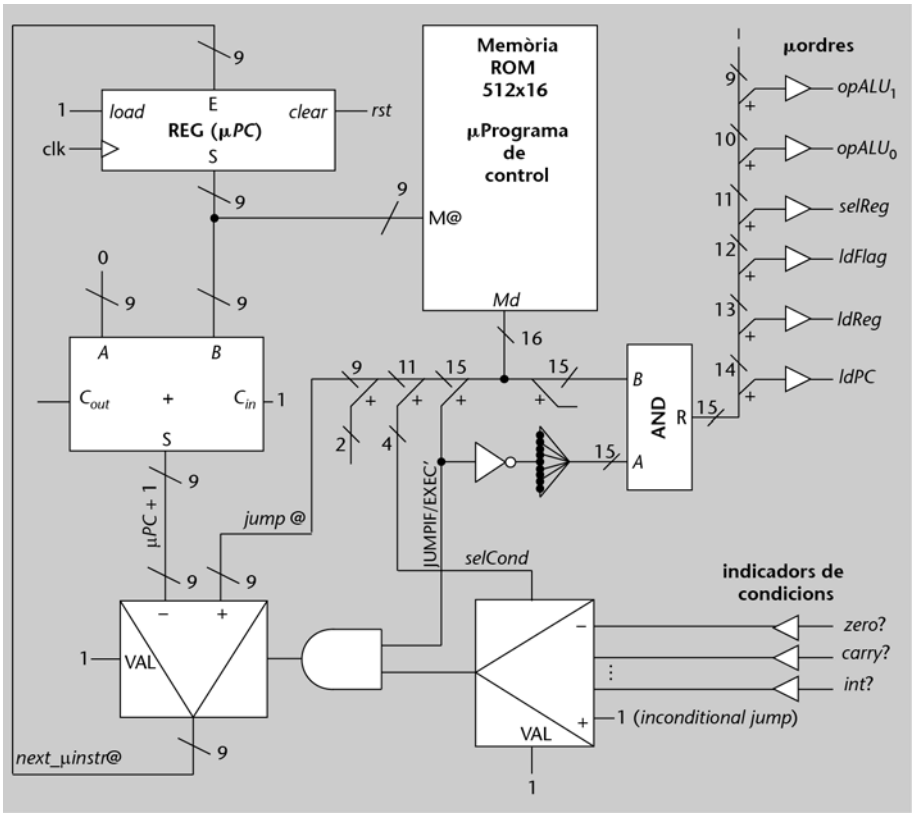
Tot i que el seqüenciador amb EXEC i JMPIF és funcionalment correcte, cal tenir present que durant l'execució del JMPIF no es du a terme cap operació en la unitat de processament. Hi ha diverses opcions per a aprofitar aquest cicle de rellotge per part del camí de dades. Una de les més senzilles és que el seqüenciador treballi amb un rellotge el doble de ràpid que el de la unitat de processament que controla i introduir un cicle d'espera entre EXEC i EXEC. Una altra consisteix a aprofitar els bits que no s'aprofiten (per exemple, en la taula anterior, el JMPIF no fa servir els bits en posició 9 i 10) com a bits de codificació de determinades microordres. Així, amb el JMPIF també es durien a terme algunes operacions en la unitat de processament (de fet, és un cas com el del repertori de tipus únic, en què totes les microinstruccions inclouen microordres i adreces de salt). Per tant, el problema de no aprofitar la unitat de processament durant els salts es minimitza.

En la figura 53 hi ha l'esquema d'un circuit seqüenciador per al repertori de microinstruccions que s'acaba de comentar. Es tracta d'un seqüenciador paral·lel, en què cada microordre té un bit associat. Per aquest motiu, aquesta mena de seqüenciadors fan servir memòries amb paraules molt amples. A la pràctica, sol compensar tenir una codificació més compacta que en redueixi l'amplada a canvi de fer servir circuits per a descodificar les microinstruccions.

La memòria ROM conté una codificació binària del microprograma que executa una màquina algorísmica d'interpretació d'instruccions particular. Cada paraula de la memòria segueix el format que s'ha detallat en la taula anterior.

En cas que es tracti d'una microinstrucció de tipus EXEC, els bits $Md_{14–0}$ són les microordres que hi estan associades. Els senyals corresponents s'han identificat amb noms que exemplifiquen els tipus de microordres que hi pot haver en una màquina tipus: *ldPC* per a carregar un nou valor al comptador de programa; *ldReg* perquè un registre desi, en acabar el cicle de rellotge actual, algun nou resultat, que es tria amb *selReg*; $opALU_0$ i $opALU_1$ per a seleccionar l'operació que ha de fer l'ALU; *ldFlag* per a actualitzar els biestables que emmagatzemen condicions sobre el darrer resultat (si ha estat zero, si s'ha produït ròssec, entre d'altres), i així fins a quinze. Al mateix temps que se subministren aquests senyals a la part operacional, es calcula l'adreça de la microinstrucció següent, que és la que es troba en la posició de memòria següent. Per això, el comptador de microprograma (*μPC*) s'incrementa amb 1: el multiplexor que genera *next_μinst@* selecciona l'entrada *μPC* + 1 quan s'està executant una microinstrucció EXEC.

Figura 53. Esquema d'un circuit seqüenciador de microinstruccions



Quan s'executa un JMPIF totes les microordres són inactives perquè es fa un producte lògic amb el bit que indica si és JMPIF o EXEC, de manera que, quan JMPIF/EXEC' (Md_{15}) sigui 1, les microordres siguin 0. En aquest cas, els bits $Md_{14–0}$ no són microordres sinó que representen, d'una banda, la selecció de la condició (*selCond*, que és $Md_{14–11}$) que cal tenir en compte per a efectuar, o no, el salt en la seqüència de microinstruccions, i, de l'altra, l'adreça de la mi-

croinstrucció següent (*jump@*, que és Md_{8-0}) en cas de salt. El senyal *selCond* s'ocupa de triar quin bit de condició provinent del camí de dades s'ha de tenir en compte a l'hora de fer el salt. Com es pot veure en la figura, aquests bits poden indicar si l'últim resultat ha estat zero (*zero*?), si s'ha produït arrossegament (*carry*?) o si cal interrompre l'execució del programa (*int*?), entre d'altres. Cal fixar-se que una de les entrades del multiplexor corresponent és 1. En aquest cas, la condició sempre es complirà. Això serveix per a fer salts incondicionals. La decisió d'efectuar o no el salt depèn de la condició seleccionada amb *selCond* i de si s'està executant una microinstrucció JMPIF, d'això que hi hagi la porta AND abans de generar el control del multiplexor que genera *next_µinst@*.

Amb un seqüenciador com aquest seria possible construir controladors per a màquines algorísmiques relativament complexes: els seus diagrames de flux podrien arribar a tenir fins a 512 nodes de condició o processament.

Per a fer una màquina capaç d'executar programes de propòsit general que segueixi l'arquitectura de Von Neumann, és habitual implementar les màquines algorísmiques d'interpretació del repertori d'instruccions corresponent amb una unitat de control microprogramada.

3.2.3. Una màquina amb arquitectura de Von Neumann

A tall d'exemple, es presenta un processador senzill d'arquitectura Von Neumann que utilitza pocs recursos en el camí de dades i una unitat de control microprogramada. De fet, és un altre dels molts processadors senzills que hi ha pel món i, per això, es denomina **YASP** (de l'anglès, *yet another simple processor*).

El YASP treballa amb dades i instruccions de 8 bits, que han d'estar emmagatzemades en la memòria principal o provenir de l'exterior per mitjà d'algun port d'entrada (un registre que carrega informació que prové d'algun perifèric d'entrada).

La memòria principal és de només 256 bytes, de manera que n'hi ha prou amb un únic byte per a representar totes les posicions de memòria possibles, des de la 0 fins a la 255.

El repertori d'instruccions que entén el YASP consta de les que fan operacions aritmeticològiques, les de moviment d'informació, les de manipulació de bits i, finalment, les de control de flux. Com és habitual, les operacions aritmètiques es fan considerant que els nombres es representen en complement a 2.

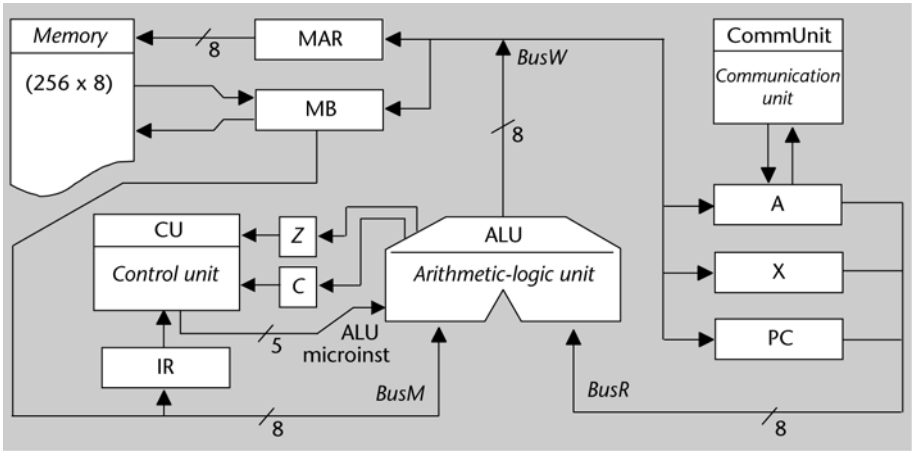
La codificació de les instruccions inclou camps (grups de bits) per a identificar el tipus d'instrucció, l'operació que ha de fer l'ALU i els operands que hi par-

ticipen, entre d'altres. L'argument per a accedir al segon operand, si cal, es troba en un byte addicional. Així doncs, hi ha instruccions que ocupen 1 byte i d'altres, 2.

Els formats de les instruccions dels llenguatges de màquina solen ser molt variables perquè ajusten les amplades de bits al tipus d'operació que fan i als operands que necessiten. Així, és habitual tenir codificacions amb instruccions que ocupen un únic byte amb d'altres que en necessiten 2, 3, 4 o més. En el llenguatge de màquina del YASP, n'hi ha que ocupen 1 byte, si no necessiten cap dada addicional, i n'hi ha que n'ocupen 2. En el primer cas hi ha, per exemple, les instruccions que treballen amb registres. En el segon, les que treballen amb dades en la memòria.

En l'esquema de YASP que hi ha a la figura 54 es poden veure tots els elements que en formen part i com estan connectats. Hi ha tres busos de connexió: el de memòria (*BusM*), el de registres (*BusR*) i, finalment, el de resultats (*BusW*), que poden ser transportats cap a la memòria o cap als registres. La comunicació amb l'exterior es fa per mitjà del registre A i un mòdul específic (CommUnit), que també està governat per la unitat de control (CU). No es mostren els senyals de control. Els senyals d'entrada de la CU són els del codi de la instrucció en curs, que s'emmagatzema a IR, i els indicadors de zero (Z) i d'arrossegament (C).

Figura 54. Diagrama de blocs del processador elemental YASP



Per a poder processar les dades, disposa d'un registre de 8 bits anomenat **acumulador** (o registre A), ja que és el registre en el qual s'acumularia el resultat de la suma d'una seqüència de bytes. De fet, el resultat de qualsevol operació aritmètica o lògica es desa en aquest registre.

Aquestes operacions es porten a terme en l'ALU, que pren un possible segon operand de la memòria principal per mitjà del registre MB, de *memory buffer*. Hi ha dos elements de memòria addicionals d'1 bit cadascun per a indicar si el resultat de l'operació ha estat 0 o no (el bit Z) i si l'operació ha generat arrossegament o no (el bit C).

Per a fer les operacions d'accés a aquest segon operand i, en general, per a accedir a qualsevol posició de memòria, cal desar-ne l'adreça en el registre d'adre-

ces de memòria (MAR, de l'anglès *memory address register*) abans d'efectuar la lectura del contingut de memòria o d'escriure-hi el contingut del MB.

Com que l'adreça del segon operand no sempre es coneix per endavant, també es pot donar l'adreça d'una posició de memòria en què l'adreça d'aquest operand estigui desada. És el que es coneix com a **indirecció**, perquè no dóna directament l'adreça de l'operand.

De vegades, però, resulta convenient treballar amb una sèrie de dades de les que se'n coneix l'adreça inicial, és a dir, la localització de la primera dada de la sèrie. En aquest cas, cal sumar a l'adreça inicial la posició de la dada amb la qual es vol treballar. Per a fer aquesta indexació, el YASP disposa d'un registre auxiliar addicional, el registre X. Així, l'adreça de l'operand es calcula sumant a l'adreça base (l'adreça de la primera dada) el contingut del registre X.

Resumint, les instruccions del llenguatge de màquina del YASP tenen quatre maneres diferents d'obtenir l'adreça del segon operand d'una operació diàdica (amb dos operands, un dels quals és el registre A):

1) Immediata. El valor de l'operand s'especifica juntament amb l'operació.

2) Directa. L'adreça de l'operand s'indica amb l'operació. Cal cercar-ne el valor en la posició indicada.

3) Indirecta. La posició de memòria en què hi ha l'operand està guardada en l'adreça que acompanya l'operació. És a dir, es dóna l'adreça en què hi ha l'adreça real de l'operand.

4) Indexada. L'adreça de l'operand s'obté sumant el registre índex X a l'adreça que s'adjunta amb l'operació.

Més formalment, les adreces dels operands s'anomenen **adreces efectives** per a distingir-les de les que es donen en les instruccions del programa que s'executa; i les maneres d'obtenir les adreces efectives s'anomenen **modes d'adreçament**. En concret, s'han descrit els modes immediat, directe, indirecte i indexat.

Havent obtingut el segon operand, l'ALU pot fer l'operació entre aquest i el contingut del registre A. Normalment, el resultat es deixa en el mateix registre. Si l'operació només necessita un operand, llavors A és l'únic operand i el resultat de l'operació monàdica s'hi desa tot seguit.

En el cas d'aquesta màquina elemental, l'ALU fa sumes, restes, càlculs de l'oposat (canvis de signe), increments, decrements, *i*s lògiques, *o*s inclusives i exclusives, desplaçaments i rotacions. També permet el pas directe de la informació, sense fer cap operació.

Normalment, la instrucció següent es troba a continuació de la que s'acaba d'executar; és a dir, n'hi ha prou d'incrementar el contingut del PC en un o en dos, segons la instrucció que s'executi. De vegades, però, interessa anar a un punt diferent del programa. Aquest salt pot ser incondicional, o condicional, si depèn de l'estat de la màquina, que, en aquest cas, està definit pels bits C i Z.

En el cas dels salts condicionals només es reserva 1 byte per a la seva codificació i només hi ha 5 bits per a expressar l'adreça de salt, ja que els altres 3 són necessaris per a codificar l'operació. Consegüentment, no es pot saltar a qualsevol posició de memòria, sinó que només es pot anar a una instrucció situada setze posicions endavant o quinze endarrere. Això és així perquè se suma un valor en Ca2 en el rang [–16, 15] al PC incrementat. Aquestes instruccions utilitzen un **mode d'adreçament relatiu**; és a dir, l'adreça que s'hi adjunta se suma al contingut del PC. Per tant, l'adreça de la instrucció a la qual se salta es dóna en relació amb l'adreça de la instrucció següent a la del salt condicional.

Sigui com sigui, l'execució de les instruccions del llenguatge de màquina del YASP s'allarguen uns quants cicles de rellotge, en els quals s'executa el microprograma corresponent. Per aquest motiu, el codi d'operació de la instrucció en curs es guarda en el registre d'instruccions (IR), de manera que la unitat de control pugui decidir què ha de fer i en quin ordre.

Tot i que el YASP és un processador senzill, és capaç d'interpretar un repertori prou complet d'instruccions gràcies a una unitat de control prou complexa, que es pot materialitzar de manera directa si es microprograma.

Fent servir un llenguatge de transferència de registres (RTL, de l'anglès *register transfer language*) es mostra a continuació el microprograma corresponent a l'execució d'una instrucció de suma entre A i un operand a memòria.

En la columna "Etiqueta" hi ha el símbol que identifica una posició concreta de la memòria de microprogrames.

La columna que conté les microinstruccions detalla quines transferències de registres es faran a cada cicle de rellotge, si es tracta d'EXEC. Les càrregues simultànies de valors a l'MB i al PC són possibles perquè fan servir busos diferents. De fet, l'MB es carrega amb el valor de l'entrada de dades que prové de la memòria.

En el cas de les microinstruccions de tipus JMPIF, el primer argument és la condició que es comprova, i el segon, l'adreça de salt, en cas que sigui certa.

En aquest exemple hi ha la seqüència completa de microinstruccions per a executar una operació de suma del llenguatge de màquina del YASP.

Etiqueta	Codi d'operació	μ–**instrucció**	Comentari
START:	EXEC	`MAR = PC`	**Fase 1**. Lectura de la instrucció
	EXEC	`MB = M[MAR], PC = PC + 1`	M[MAR] fa referència al contingut de la memòria a la posició MAR
	EXEC	`IR = MB`	
	JMPIF	`2nd byte?, DYADIC`	Descodifica si cal llegir un segon byte
...			
DYADIC:	EXEC	`MAR = PC`	**Fase 2**. Càlcul de l'operand
	EXEC	`MB = M[MAR], PC= PC +1`	
	JMPIF	`Immediate?, DO`	Adreçament immediat, operand llegit
	JMPIF	`Not indexed?, SKIP`	
	EXEC	`MB = MB + X`	Adreçament indexat
SKIP:	EXEC	`MAR = MB`	
	EXEC	`MB = M[MAR]`	Adreçament directe o indexat
	JMPIF	`Not indirect, DO`	
	EXEC	`MAR = MB`	
	EXEC	`MB = M[MAR]`	Adreçament indirecte
DO:	JMPIF	`ADD?, X_ADD`	**Fase 3**. Descodificació operació i execució
...			
X_ADD:	EXEC	`A = A + MB`	
	JMPIF	`Inconditional, START`	Bucle infinit d'interpretació
...			

El gran avantatge de les unitats de control microprogramades és la facilitat de desenvolupament i manteniment posterior. En l'exemple no s'han posat els codis binaris corresponents, però la taula del microprograma és suficient per a obtenir el contingut de la memòria de microprogramació del seqüenciador associat.

Activitats

15. Sense considerar les operacions possibles de l'ALU, que es codifiquen amb 5 bits, tal com es mostra en la figura 54, indiqueu quines microordres caldrien per a controlar la unitat de processament del YASP. Per exemple, caldria tenir-ne una perquè el registre PC carregui la dada del *BusW*, que es podria dir *ld_PC*.

16. Amb l'esquema del YASP i prenent el microprograma de la suma com a exemple, feu-ne un per a una instrucció que incrementi el contingut del registre X. Podeu suposar que l'ALU pot fer una cosa com *BusW* = *BusR* + 1.

3.3. Processadors

Com ja s'ha comentat en la secció 3.1.1, les màquines algorísmiques dedicades a interpretar programes són, de fet, màquines que processen les dades d'acord amb les instruccions dels seus programes, és a dir, són **processadors**.

La implementació dels processadors és molt diversa, ja que depèn del repertori d'instruccions que hagin d'interpretar i de la funció de cost que es vulgui minimitzar.

En aquest subapartat s'explicaran les diferents organitzacions internes dels processadors i es farà una introducció a les que s'utilitzen en els de propòsit general i en els que estan destinats a tasques més específiques.

3.3.1. Microarquitectures

Les **microarquitectures** (de vegades s'abreugen com a *µarch* or *uarch*, en anglès) són els models de construcció d'un determinat repertori d'instruccions en un processador o, si es vol, d'una determinada arquitectura d'un conjunt d'instruccions (*instruction set architecture* o ISA, en anglès). Cal tenir present que una ISA es pot implementar amb diferents microarquitectures i que aquestes implementacions poden variar segons els objectius que es persegueixin en un determinat disseny o per canvis tecnològics (l'arquitectura d'un computador és la combinació de la microarquitectura i de l'ISA).

En general, tots els programes presenten unes certes característiques de "localitat", és a dir, de treballar localment. Per exemple, en un tros de codi concret, se sol treballar amb un petit conjunt de dades amb iteracions que es fan dins del mateix bloc d'instruccions. En aquest sentit, és raonable que les instruccions treballin amb dades emmagatzemades en un banc de registres i que n'hi hagi que facilitin la implementació de diverses formes d'iteració. En tot cas, un determinat repertori d'instruccions anirà bé per a un determinat tipus de programes però pot no ser tan eficient per a d'altres.

Hi ha arquitectures d'instruccions que requereixen molts recursos perquè cobreixen un rang d'operacions molt ampli (per exemple, amb aritmètica entera i de coma flotant, amb multitud d'operadors relacionals, amb operacions aritmètiques adaptades a la programació, com increments i decrements, etc.) i perquè fan servir moltes variables (per exemple, per a poder emmagatzemar dades temporals o per a servir d'ajuda a estructures de control). En general, les màquines que implementen aquestes ISA es coneixen com a CISC, de l'anglès *complex instruction set computer* (computador amb un conjunt d'instruccions complex).

En contraposició, hi ha repertoris d'instruccions que requereixen menys recursos i que són més fàcils de descodificar. Normalment, són força reduïts en

comparació als CISC, cosa que fa que rebin el nom de RISC, de l'anglès *reduced instruction set computer* (computador amb un conjunt d'instruccions reduït).

En general, els RISC tenen una arquitectura basada en un únic tipus d'instrucció de lectura i emmagatzemament. En altres paraules, tenen una única instrucció per a la lectura i una altra per a l'escriptura de/a memòria, encara que tinguin variants segons els operands que admetin. Aquest tipus d'arquitectures es denominen ***load/store***, i se solen oposar a d'altres en què les instruccions combinen les operacions amb els accessos a la memòria.

D'alguna manera, es podria dir que el Femtoproc, que s'ha vist en l'apartat 3.1.1, és una màquina RISC i que el YASP, que s'ha explicat en l'apartat 3.2.3, és una màquina CISC.

Tant els RISC com els CISC es poden construir amb microarquitectures similars, ja que tenen problemes comuns: accés a la memòria per a la lectura d'instruccions, accés a la memòria per a la lectura/escriptura de dades i processament de les dades, entre d'altres.

3.3.2. Microarquitectures amb *pipelines*

Per a anar més ràpid en el processament de les instruccions, se'n poden tractar diverses formant un ***pipeline***: una mena de canonada en què passa un flux que es tracta a cada segment d'aquesta, de manera que no cal esperar que es processi totalment una porció del flux per a tractar-ne una altra de diferent.

En les unitats de processament, els *pipelines* se solen formar amb les diferents fases del cicle d'execució d'una instrucció. En la figura 55, hi ha un *pipeline* d'una màquina similar al YASP, però que ha reduït els operands als immediats i directes.

Així doncs, el cicle d'execució d'una instrucció seria:

1) **Fase 1**. Lectura d'instrucció (LI).

2) **Fase 2**. Lectura d'argument (LA), que pot ser buida si no hi ha argument.

3) **Fase 3**. Càlcul d'operand (CO), que fa una nova lectura si és directe.

4) **Fase 4**. Execució de l'operació (EO), que pot fer diverses coses, segons el tipus d'instrucció: un salt, actualitzar l'acumulador o escriure en la memòria, entre d'altres.

Amb aquesta simplificació, cada fase es correspondria amb un cicle de rellotge i, per tant, una unitat de processament sense el *pipeline* trigaria quatre cicles

de rellotge a executar una instrucció. Com es pot veure, amb el *pipeline*, se'n podria tenir una d'acabada a cada cicle de rellotge, tret de la latència inicial de quatre cicles.

Figura 55. *Pipeline* de quatre etapes

Instrucció	Etapa del *pipeline* (fase del cicle)						
1	LI	LA	CO	EO			
2		LI	LA	CO	EO		
3			LI	LA	CO	EO	
4				LI	LA	CO	EO
5					LI	LA	CO
Període	1	2	3	4	5	6	7
	Latència						

Els *pipelines* són, doncs, una opció per a augmentar el rendiment de les unitats de processament i, per aquest motiu, la majoria de microarquitectures els inclouen.

Ara bé, també presenten un parell d'inconvenients. El primer és que calen registres entre etapa i etapa per a emmagatzemar les dades intermèdies. El segon és que tenen problemes a l'hora executar determinades seqüències d'instruccions. Cal tenir en compte que no és possible que una de les instruccions del *pipe* en modifiqui una altra que també hi sigui i, també, que cal "buidar" el *pipe* en acabat d'executar una instrucció de salt que el faci. En aquest últim cas, la conseqüència és que no sempre s'obté el rendiment d'una instrucció acabada per cicle de rellotge, ja que depèn dels salts que es facin en un programa.

Activitat

17. Refeu la taula de la figura 55 suposant que la segona instrucció és un salt condicional que sí s'efectua. És a dir, que després de l'execució de la instrucció la següent és una altra de diferent de la que hi ha a continuació. Amb vista a la resolució, suposeu que la seqüència d'instruccions és (1, 2, 11, 12, ...), és a dir, que salta de la segona a l'onzena instrucció.

3.3.3. Microarquitectures paral·leles

L'augment del rendiment també es pot aconseguir incrementant el nombre de recursos de càlcul de la unitat de processament per a poder fer diverses operacions de manera simultània. Per exemple, s'hi poden incloure diverses ALU que facilitin l'execució paral·lela de les diverses etapes d'un *pipeline* o de diverses operacions d'una mateixa instrucció. En aquest darrer cas, el paral·lelisme pot ser implícit (depèn de la instrucció) o explícit (la instrucció inclou arguments que indiquen quines operacions, i amb quines dades, es fan). Per al paral·lelisme explícit, la codificació de les instruccions necessita molts bits i,

per aquest motiu, es parla d'**arquitectures VLIW** (de l'anglès, *very long instruction word*).

D'altra banda, aquesta solució de múltiples ALU permet d'avançar l'execució d'una sèrie d'instruccions que siguin independents d'una que ocupi una unitat de càlcul uns quants cicles. És el cas de les unitats de coma flotant, que necessiten molt de temps per a acabar una operació en comparació amb una ALU que treballi amb nombres enters.

Una altra opció per a augmentar el rendiment és disposar de diverses unitats de processament independents (o *cores*, en anglès), cadascuna amb el seu propi *pipeline*, si és el cas.

Aquesta microarquitectura *multi-core* és adequada per a processadors que executen diversos programes en paral·lel, ja que, en un moment determinat, cada *core* es pot ocupar d'executar un programa diferent.

3.3.4. Microarquitectures amb CPU i memòria diferenciades

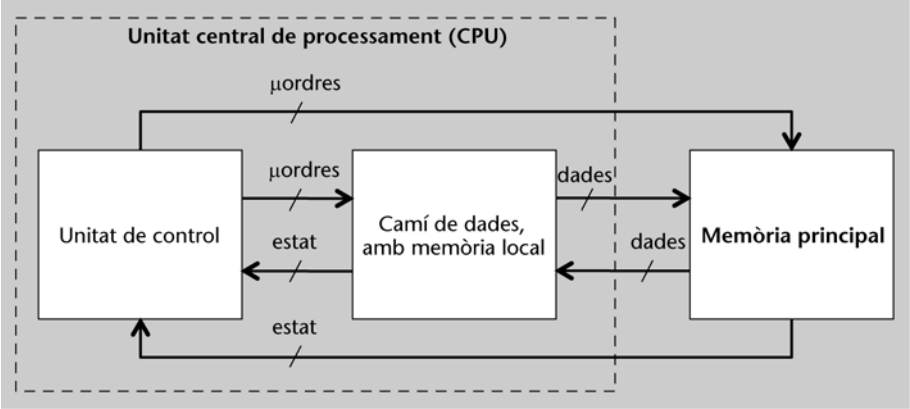
Atesa la necessitat de disposar de processadors amb memòries de gran capacitat, el que es fa és construir-los de manera que la part processadora tingui recursos de memòria suficients per a treballar de manera eficient (és a dir, que el nombre d'instruccions executades per unitat de temps sigui el més gran possible) però que no inclogui el gros de la informació (dades i instruccions), ja que es pot emmagatzemar millor amb tecnologia específica.

Així doncs, el model de construcció dels processadors ha de tenir en compte els condicionaments tecnològics que hi ha per a la seva materialització. Generalment, les memòries es construeixen amb tecnologies que n'incrementin la capacitat. En canvi, els circuits de processament es fan amb uns criteris ben diferents: poder processar moltes dades amb el mínim temps possible i, conseqüentment, la tecnologia és diferent. Això fa que les microarquitectures dels processadors hagin de tenir en compte que la unitat de processament haurà d'estar dividida en dues parts per motius tecnològics:

- La **unitat central de processament**, o **CPU** (de les sigles en anglès), és la part del processador que es dedica a processar la informació. També s'hi inclou la unitat de control corresponent.

- La **memòria principal** és la part del processador que s'ocupa d'emmagatzemar dades i instruccions (programes). Com s'ha vist, pot estar organitzada de manera que es vegi com un bloc únic (arquitectura de Von Neumann) o com dos blocs que guarden dades i instruccions per separat (arquitectura de Hardvard).

Figura 56. Organització d'un processador amb CPU i memòria



La relació entre les CPU i les memòries té molta influència en el rendiment del processador. Generalment, les CPU inclouen un banc de registres prou gran per a contenir la majoria de les dades que es fan servir en un moment determinat, sense haver d'accedir a la memòria, amb independència de si s'implementa un RISC o un CISC. En tot cas, per gran que sigui aquest banc de registres, no pot contenir les dades necessàries per a l'execució d'un programa qualsevol i s'ha de recórrer a les memòries externes a la CPU.

Les **arquitectures *load/store*** es poden construir sobre microarquitectures optimitzades per a fer operacions simultànies d'accés a la memòria i operacions internes a la CPU. Normalment, són construccions en què es pot executar en paral·lel una instrucció *load* o *store* amb alguna altra de treball sobre registres de la CPU. En el cas contrari, amb ISA que inclouen instruccions de tota mena amb accés a memòria i operacions, és més complicat de dur a terme execucions en paral·lel, ja que les mateixes instruccions imposen una seqüència en l'execució dels accessos a la memòria i la realització de les operacions.

Amb tot, hi ha microarquitectures que separen la part de treball amb dades de la part de treball amb adreces de memòria per a poder materialitzar CPU més ràpides.

De manera similar, per a determinats processadors, també és habitual separar la memòria de dades de la d'instruccions. D'aquesta manera es pot aconseguir un grau de paral·lelisme elevat i una millora del rendiment quant a nombre d'instruccions executades per unitat de temps (de vegades, es parla de les MIPS o milions d'instruccions per segon que pot executar un determinat processador).

De totes maneres, per a un processador de propòsit general, sol ser més convenient sacrificar aquest factor de millora del rendiment per a poder executar una varietat més gran de programes.

Sigui quina sigui la microarquitectura del processador, les unitats de processament acaben tenint una velocitat de treball més elevada que la de les memòries de gran capacitat. Això passa, en part, perquè un dels factors que més pesa

a l'hora de materialitzar una memòria és, justament, la capacitat per sobre de la velocitat de treball. Ara bé, sabent que els programes presenten molta "localitat" de tota mena, és possible tenir les dades i les instruccions més utilitzades (o les que es puguin fer servir amb més probabilitat) en memòries més ràpides, encara que no tinguin tanta capacitat. Aquestes memòries són les denominades **memòries cau** (o *cache*, perquè són transparents a la unitat de processament).

Memòries cau

Les **memòries cau** es denominen així perquè són memòries que es fan servir com els caus, per a emmagatzemar-hi coses d'amagat de la vista dels altres. En la relació entre les CPU i les memòries principals es posen memòries cau, més ràpides que les principals, per a emmagatzemar-hi dades i instruccions de la memòria principal perquè la CPU, que treballa a més velocitat, hi tingui un accés més ràpid. Si no hi hagués memòries cau, la transferència d'informació es faria igualment, però a la velocitat de la memòria principal. D'això, que siguin com amagatalls o caus.

3.3.5. Processadors de propòsit general

Els **processadors de propòsit general** (o **GPP**, de l'anglès *general-purpose processor*) són processadors destinats a poder executar programes de tota mena per a aplicacions de processament d'informació. És a dir, són processadors per a ordinadors d'ús genèric, que tant poden servir per a gestionar una base de dades en una aplicació professional com per a jugar-hi.

Per aquest motiu, solen ser processadors amb un repertori d'instruccions prou ampli per a executar eficientment tant un programa que necessita fer molts càlculs com un altre que té un requeriment més elevat de moviment de dades. Tot i que aquest fet fa pensar en una arquitectura CISC, és freqüent trobar GPP construïts amb RISC perquè solen tenir cicles d'execució d'instruccions més curts que milloren el rendiment de les microarquitectures en *pipeline*. Cal tenir en compte que els *pipelines* amb moltes etapes són complexos de gestionar quan hi ha salts, per exemple.

Atesa la variabilitat de volums de dades i de grandària de programes que hi pot haver en un cas general, els GPP estan construïts seguint els principis de l'arquitectura de Von Neumann, en la qual el codi i les dades resideixen en un únic espai de memòria.

3.3.6. Processadors de propòsit específic

En el cas dels processadors destinats a algun tipus d'aplicació concret, tant el repertori d'instruccions com la microarquitectura s'ajusten per a obtenir la màxima eficiència possible en l'execució dels programes corresponents.

El repertori d'instruccions n'inclou d'específiques per al tipus d'aplicació a què es destinarà el processador. Per exemple, en pot tenir d'operacions amb vectors o amb coma flotant, si es tracta d'aplicacions que necessiten molt de processament numèric com la generació d'imatges o l'anàlisi de vídeo. Evidentment, si aquest és el cas, la microarquitectura corresponent ha de tenir en compte que la CPU haurà de fer servir diverses unitats aritmeticològiques (o d'altres elements de processament) en paral·lel.

De fet, l'exemple anterior es correspon amb els **processadors digitals de senyal** o **DSP** (de l'anglès, *digital signal processors*). Els DSP, doncs, estan orientats a aplicacions que requereixin el processament d'un flux continu de dades que es pot veure com un senyal digital que cal anar processant per a obtenir-ne un de sortida elaborat convenientment.

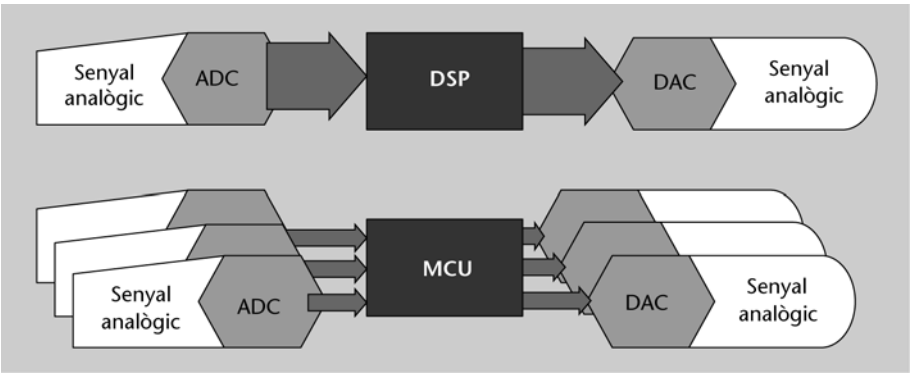
Per al cas particular de les imatges, hi ha DSP específics, denominats **unitats de processament d'imatges** o **GPU** (de l'anglès, *graphics processing units*), que tenen una ISA similar, però que solen fer servir un sistema de memòria diferent, amb les adreces organitzades en 2D per a facilitar les operacions amb les imatges.

Un cas de característiques ben diferents és el dels controladors, que han de copsar molts senyals dels sistemes que controlen i generar els senyals de control convenients. Les aplicacions de control, doncs, treballen amb molts senyals que, si bé finalment també són senyals digitals, no ocupen gaires bits i el seu processament individual no és especialment crític. Ara bé, aquestes aplicacions necessiten processadors amb un repertori d'instruccions que n'inclogui amb una varietat gran d'operacions d'entrada/sortida i una microarquitectura que les permeti dur a terme de manera eficient.

En aquests casos, els processadors actuen com a **microcontroladors** o **MCU** (de l'anglès, *microcontroller unit*). Pel fet de tenir capacitat de captar dades de l'exterior i treure'n cap a fora sense necessitat d'elements addicionals, es poden veure com a petits computadors: perifèrics i processador en un únic xip.

La dualitat entre DSP i MCU es pot resumir en el fet que els primers processen pocs senyals de molts bits molt ràpidament, i que els segons són capaços de tractar molts senyals de pocs bits gairebé simultàniament.

Figura 57. Exemplificació de la dualitat entre DSP i MCU (ADC i DAC són, respectivament, convertidors analògic-digital i digital-analògic).



En la pràctica, però, hi ha moltes aplicacions que necessiten combinar les característiques dels DSP i dels MCU, per la qual cosa molts processadors específics es denominen d'una manera o d'una altra segons les característiques dominants, però no perquè les altres no les tinguin.

A més d'implementar un repertori específic d'instruccions, hi ha dos factors que cal tenir en compte: el temps que es triga a executar-les i el consum d'energia per a fer-ho. Sovint hi ha aplicacions que són molt exigents en els dos aspectes.

Per exemple, qualsevol dispositiu mòbil amb capacitat de captar/mostrar vídeo necessita una capacitat de processament molt elevada en un temps relativament curt i sense consumir gaire energia.

El temps i el consum d'energia són factors contradictoris i les microarquitectures orientades a la rapidesa de processament solen consumir molta més energia que les que inclouen mecanismes d'estalvi energètic a canvi d'anar una mica més lents.

Per a anar ràpid, els processadors inclouen mecanismes de *pipelining* i de processament en paral·lel. A més, solen estar construïts segons l'arquitectura Hardvard, per a treballar concurrentment amb instruccions i dades. Per a consumir menys, inclouen mecanismes de gestió de la memòria interna de la CPU que eviten lectures i escriptures innecessàries a la memòria principal. En general, però, els processadors de baix consum (*low-power processors*) solen ser més senzills que els d'alt rendiment (*high-performance processors*) perquè la gestió dels recursos es complica quan n'hi ha molts.

3.4. Computadors

Els **computadors** són màquines que inclouen, al menys, un processador per a poder processar informació, en el sentit més general. Habitualment treballen de manera interactiva amb els usuaris, encara que també poden treballar independentment. La interacció dels computadors no es limita als usuaris sinó que també arriba a l'entorn o al sistema en què es troben. Així doncs, hi ha computadors que es perceben com a tals i que tenen un mode de treball interactiu amb els humans: creem, consultem i gestionem informació de tota mena, hi juguem, hi treballem i hi fem gairebé qualsevol cosa que ens passi per la imaginació. I també n'hi ha de més "amagats", que controlen aparells de tota mena o que permeten que hi hagi dispositius de tot tipus que tinguin un "comportament intel·ligent".

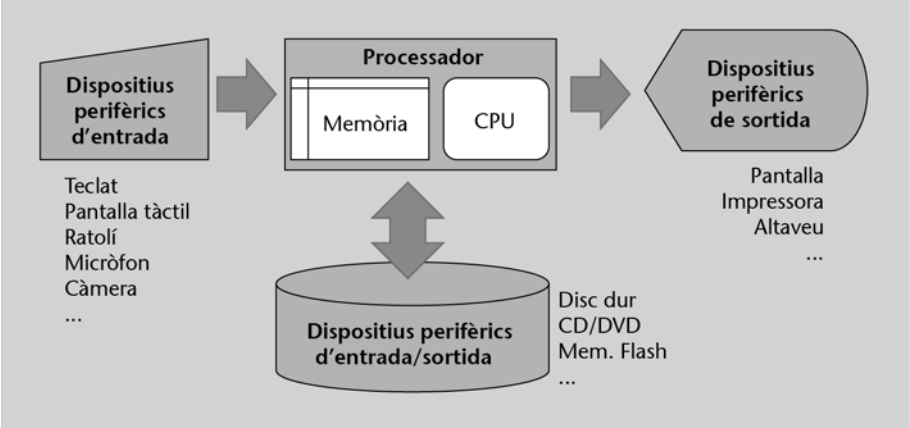
Siguin com siguin, els computadors tenen uns trets comuns que es comentaran a continuació.

3.4.1. Arquitectura bàsica

Els computadors processen informació. Per a fer-ho els cal un processador. Els processadors són els nuclis dels computadors.

Perquè els processadors puguin fer la seva feina, cal donar-los informació per a processar (i també els programes per a fer-ho) i cal dotar-los de mecanismes per a recollir la informació processada. D'aquestes tasques, se n'ocupen els denominats **dispositius perifèrics** o, simplement, **perifèrics** (i són perifèrics perquè es troben a la perifèria del nucli del computador).

Figura 58. Arquitectura general d'un computador



Els perifèrics d'entrada més habituals són el teclat i el ratolí; i els de sortida més comuns, la pantalla (o monitor) i la impressora. Alguns haureu notat que n'hi ha força més, de perifèrics: altaveus, escàners (per a capturar imatges impreses), micròfons, càmeres, televisors, i un reguitzell addicional que s'amplia contínuament per a ajustar-se a les aplicacions informàtiques que pot portar a terme un computador.

També hi ha perifèrics que poden fer totes dues tasques. És a dir, es poden ocupar de l'entrada de les dades a l'ordinador i de la sortida de les dades resultants. Els dispositius perifèrics d'entrada/sortida més visibles són les unitats (els elements de l'ordinador) de discos òptics (CD, DVD, Blu-ray, etc.) i de targetes de memòria. Altres perifèrics d'aquesta mena són els discos durs, que estan a dins la caixa de l'ordinador, i els mòduls de connexió a xarxa sense fil.

Generalment, aquest tipus de perifèrics s'utilitzen per a emmagatzemar dades que es poden recuperar quan un determinat procés ho sol·liciti i, igualment, modificar-les si ho requereix. El cas dels mòduls de connexió a xarxa resulta una mica especial, atès que són uns perifèrics que no emmagatzemen informació però que permeten obtenir-ne i enviar-ne per la xarxa.

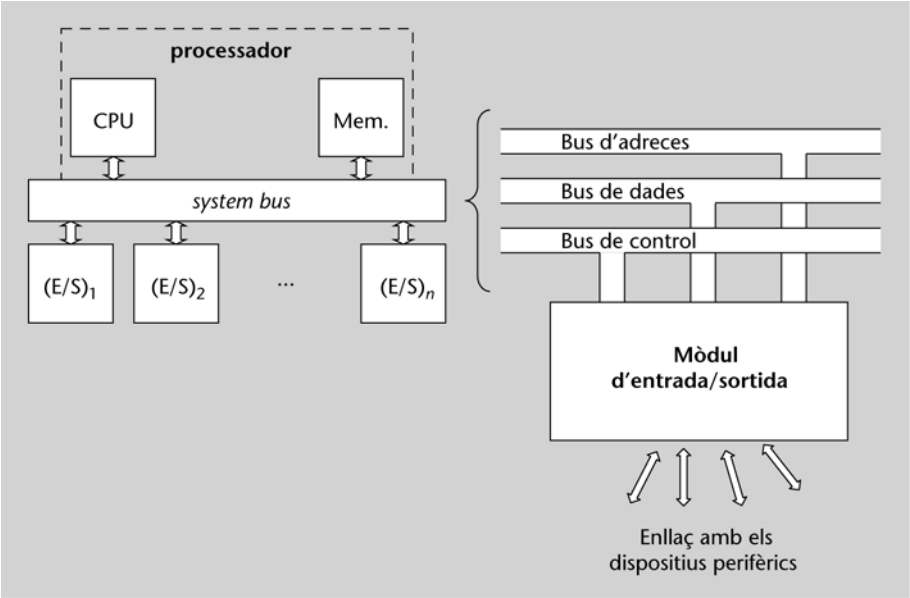
Per a comunicar-se amb el processador, els dispositius perifèrics disposen de controladors que, a més de governar-ne el funcionament, es relacionen amb un **mòdul d'entrada/sortida** del computador que sigui. Aquests mòduls fan de pont entre el perifèric i el processador, tot adaptant les diferents maneres de funcionar, els formats de les dades i la velocitat de treball. Per exemple, les lectures de disc es fan per blocs de diversos KB que es processen per a eliminar-ne els errors i organitzar-los convenientment per a poder ser transferits al pro-

cessador en unitats (de pocs bytes) i velocitats de transferència ajustades als seus busos.

A mode d'il·lustració, val a dir que la CommUnit del YASP fa les funcions de mòdul d'entrada/sortida del processador. Qualsevol perifèric que s'hi connecti rep i envia senyals d'aquest mòdul que, al seu torn, es relaciona tant amb la unitat de control com amb el registre A, si es tracta d'una transferència de dades. Evidentment, un processador més complex necessita més mòduls d'E/S, amb funcionalitats específiques per a cada tipus de perifèric, que són, també, més complexos.

Per tant, una arquitectura bàsica d'un computador hauria de tenir diversos mòduls d'entrada/sortida connectats a un bus del sistema, que també hi tindria connectats la CPU i la memòria principal, tal com es mostra en la figura 59.

Figura 59. Esquema d'un computador basat en un únic bus



El bus del sistema s'organitza en tres busos diferents:

1) El bus de dades es fa servir per a transportar-hi dades entre tots els elements que connecta. Freqüentment, les dades van de la memòria principal a la CPU, però també al revés. Les transferències entre CPU i mòduls d'E/S o entre mòduls d'E/S i memòria són relativament més esporàdiques.

2) El bus d'adreces transmet les adreces de les dades (i de les instruccions) a les quals accedeix la CPU a la memòria o a algun mòdul d'E/S. També l'aprofiten els mòduls d'E/S per accedir directament a dades en memòria.

3) El bus de control transmet tots els senyals de control perquè tots els elements es puguin comunicar correctament. Des de la CPU és la unitat de control l'encarregada de rebre'n els senyals que li toquen i emetre-hi els corresponents a cada estat en què es trobi. És important tenir present que, de fet, cada mòdul connectat al bus té la seva pròpia unitat de control.

L'accés als perifèrics des del processador es fa per mitjà de mòduls d'entrada/sortida específics. En una arquitectura bàsica, però perfectament funcional, els mòduls d'entrada/sortida es poden relacionar directament amb la CPU ocupant una part de l'espai d'adreces de la memòria. És a dir, que hi ha posicions de la memòria del sistema que no es corresponen amb dades emmagatzemades en la memòria principal sinó que són posicions de les memòries dels mòduls d'E/S. De fet, aquests mòduls tenen una memòria interna en la qual poden recollir dades per al perifèric o en la qual poden emmagatzemar temporalment les que en provenen.

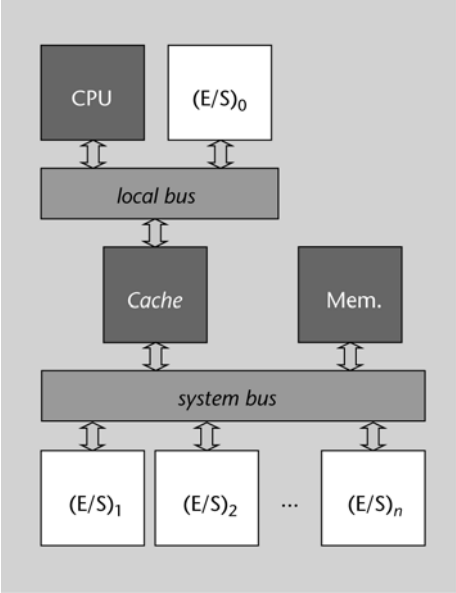
Les operacions d'entrada/sortida de dades programades implicarien sincronitzar el processador amb el perifèric corresponent. És a dir que, per a transmetre les dades, el processador hauria d'esperar que el perifèric estigués disponible, cosa que significaria una pèrdua notable de rendiment en l'execució de l'aplicació associada. A tall d'exemple il·lustratiu, cal tenir en compte que un disc pot transferir dades a velocitats comparables (en bytes/segon) a les velocitats a què pot treballar una CPU o la memòria principal, però la part mecànica fa que l'accés a les dades pugui trigar uns quants millisegons, temps que perdria el processador.

En general, la transferència de dades entre perifèrics i el processador es fa per mitjà de mòduls amb **mecanismes d'accés directe a la memòria** (**DMA**, de l'anglès *direct memory access*). Els mòduls dotats de DMA transfereixen dades entre la seva memòria interna i la principal del computador sense intervenció de la CPU. D'aquesta manera la CPU queda "alliberada" per a executar instruccions del programa en curs. En la majoria dels casos, els mòduls amb DMA i la CPU es comuniquen per establir quin té accés a memòria per mitjà del bus del sistema, amb prioritat per a la CPU, òbviament.

Com que la memòria principal encara resulta relativament lenta quant a la CPU, és habitual interposar-hi memòria cau. Com ja s'ha comentat, aquesta memòria resulta transparent a la CPU i a la memòria principal: el controlador del mòdul de la memòria cau s'ocupa de "capturar" les peticions de lectura i escriptura de la CPU i de respondre com si fos la memòria principal, però més ràpidament. En la figura 60, es mostra l'organització d'un computador amb un bus local per a comunicar la CPU amb la memòria cau i un mòdul d'entrada/sortida per a les comunicacions amb els perifèrics que no es fan per mitjà de memòria. El bus local treballa a més velocitat que no pas el de sistema, més d'acord amb les freqüències d'operació de la CPU i de la memòria cau.

Aquesta configuració bàsica d'un computador es pot millorar incorporant-hi més recursos. En el subapartat següent es comentaran breument algunes ampliacions possibles, segons l'aplicació a què es destinin els computadors corresponents.

Figura 60. Esquema d'un computador amb memòria cau



3.4.2. Arquitectures orientades a aplicacions específiques

Les arquitectures genèriques es poden estendre amb recursos que incideixin en la millora d'algun aspecte del rendiment d'una aplicació específica: si necessita molta capacitat de memòria, se les dota amb més recursos de memòria, i si necessita molta capacitat de processament, se les dota amb més recursos de càlcul.

La capacitat de processament es pot augmentar amb coprocessadors especialitzats o multiplicant el nombre de processadors del computador.

Per exemple, molts ordinadors inclouen GPU com a coprocessadors que alliberen les CPU del treball de tractament d'imatges. En aquest cas, cada unitat (la CPU i la GPU) treballa amb la seva pròpia memòria. En el cas de múltiples processadors, la memòria principal sol ser compartida, però cada CPU té la seva pròpia memòria cau.

L'augment de la capacitat de la memòria es pot fer amb la contrapartida de mantenir (o augmentar) la diferència de velocitats de treball de memòria principal i CPU. La solució passa per interposar-hi més d'un nivell de memòries cau: les més ràpides i petites a prop de la CPU, i les més lentes i grans, de la memòria principal.

Les necessitats creixents de processament i memòria en tota mena d'aplicacions han fet que les arquitectures originalment concebudes per a computadors d'alt rendiment siguin cada cop més comunes en qualsevol ordinador.

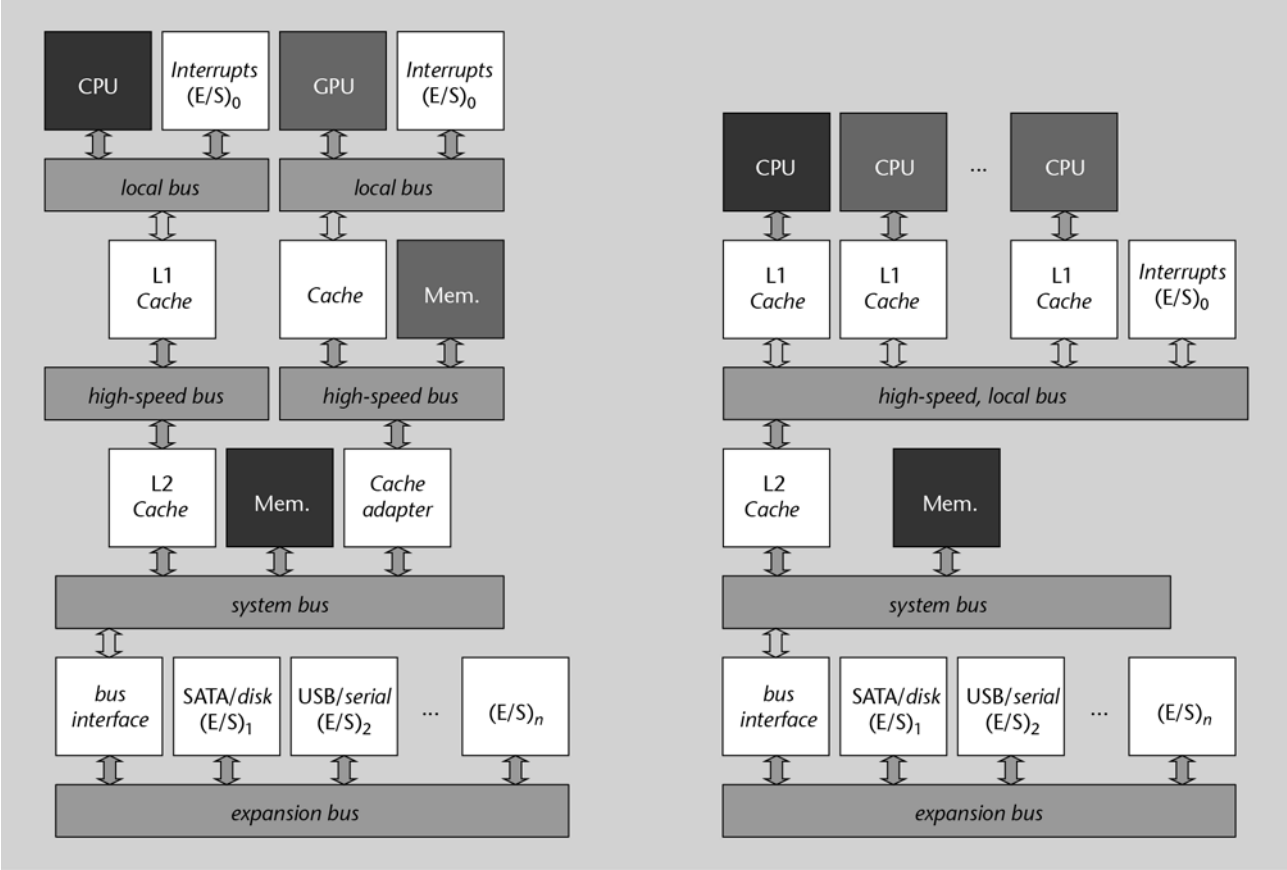
Computador d'alt rendiment

Un **computador d'alt rendiment** és el capaç d'executar, en un mateix temps, programes amb més memòria i instruccions que els normals, entenent per normals els que són majoria en un moment determinat.

En la figura 61 es mostren dues arquitectures específiques per a aplicacions gràfiques i multiprocés. Totes dues fan servir dos nivells de memòria cau, *cache* L1 i L2, i organitzen el sistema en una jerarquia de busos. En la dedicada a aplicacions gràfiques, hi ha un coprocessador específic (una GPU) que té una memòria principal local i una memòria cau pròpies. L'opció multiprocessadora es

basa en diverses CPU que tenen una memòria cau local, però que en comparteixen una de general per a la memòria principal.

Figura 61. Arquitectures de computadors amb GPU (esquerra) i múltiples CPU (dreta)



Aquestes dues configuracions es poden combinar per a obtenir computadors de molt alt rendiment per a tota mena d'aplicacions i, especialment, per a les intensives en càlcul.

L'evolució tecnològica fa possible integrar cada cop més components en un únic xip, cosa que aporta l'avantatge de posar més recursos per a construir els computadors, però que també en complica cada cop més l'arquitectura.

Per exemple, hi ha xips *multicore* que inclouen diversos nivells de memòria cau i també coprocessadors específics, de tipus GPU.

Resum

Els computadors són circuits seqüencials complexos, molt complexos. S'ha vist que estan organitzats en diversos mòduls més petits que interactuen per aconseguir executar programes descrits en llenguatges d'alt nivell de la manera més eficient possible.

Un computador es pot despullar dels seus dispositius perifèrics i continuar mantenint l'essència que el defineix: la capacitat de processar informació de manera automàtica seguint un programa. Els processadors, però, també poden estar organitzats de maneres molt diferents encara que, per mantenir la capacitat d'executar qualsevol programa, sempre treballen executant una seqüència d'instruccions emmagatzemada en la memòria. De fet, aquesta organització de la màquina es denomina **arquitectura de Von Neumann**, en honor al model establert per a un computador descrit per aquest autor.

S'ha vist que els processadors es poden materialitzar a partir de descripcions algorísmiques de màquines que interpreten les instruccions d'un determinat repertori, i també que això es pot fer tant per a construir unitats de control sofisticades com per a materialitzar processadors senzills.

Les màquines algorísmiques també es poden materialitzar exclusivament com a circuits seqüencials. En aquest sentit, se n'ha tractat un cas particular a mode d'exemple. En el fons, aquestes màquines són màquines d'estats algorísmiques focalitzades més en els càlculs que en les combinacions de senyals d'entrada i de sortida. S'ha vist que les ASM són útils quan les transicions entre estats depenen de pocs senyals binaris (externs o interns) i que resulten efectives per a implementar controladors de tota mena, en aquestes circumstàncies.

Amb tot, el problema de l'augment del nombre d'estats quan hi ha seqüències d'accions (i/o càlculs) en una màquina també es resol bé amb una PSM. En aquestes màquines, alguns estats poden representar l'execució d'un programa complet i, consegüentment, tenen menys estats que una màquina d'estats amb els programes desplegats com a seqüències d'estats. A canvi, han de fer servir una variable específica, que es denomina **comptador de programa**.

En general, però, s'ha vist que totes aquestes màquines es poden materialitzar d'acord amb una arquitectura de màquina d'estats amb camí de dades (FSMD) i que, de fet, són casos concrets de màquines d'estats finits esteses.

Les EFSM són un tipus de màquines que inclouen la part de control i de processament de dades en la mateixa representació, cosa que en facilita tant l'anàlisi com la síntesi.

S'ha tractat un cas paradigmàtic de les EFSM, el comptador. El comptador fa servir una variable per a emmagatzemar el compte i, amb això, discrimina els estats de la màquina de control principal de l'estat general de la màquina, que també inclou el contingut de totes les variables que conté.

Prèviament, s'ha vist que les màquines d'estats finits o FSM serveixen per a modelar controladors de tota mena, tant de sistemes externs com de la part de processament de dades del circuit.

En resum, doncs, s'ha passat d'un model d'FSM amb entrades i sortides d'1 bit que servia per a controlar 'coses' a models més complets que, a l'hora que permetien representar comportaments més complexos, també permetien expressar algorismes d'interpretació de programes. S'han mostrat exemples de materialització de tots els casos per a facilitar la comprensió del funcionament dels circuits seqüencials més senzills i també la dels que són més complexos del que pot incloure aquest material: els que constitueixen les màquines que anomenem **computadors**.

Exercicis d'autoavaluació

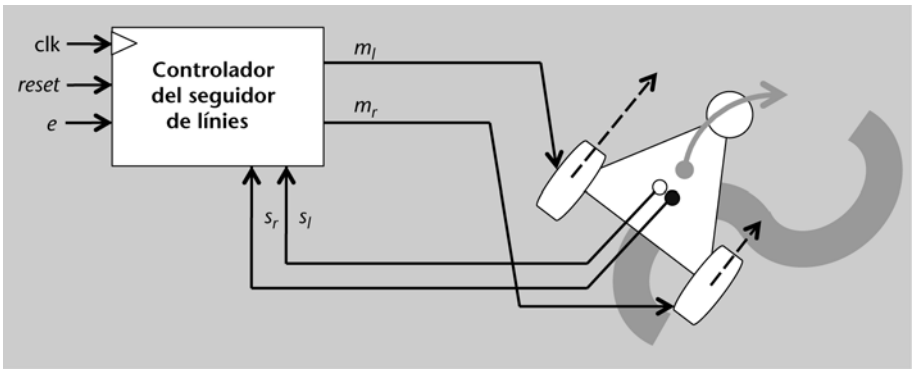
Els problemes que es proposen a continuació s'haurien de resoldre havent acabat l'estudi del mòdul, de manera que serveixin per a comprovar el grau d'assoliment dels objectius indicats al principi. Amb tot, els quatre primers es poden aprofitar com a preparació d'una possible pràctica de l'assignatura i, per aquest motiu, seria útil resoldre'ls en paral·lel a l'estudi de la darrera part del mòdul (apartat 3).

1. Aquest exercici ha de servir per a reforçar la visió de les FSM com a representacions del comportament de controladors. Per a això es demana que dissenyeu el controlador d'un robot seguidor de línies. El vehicle del robot es mou per parell diferencial (figura 17), tal com s'explica en l'apartat 1.3, i està dotat d'un parell de sensors a la part de sota, que serveixen per a detectar si el robot és a sobre de la línia de manera total o parcial. Les entrades del controlador són, doncs, les provinents d'aquests sensors, s_l i s_r, per sensor esquerre i dret respectivament. Hi ha una entrada addicional d'activació i aturada, e, que ha d'estar a 1 perquè el robot segueixi la línia. Les sortides són les que controlen els motors esquerre i dret per fer girar les rodes corresponents: m_l i m_r. El funcionament és el que es mostra en la taula següent:

m_l	m_r	Efecte
0	0	Robot aturat
0	1	Gir a l'esquerra
1	0	Gir a la dreta
1	1	Avançament recte endavant

Els girs s'han de fer quan, en seguir una línia, un dels dos sensors no la detecti. Per exemple, si el sensor esquerre la deixa de detectar, s'ha de girar cap a la dreta.

Figura 62. Esquema de blocs del robot seguidor de línies



Se suposa que, inicialment, el robot estarà localitzat sobre una línia i que, en cas que no en detecti per cap dels sensors, s'aturarà. Amb aquesta informació, s'ha de fer el diagrama de l'FSM corresponent (no cal fer el disseny del circuit que l'implementi).

2. En aquest problema es repassa el fet que les EFSM són FSM amb operands (habitualment, nombres) i operadors de múltiples bits i que es construeixen amb una arquitectura d'FSMD. Heu de construir l'EFSM d'un detector de sentit d'avançament d'un eix i implementar-ne el circuit corresponent.

Aquest detector fa servir el que es denomina un **codificador de rotació**, que és un dispositiu que converteix la posició angular d'un eix a un codi binari. El detector a construir ha de tenir, com a entrada, un nombre natural de 3 bits, $A = (a_2, a_1, a_0)$, que indicarà la posició angular absoluta de l'eix, i, com a sortida, 2 bits, $S = (s_1, s_0)$, que indicaran si l'eix gira en sentit de les agulles del rellotge, $S = (0, 1)$, o si ho fa en sentit contrari, $S = (1, 0)$. Si l'eix està aturat o no canvia de sector, la sortida serà (0, 0). Un sector queda definit per un rang de posicions angulars que tenen el mateix codi.

Tot i que no es representi en l'EFSM del detector, la materialització del circuit corresponent ha de tenir una entrada de *reset* que forci la transició cap a l'estat inicial.

L'esquema de la figura 63 presenta el sistema que cal dissenyar. El detector fa servir un codificador de rotació que proporciona la posició de l'angle en format de nombre natural de 3 bits segons la codificació que es mostra en la taula següent:

Sector	s_2	s_1	s_0
[0º, 45º)	0	0	0
[45º, 90º)	0	0	1
[90º, 135º)	0	1	0
[135º, 180º)	0	1	1
[180º, 225º)	1	0	0
[225º, 270º)	1	0	1
[270º, 315º)	1	1	0
[315º, 360º)	1	1	1

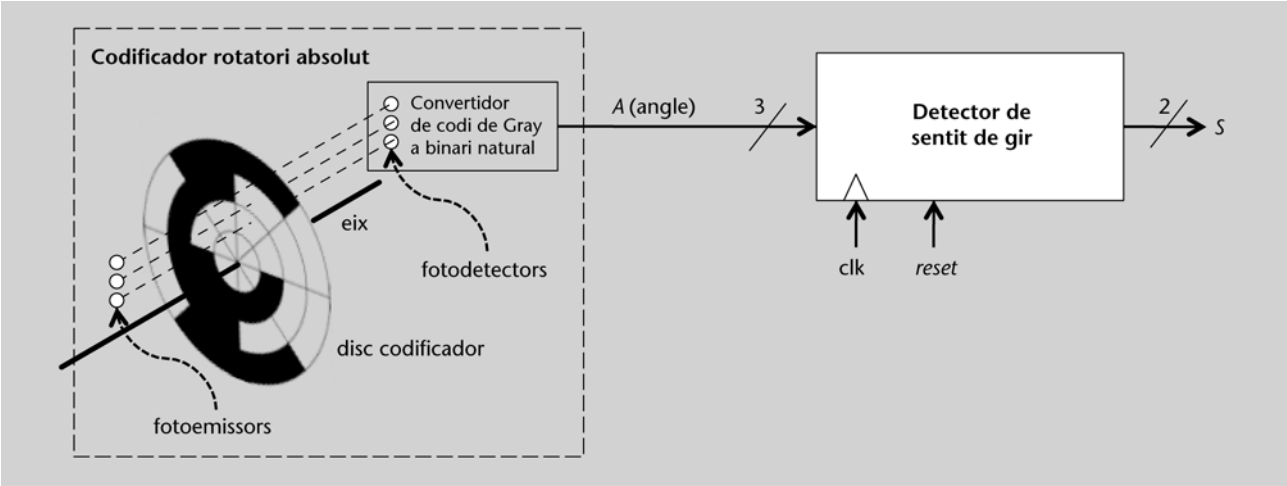
Codis de Gray

Els codis de Gray es fan servir en aquest tipus de codificadors rotatoris perquè entre dos codis consecutius només canvia un bit, cosa que fa que el canvi de sector es detecti amb aquest canvi. Entre el codi del sector inicial i el del final d'aquest canvi poden detectar-se, transitòriament, canvis a més d'un bit. En aquest cas, es descarten els codis intermedis.

Cal tenir present que del sector [315º, 360º) es passa directament al sector [0º, 45º), en girar en el sentit de les agulles del rellotge.

El codificador rotatori que es presenta funciona amb un disc amb tres pistes (una per bit) que està dividit en vuit sectors. A cada sector correspon un codi de Gray de 3 bits que es llegeix amb tres fotodetectors i que, posteriorment, es converteix a un nombre binari natural que identifica el sector tal com s'ha presentat en la taula anterior.

Figura 63. Detector del sentit del gir amb un codificador rotatori absolut



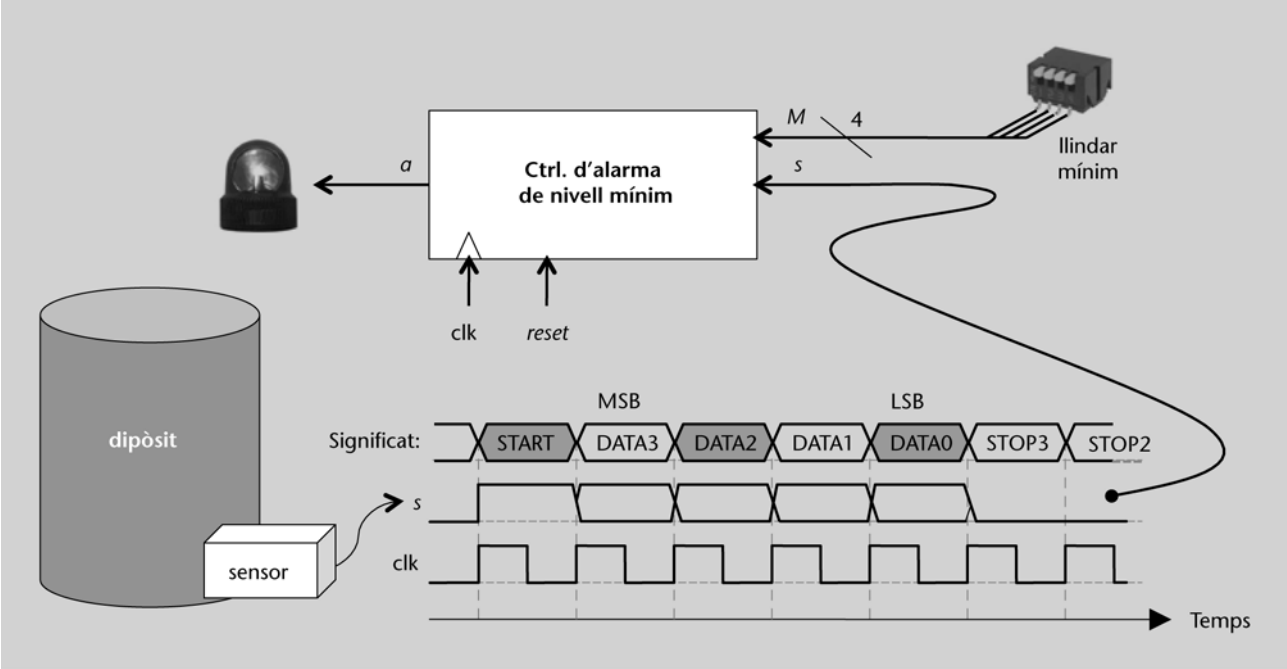
Per a resoldre aquest problema cal tenir present que el detector del sentit del gir ha de comparar el sector actual amb l'anterior. Se suposa que la cadència de les lectures dels angles *A* és prou elevada perquè no hi pugui haver salts de dos o més sectors entre dues lectures consecutives.

El circuit corresponent es pot dissenyar fent servir els recursos que us convinguin dels que heu vist fins ara.

3. Amb vista a veure la utilitat de les PSM a l'hora de representar comportaments que inclouen seqüències d'accions heu de fer el model d'un controlador d'una alarma de nivell mínim d'un dipòsit.

El controlador ha d'enviar, pel senyal de sortida *a*, un 1 durant un cicle de rellotge a una alarma per a activar-la en cas que el nivell del dipòsit sigui igual o més baix que una referència, *M*, donada. La referència s'estableix amb uns petits commutadors. El nivell del líquid en el dipòsit li proporciona un sensor en forma de nombre binari natural de 4 bits. Aquest nombre indica el percentatge d'ompliment del dipòsit, des de 0% ($0000_{(2)}$) fins al 100% ($1111_{(2)}$) i, per tant, cada unitat equival al 6,25%. A fi de reduir cables, les dades del sensor s'envien en sèries de bits. Tal com es mostra en la figura 64, primer s'envia 1 bit a 1 i llavors els 4 bits que conformen el nombre que representa el percentatge d'ompliment del dipòsit. Els bits d'aquest nombre s'envien començant pel més significatiu i acabant pel menys significatiu. Sempre hi ha d'haver, com a mínim, 4 bits a 0 seguint el darrer bit del nombre en una transmissió.

Figura 64. Controlador d'alarma de nivell mínim



Dissenyeu la PSM que representa el comportament d'aquest controlador, tenint present que l'alarma s'ha de mantenir activada sempre que es compleixi que la lectura del sensor és inferior o igual al llindar mínim establert i que el sensor envia dades cada cop que detecta algun canvi significatiu en el nivell.

4. Aquest problema tracta de l'ús d'ASM en els casos en què hi ha un nombre abundós d'entrades, de manera que la representació del comportament sigui més compacte i entenedora: cal fer el model de control d'un forn de microones.

El forn té una sèrie de sensors que li proporcionen informació per mitjà dels senyals que s'enumeren a continuació:

- *s* indica que s'ha premut el botó d'inici/aturada amb un pols a 1 durant un cicle de rellotge;
- *d* és 0 si la porta és oberta o 1 si és tancada;
- *W* és un nombre relacionat amb la potència de treball del microones, que va dels valors 1 a 4, codificats de 0 a 3, i
- *t* és un senyal que es manté a 1 mentre el temporitzador és en marxa; la roda del temporitzador es programa girant-la en el sentit de les agulles del rellotge i, a mesura que passa el temps, gira en sentit contrari fins a la posició inicial. El senyal *t* només és 0 quan la roda del temporitzador és en la posició 0.

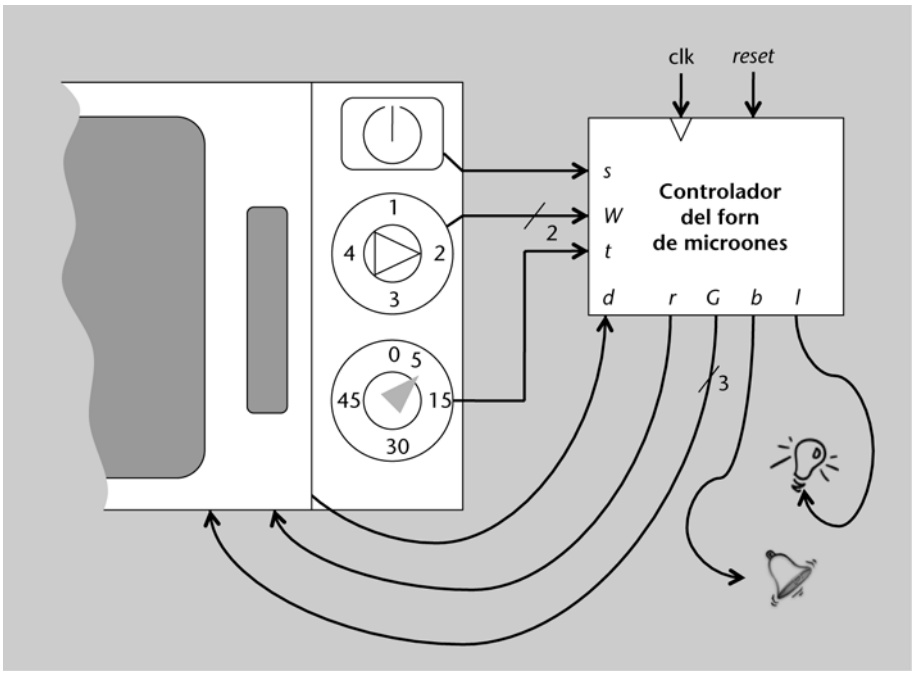
Cal tenir present que el botó d'inici/aturada es pot prémer en qualsevol moment i que la roda del temporitzador només pot retornar a la posició 0 quan hagi passat el temps corresponent o si l'usuari en força el retorn girant-la fins a aquesta posició. Si es prem per a engegar el forn amb la porta oberta o sense el temporitzador activat, no té efecte i el forn continua apagat. El forn s'ha d'apagar sempre que s'obri la porta o es premi el botó d'inici/aturada.

El controlador pren la informació proporcionada per aquests sensors i fa les actuacions corresponents, segons el cas, mitjançant els senyals següents:

- *b* s'ha de posar a 1 durant un cicle de rellotge per a fer sonar un timbre d'avís d'acabament del període de temps marcat pel temporitzador;
- *l* controla el llum de l'interior, que només és encès si és 1 (com a mínim, durant el funcionament del forn);
- *r* ha de ser 1 per a fer girar el plat interior, i
- *G* és un nombre natural que governa el funcionament del generador de microones, essent 0 el valor per a apagar-lo, i 4 per a funcionar a potència màxima.

Amb tot, l'esquema del conjunt es pot veure en la figura 65.

Figura 65. Esquema de blocs d'un forn de microones



Es tracta, doncs, que dissenyeu l'ASM corresponent a un possible controlador del forn de microones que s'ha descrit.

5. En molts casos, els controladors han de fer càlculs complexos que cal tractar independentment amb esquemes de càlcul. En aquest exercici heu de dissenyar un d'aquests esquemes per a calcular el resultat de l'expressió següent:

$$ax^2 + bx + c$$

L'esquema s'ha d'organitzar per a fer servir els mínims recursos possibles. Per simplicitat, totes les dades seran del mateix format, tant les d'entrada com les intermèdies i de sortida.

6. Els diagrames de flux serveixen per a representar algorismes per a fer càlculs més complexos que poden incloure esquemes com els anteriors. En aquest exercici es tracta de veure com es pot transformar un diagrama de flux en un circuit amb arquitectura d'FSMD. Així, heu d'implementar el càlcul de l'arrel quadrada entera.

L'arrel quadrada entera d'un nombre c és el valor enter a que compleix les desigualtats següents:

$$a^2 \leq c < (a + 1)^2$$

És a dir, l'arrel quadrada entera d'un nombre c és el valor enter més proper per l'esquerra a la seva arrel quadrada real.

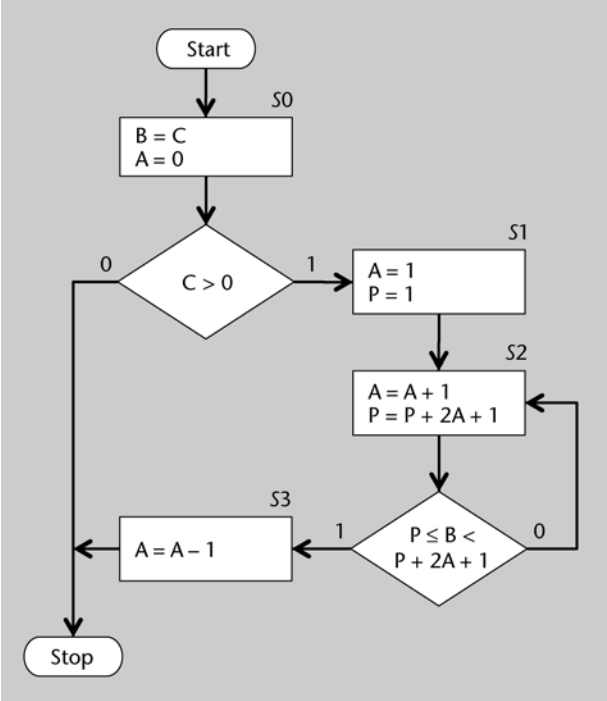
Per a calcular a es pot seguir un procediment iteratiu a partir d'$a = 1$, incrementant gradualment a fins que compleixi la condició que s'ha mencionat anteriorment.

En el diagrama de flux de la figura 66 es pot veure l'algorisme corresponent, amb les variables en majúscula, atès que són vectors de bits. S'hi inclou una comprovació que c no sigui 0 perquè en aquest cas el procediment iteratiu fallaria. Per a simplificar els càlculs dels quadrats, s'emmagatzemen en una variable P que s'actualitza tenint present que $(A + 1)^2 = A^2 + 2A + 1$. És a dir, que el valor següent de la variable P es calcula com $P + 2A + 1$.

A la vista del diagrama, expliqueu per què és necessari l'estat $S3$.

Dissenyeu el circuit corresponent, seguint l'arquitectura d'FSMD, amb la unitat de control segons el model vist que reprodueix el flux del diagrama.

Figura 66. Algorisme per a calcular l'arrel quadrada entera



7. Es tracta de veure com es poden construir processadors a partir d'altres ampliant-ne el repertori d'instruccions. En aquest cas, heu de modificar el diagrama de flux de la figura 48, que representa el comportament del Femtoproc, de manera que pugui treballar amb més dades que les que es poden emmagatzemar en els sis registres lliures que té. Per a això, cal tenir present que el nou repertori té un format d'instruccions de 9 bits, tal com es mostra en la taula següent:

Instrucció	**Bits**								
	8	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ADD	0	0	0	*operand*$_1$			*operand*$_0$		
AND	0	0	1	*operand*$_1$			*operand*$_0$		
NOT	0	1	0	*operand*$_1$			*operand*$_0$		
JZ	0	1	1	adreça de salt					
LOAD	1	0	adreça de dades				*operand*$_0$		
STORE	1	1	adreça de dades				*operand*$_0$		

De fet, si el bit més significatiu és 0, se segueix el format anterior. Si és 1, s'hi introdueixen les instruccions de LOAD i STORE, que permeten llegir una dada de la memòria de dades o escriure-la-hi, respectivament. En aquests casos, l'*operand*$_0$ identifica el registre (Rf) del banc de registres que es farà servir per a desar-hi la dada o per a obtenir-la. L'adreça de la memòria de dades que es farà servir s'especifica en els bits intermedis (*Adr*) i, com que està formada per 4 bits, la memòria de dades serà de $2^4 = 16$ paraules. La memòria de dades és una memòria RAM tal com s'ha presentat en el mòdul anterior, amb una entrada per a les adreces (*M@*) de 4 bits i una altra per a indicar quina és l'operació que s'hi ha de fer (*L'/E*) i amb una entrada/sortida de dades (*Md*). Se suposa que la memòria fa l'operació en un cicle de rellotge.

Cal doncs, que amplieu el diagrama de flux de la figura 48 per representar el funcionament d'una màquina algorítmica capaç de processar programes descrits amb el repertori d'instruccions anterior. Cal recordar que, en aquesta figura, *Q* és l'entrada de la memòria de programa que conté la instrucció a executar.

8. Indiqueu quines modificacions caldrien per a adaptar la microarquitectura del YASP (figura 54) a un repertori d'instruccions capaç de treballar amb adreces de 16 bits i, així, poder disposar d'una memòria de fins a 64 KB. En aquest cas, el format de les instruccions que tenen un camp d'adreces ocuparia 3 bytes: un per al codi d'operació i dos per a les adreces. Això inclou la de salt incondicional i exclou les instruccions amb mode d'adreçament immediat. Què s'hauria de tocar en el camí de dades? Quines implicacions tindria en la unitat de control?

Solucionari

Activitats

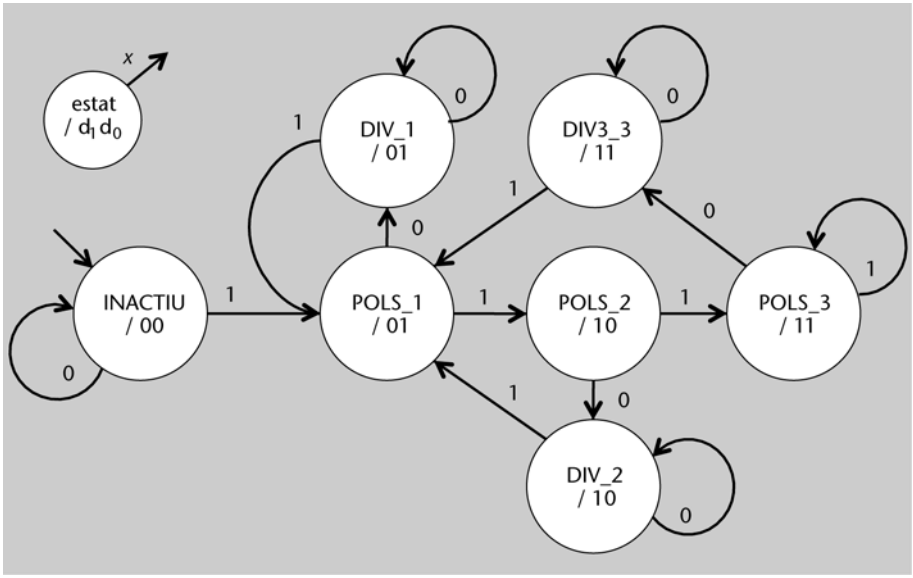
1. Per a fer el graf d'estats, cal començar per l'estat inicial (INACTIU) i decidir els estats següents segons totes les possibilitats de combinacions d'entrades. En aquest cas, només poden ser o $x = 0$ o $x = 1$. Mentre no es detecti cap canvi en l'entrada ($x = 0$) la màquina restarà inactiva. En l'instant en què arribi un 1, cal passar a un nou estat per a recordar que s'ha iniciat un pols (POLS_1). A partir d'aquest moment, es faran transicions cap a d'altres estats (POLS_2 i POLS_3) mentre $x = 1$. D'aquesta manera, la màquina descobreix si l'amplada del pols és d'un, dos, tres o més cicles de rellotge.

En el darrer d'aquests estats (POLS_3), s'hi ha de mantenir fins que no acabi el pols d'entrada, tingui l'amplada que tingui. En altres paraules, un cop x hagi estat 1 durant tres cicles de rellotge, ja no es distingirà si l'amplada del pols és de tres o més cicles de rellotge.

La màquina ha de recordar l'amplada del darrer pols rebut, de manera que la sortida D es mantingui a 01, 10 o 11, segons el cas. Per a això, en els estats de detecció d'amplada de pols (POLS_1, POLS_2 i POLS_3), quan es rep un 0 de finalització de pols ($x = 0$) es passa a un estat de manteniment de la sortida que recorda quin és el factor de divisió de la freqüència del rellotge: DIV_1, DIV_2 i DIV_3.

Des de qualsevol d'aquests estats s'ha de passar a POLS_1 quan es rep, de nou, un altre 1 a l'entrada.

Figura 67. Graf d'estats del detector de velocitat de transmissió



Del graf d'estats de la figura anterior es pot deduir la taula de transicions següent:

Estat actual				Entrada	Estat següent			
Identificació	q_2	q_1	q_0	x	Identificació	q_2^+	q_1^+	q_0^+
INACTIU	0	0	0	0	INACTIU	0	0	0
INACTIU	0	0	0	1	POLS_1	0	0	1
POLS_1	0	0	1	0	DIV_1	1	0	1
POLS_1	0	0	1	1	POLS_2	0	1	0
POLS_2	0	1	0	0	DIV_2	1	1	0
POLS_2	0	1	0	1	POLS_3	0	1	1
POLS_3	0	1	1	0	DIV_3	1	1	1
POLS_3	0	1	1	1	POLS_3	0	1	1
DIV_1	1	0	1	0	DIV_1	1	0	1
DIV_1	1	0	1	1	POLS_1	0	0	1

Estat actual				Entrada	Estat següent			
Identificació	q_2	q_1	q_0	x	Identificació	q_2^+	q_1^+	q_0^+
DIV_2	1	1	0	0	DIV_2	1	1	0
DIV_2	1	1	0	1	POLS_1	0	0	1
DIV_3	1	1	1	0	DIV_3	1	1	1
DIV_3	1	1	1	1	POLS_1	0	0	1

En aquest cas, és convenient fer una codificació dels estats que respecti el fet que l'estat inicial sigui el 000, però que la sortida es pugui obtenir directament de la codificació de l'estat (en l'exercici, això no es demana i, per tant, només s'ha de prendre com a exemple).

La taula de veritat per a les sortides és la següent:

Estat actual				Sortida
Identificació	q_2	q_1	q_0	D
INACTIU	0	0	0	00
POLS_1	0	0	1	01
POLS_2	0	1	0	10
POLS_3	0	1	1	11
DIV_1	1	0	1	01
DIV_2	1	1	0	10
DIV_3	1	1	1	11

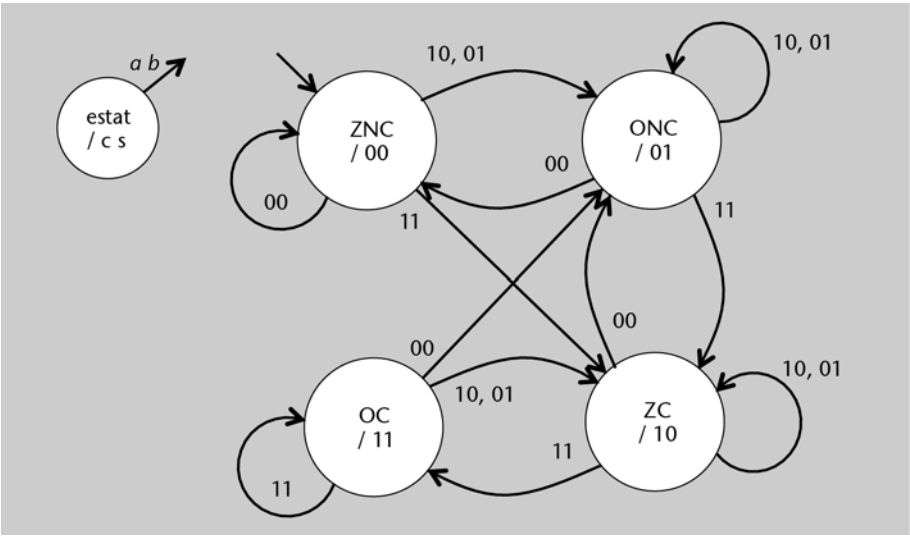
2. L'estat d'un sumador, en un moment determinat, el fixa tant el resultat de la suma com del ròssec de sortida del període de rellotge anterior, que és el que cal recordar. Per tant, la màquina d'estats corresponents comença en un estat en què se suposa que la suma prèvia ha estat (c, s) = 0 + 0, en què + indica la suma aritmètica dels 2 bits.

En l'estat inicial (ZNC, de *zero and no carry*) s'hi queda mentre els bits a sumar siguin (0, 0). Si un dels dos és a 1 i l'altre no, llavors la suma dóna 1 i el bit de ròssec 0, per tant, cal passar a l'estat corresponent (ONC, de *one and no carry*). El cas que falta és (1, 1), que genera ròssec per a la suma següent, però el resultat de la suma és 0. Per tant, de ZNC passa a C (*zero and carry*).

Una reflexió similar es pot fer estant a ONC, a ZC, i a OC (*one and carry*). Cal tenir present que, un cop feta, s'ha de comprovar que s'han previst tots els casos d'entrada.

En la figura 68 hi ha el graf d'estats corresponent. Les entrades (a, b) s'hi mostren de manera compacte amb els 2 bits consecutius.

Figura 68. Graf d'estats del sumador en sèrie



Del graf d'estats de la figura 68 es pot deduir la taula de transicions següent:

Estat actual			Entrada		Estat següent		
Identificació	q_1	q_0	a	b	**Identificació**	q_1^+	q_0^+
ZNC	0	0	0	0	ZNC	0	0
ZNC	0	0	0	1	ONC	0	1
ZNC	0	0	1	0	ONC	0	1
ZNC	0	0	1	1	ZC	1	0
ONC	0	1	0	0	ZNC	0	0
ONC	0	1	0	1	ONC	0	1
ONC	0	1	1	0	ONC	0	1
ONC	0	1	1	1	ZC	1	0
ZC	1	0	0	0	ONC	0	1
ZC	1	0	0	1	ZC	1	0
ZC	1	0	1	0	ZC	1	0
ZC	1	0	1	1	OC	1	1
OC	1	1	0	0	ONC	0	1
OC	1	1	0	1	ZC	1	0
OC	1	1	1	0	ZC	1	0
OC	1	1	1	1	OC	1	1

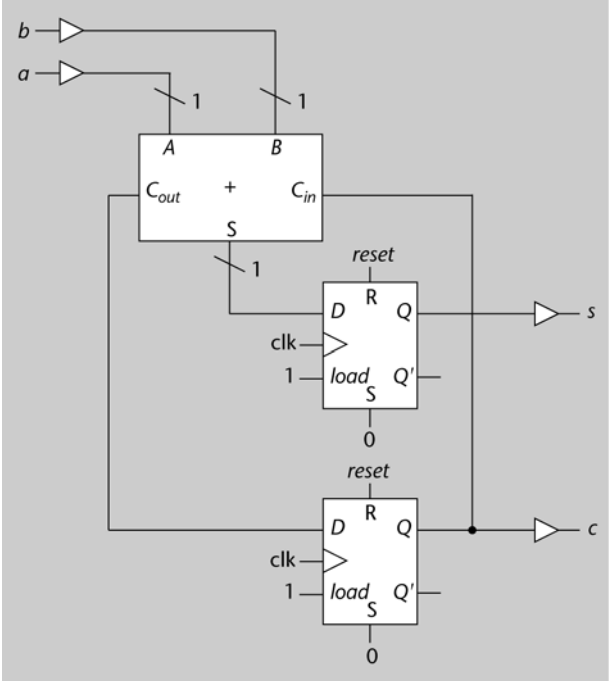
En aquest cas, la codificació dels estats s'ha fet respectant les sortides del sumador en sèrie. Per tant, $c = q_1$ i $s = q_0$, com es pot comprovar en la taula anterior.

Les funcions d'excitació són, de fet, coincidents amb les d'un sumador complet:

$$q^+_1 = q_1\,(a \oplus b) + ab$$
$$q^+_0 = (a \oplus b) \oplus q_1$$

La implementació, que no es demana en l'exercici, es dóna a tall d'exemple de construcció d'una màquina d'estats amb blocs combinacionals.

Figura 69. Circuit seqüencial de suma de nombres en sèrie



3. El primer que cal fer són les taules de transició i de sortides. En aquest cas, la de sortides es pot deduir de la codificació que s'explica a continuació.

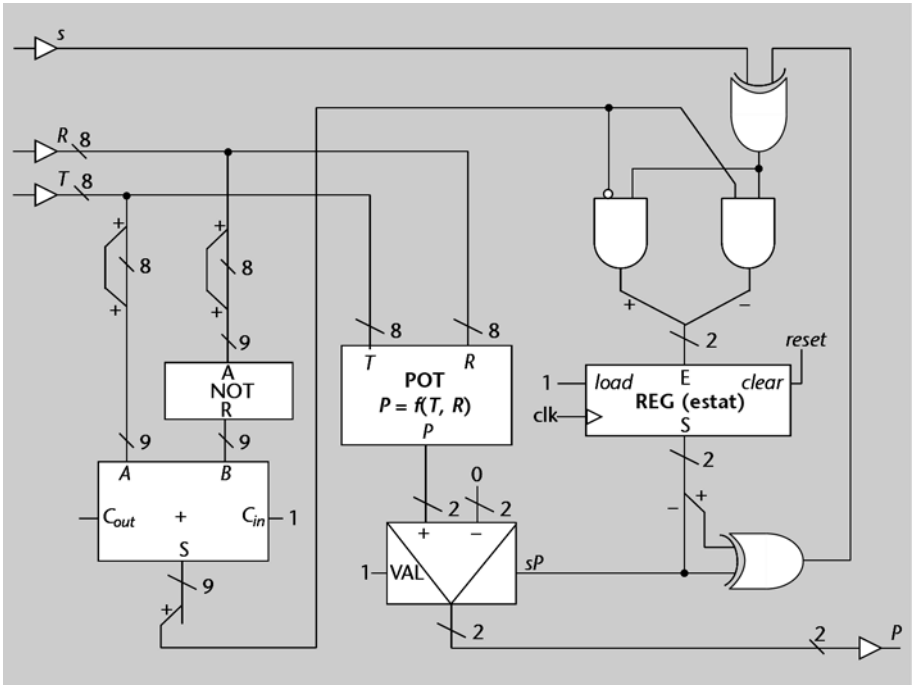
La codificació dels estats es fa segons el nombre binari corresponent, tenint en compte que el 0 es reserva per a l'estat inicial (APAGAT). La codificació de l'entrada de control serà un bit per al senyal de commutació *s*. I la condició ($T - R < 0$) s'expressarà amb un senyal que es denominarà *c*, que serà 1 quan es compleixi. Finalment, caldrà un selector de valor per a *P*, que serà *sP*. Quan *sP* sigui 0, $P = 0$ i quan sigui 1, $P = f(T, R)$.

Del graf d'estats de la figura 9 es pot deduir la taula de transicions següent:

Estat actual			Entrades		Estat següent		
Identificació	q_1	q_0	*s*	*c*	Identificació	q_1^+	q_0^+
APAGAT	0	0	0	x	APAGAT	0	0
APAGAT	0	0	1	0	ENCÈS	1	0
APAGAT	0	0	1	1	ESCALFANT	0	1
ESCALFANT	0	1	1	x	APAGAT	0	0
ESCALFANT	0	1	0	0	ENCÈS	1	0
ESCALFANT	0	1	0	1	ESCALFANT	0	1
ENCÈS	1	0	1	x	APAGAT	0	0
ENCÈS	1	0	0	0	ENCÈS	1	0
ENCÈS	1	0	0	1	ESCALFANT	0	1

A partir d'aquesta taula es poden obtenir les funcions q_1^+ i q_0^+. La sortida *sP* coincideix amb q_0, ja que només cal posar una potència del radiador diferent de 0 en l'estat d'ESCALFANT, que és l'únic que té q_0 a 1. L'esquema que presenta la figura 70 és el del circuit seqüencial corresponent.

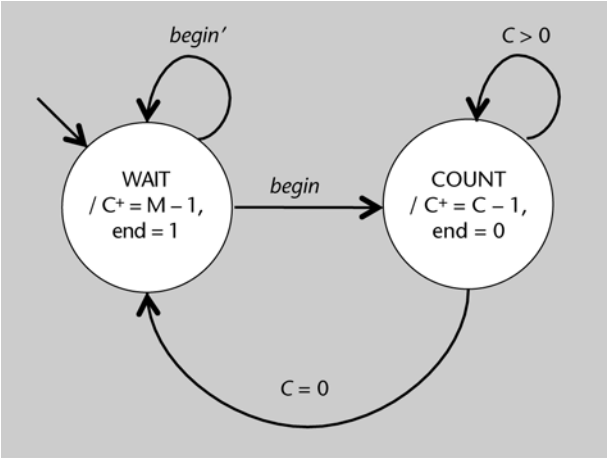
Figura 70. Controlador d'un calefactor amb termòstat



El càlcul de *c* es fa amb el bit de signe del resultat de l'operació $T - R$. Per a garantir que no hi hagi problemes de sobreeiximent (cas d'una temperatura negativa, per exemple), s'amplia el nombre de bits en 1, tot estenent el bit de signe.

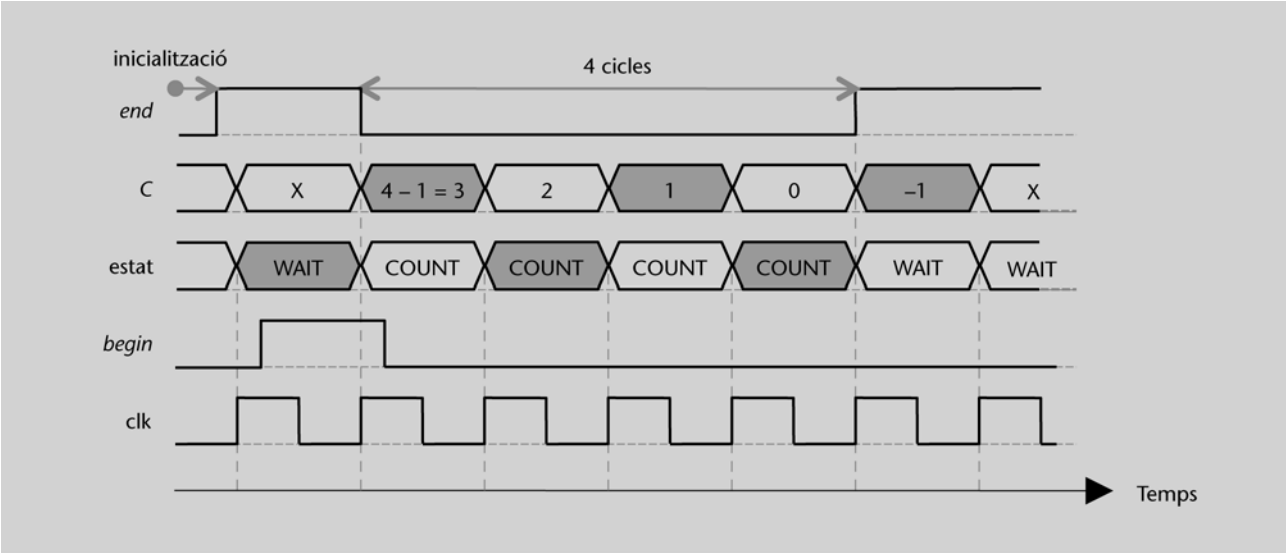
4. En aquest cas, en l'estat d'espera WAIT s'han d'anar efectuant càrregues de valors decrementats ($M - 1$) al registre de compte C fins que *begin* = 1. Llavors cal passar a l'estat d'anar comptant, COUNT. En aquest estat, es decrementa C i, si el darrer valor de C ha estat 0, s'acaba el compte. El diagrama de transicions d'estats és, doncs, molt similar al del comptador de la figura 10:

Figura 71. Graf d'estats d'un comptador enrere "programable"



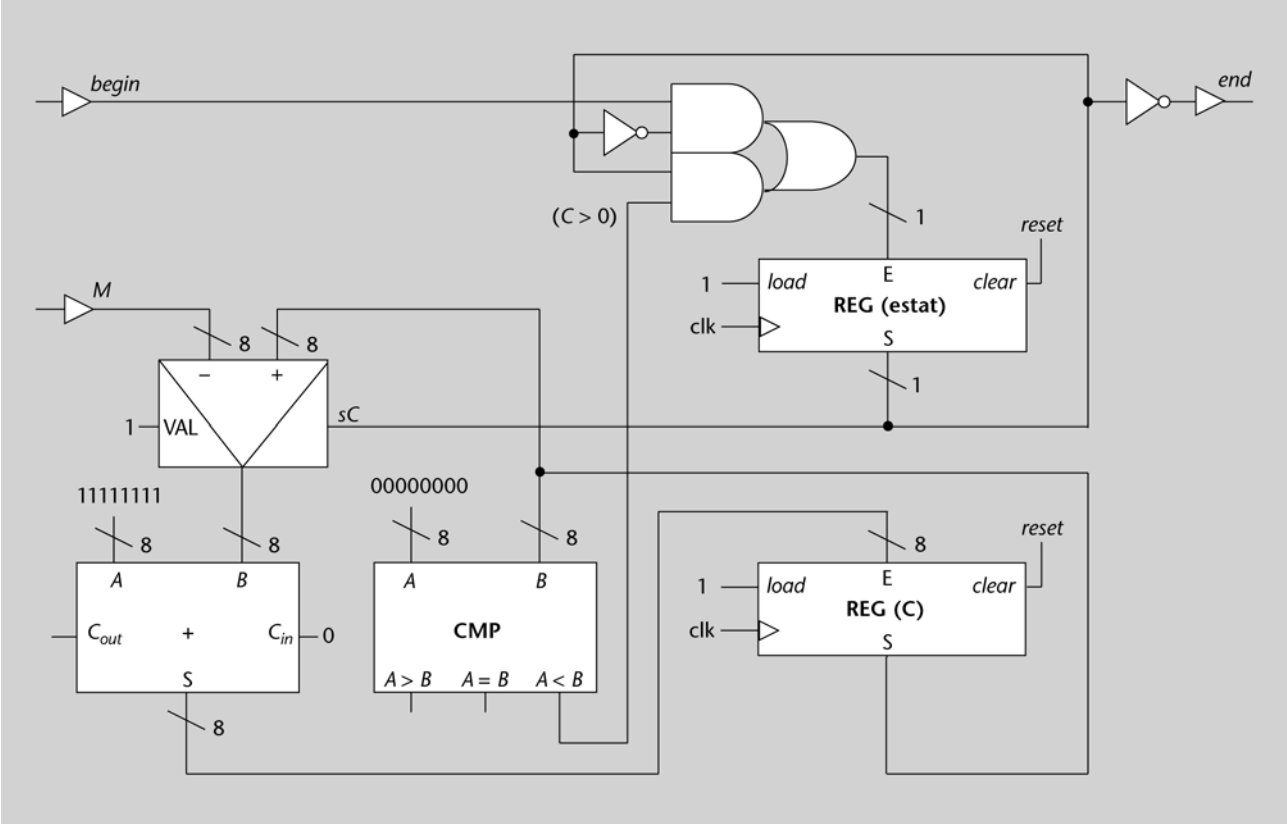
Cal tenir present que, tal com passa amb el comptador incremental, el registre C acaba carregant un valor de més que el compte que fa. En aquest cas, passa a tenir el valor –1 just en tornar a l'estat WAIT. Aquest valor es perd en el cicle següent, amb la càrrega d'un possible nou valor a comptar.

Figura 72. Cronograma d'exemple per al comptador enrere



El circuit corresponent a aquest comptador és més senzill que el del comptador incremental, ja que n'hi ha prou amb un únic sumador que faci les funcions de restador per a decrementar el proper valor de C i un únic registre. Atès que el graf d'estats és equivalent, la funció que calcula l'estat següent és la mateixa. En aquest cas, la condició ($C < B$) es transforma en ($C > 0$).

Figura 73. Circuit seqüencial corresponent al comptador enrere



5. L'EFSM del velocímetre té un punt en comú amb el comptador: la velocitat és, de fet, el compte de vegades que $c = 1$ fins que $s = 1$. En aquest cas, però, no s'ha de comptar a cada cicle i, per tant, caldran dos estats diferents per a tenir en compte si $c = 1$ (COUNT) o $c = 0$ (WAIT). A més, la condició d'aturada del compte no és una condició sobre una variable sinó sobre una entrada (s).

El fet que el valor de la velocitat s'hagi de mantenir durant tot un segon implica que cal una variable, V, per a emmagatzemar-la. Aquesta variable s'actualitzarà cada cop que passi un segon, és a dir, cada cop que $s = 1$.

Per al seu càlcul caldrà una variable que emmagatzemi el nombre de centímetres que s'han detectat, C.

Si es dóna el cas que, en un mateix període de rellotge, s i c són a 1, caldrà tenir-ho en compte: el centímetre es pot tenir en compte en el segon actual o bé en el posterior.

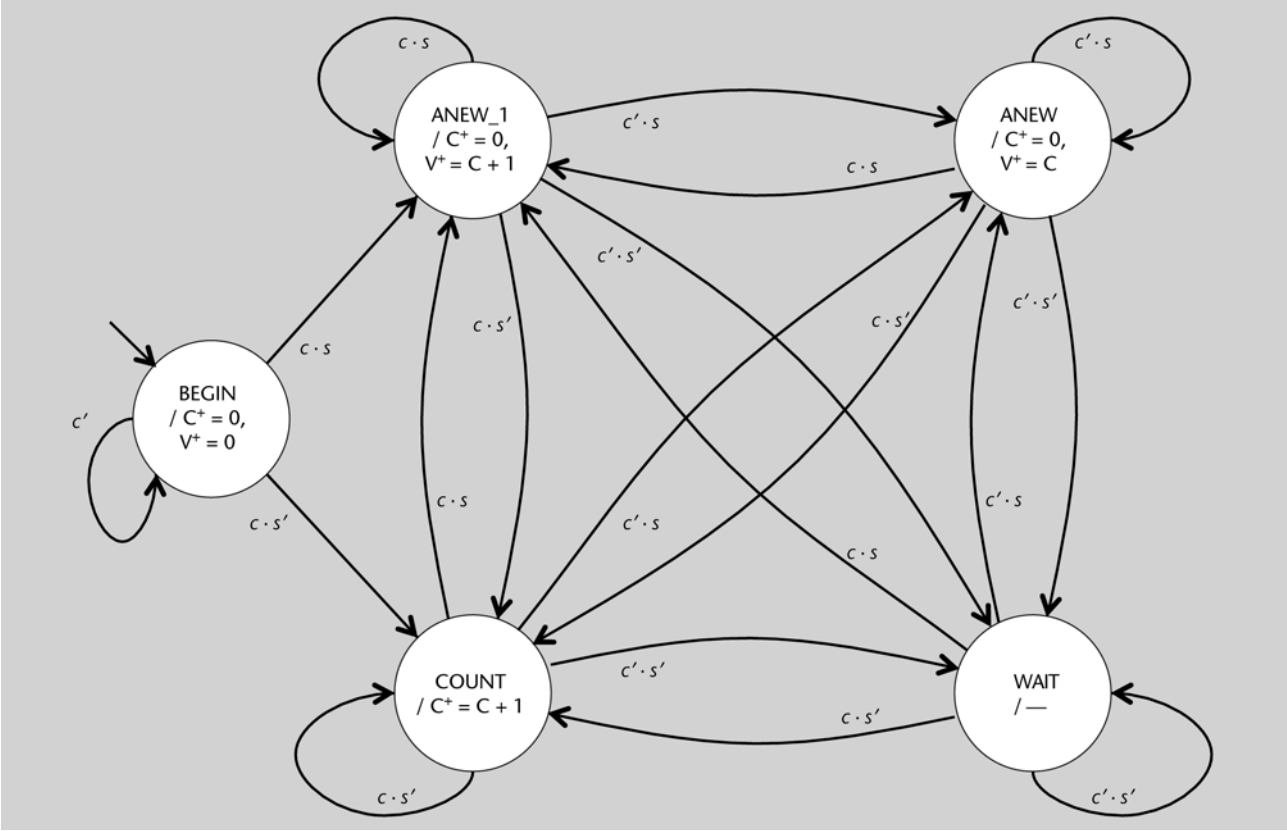
De fet, quan $s = 1$, hi haurà dos estats que reiniciaran el compte: ANEW, si no hi ha simultaneïtat amb c, i ANEW_1, si n'hi ha. En el darrer cas es pot optar per inicialitzar el comptador C a 1, que significa que el centímetre es comptaria en el segon posterior, o per acumular-lo en el compte del segon actual, que és el que es fa en el graf que es mostra a continuació.

Tot i que té una aparença complicada, les transicions són senzilles, ja que cal pensar en les accions associades a cada estat. Així, a COUNT s'hi va cada cop que cal comptar un centímetre sense que hagi passat un segon (cs'), a WAIT, cada cop que les entrades són a 0 ($c's'$), a ANEW, cada cop que s'acaba un compte i s'ha d'actualitzar $V(sc')$ sense considerar un centímetre simultani, cosa que es fa a ANEW_1.

Val a dir que el graf d'estats es pot simplificar si se suposa que, de bon principi, es fa un *reset* al circuit. Essent així, els registres C i V també es posarien a 0 i ja no caldria l'estat BEGIN. En aquest cas, l'estat inicial hauria de ser el d'espera, WAIT.

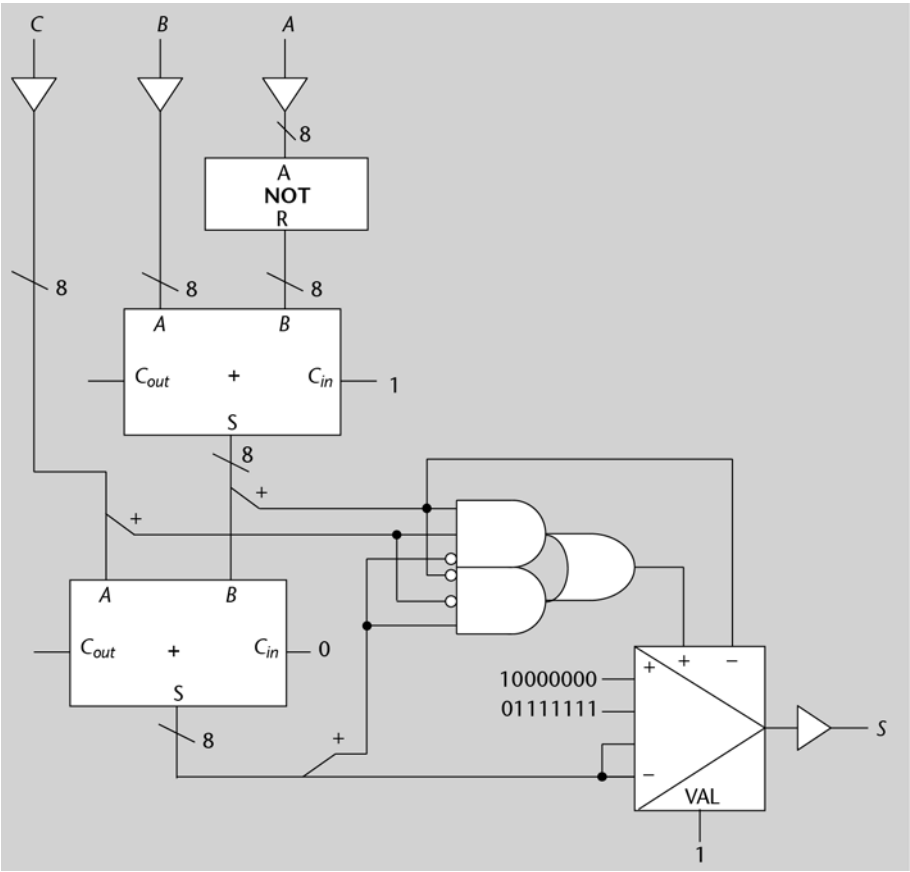
A tall de curiositat, es podria fer servir el comptador de l'activitat anterior de manera que comptés tants cicles de rellotge com caben en un segon. D'aquesta manera, el circuit conjunt només necessitaria una entrada que l'indiqués quan s'ha avançat un centímetre, a més de la de *reset* i de rellotge, és clar.

Figura 74. Graf d'estats del velocímetre



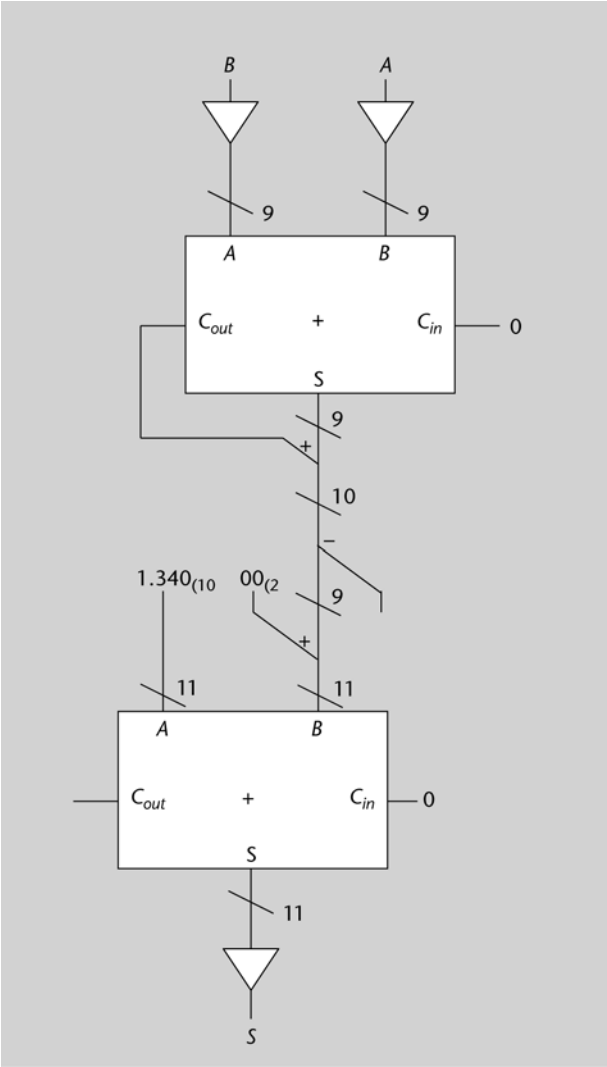
6. El mòdul que calcula $C + B - A$ pot incórrer en sobreeiximent. Per a simplificar-ne el disseny, primer es fa la resta, que no en genera mai, i després, la suma. En cas que hi hagi sobreeiximent, el resultat serà el nombre més gran o el més petit possible, segons si la suma era de nombres positius o negatius, respectivament.

Figura 75. Mòdul de suma d'error a la referència del controlador de velocitat



El mòdul que calcula 1.340 + (*A* + *B*)/2 no té problemes de sobreeiximent perquè el resultat pot ser, com a màxim, 1.340 + (2^9 – 1) = 1.851, que és més petit que el nombre més gran que es pot representar amb 11 bits, 2^{11} – 1 = 2.047. De fet, el valor màxim del resultat és 1.340 + 320 = 1.660, considerant la funció de la figura 20. La divisió per 2 es fa desplaçant el resultat de la suma d'*A* i *B* a la dreta, descartant el bit menys significatiu, però s'ha de tenir en compte que la suma pot generar 1 bit de ròssec que s'ha d'incloure en el valor a desplaçar. Per a això, primer es forma un nombre de 10 bits amb el bit de ròssec i, després, es fa el desplaçament a la dreta.

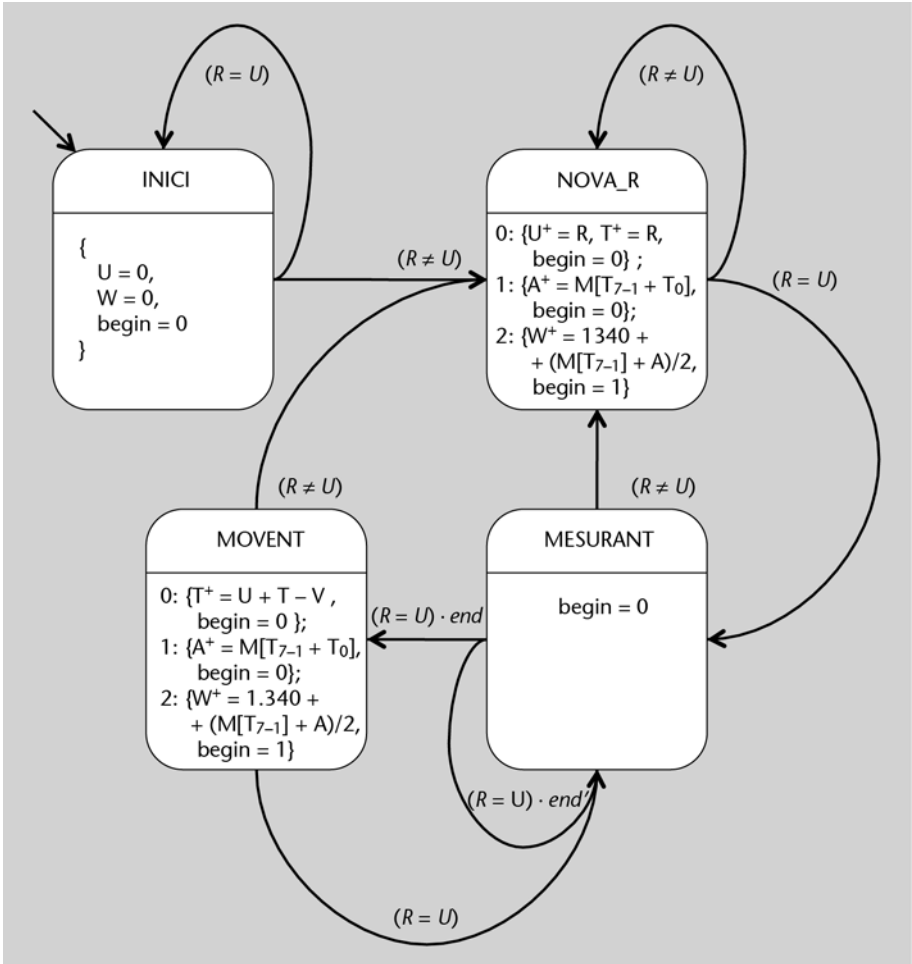
Figura 76. Mòdul de càlcul de l'amplada de pols del controlador de velocitat



7. La detecció de si cal carregar una nova referència o no es pot fer comparant el valor actual de la variable *U* amb el de l'entrada *R*. Si és igual, significa que no hi ha canvis. Si és diferent, cal fer una càrrega d'un nou valor. Per tant, $R \neq U$ és equivalent a *ld_r*. Així doncs, la modificació en el model del PSM consisteix a substituir *ld_r* per ($R \neq U$) i *ld_r'* per ($R = U$).

El circuit corresponent seria molt similar, però hauria d'incorporar un comparador entre *U* i *R* per a generar, de manera interna, el senyal *ld_r*.

Figura 77. Model modificat de la PSM del control adaptatiu de velocitat



8. La taula de veritat es pot construir a partir de la representació de l'ASM corresponent en la figura 31.

Estat actual		**Entrades**						**Estat següent**	
Identificació	S_{2-0}	*d*	*t*	*r*	*s*	*e*	*h*	**Identificació**	S^+_{2-0}
IDLE	000	0	x	x	x	x	x	IDLE	000
IDLE	000	1	0	x	x	x	x	UP	100
IDLE	000	1	x	0	x	x	x	UP	100
IDLE	000	1	1	1	x	x	x	MORE	001
MORE	001	x	x	x	x	x	x	HEAT	101
HEAT	101	x	x	x	0	0	x	HEAT	101
HEAT	101	x	x	0	0	1	0	UP	100
HEAT	101	x	x	x	1	x	x	UP	100
HEAT	101	x	x	0	0	1	1	KEEP	110
HEAT	101	x	x	1	0	1	x	MORE	001
KEEP	110	x	x	x	0	x	x	KEEP	110
KEEP	110	x	x	x	1	x	x	UP	100
UP	100	x	x	x	x	x	x	IDLE	000

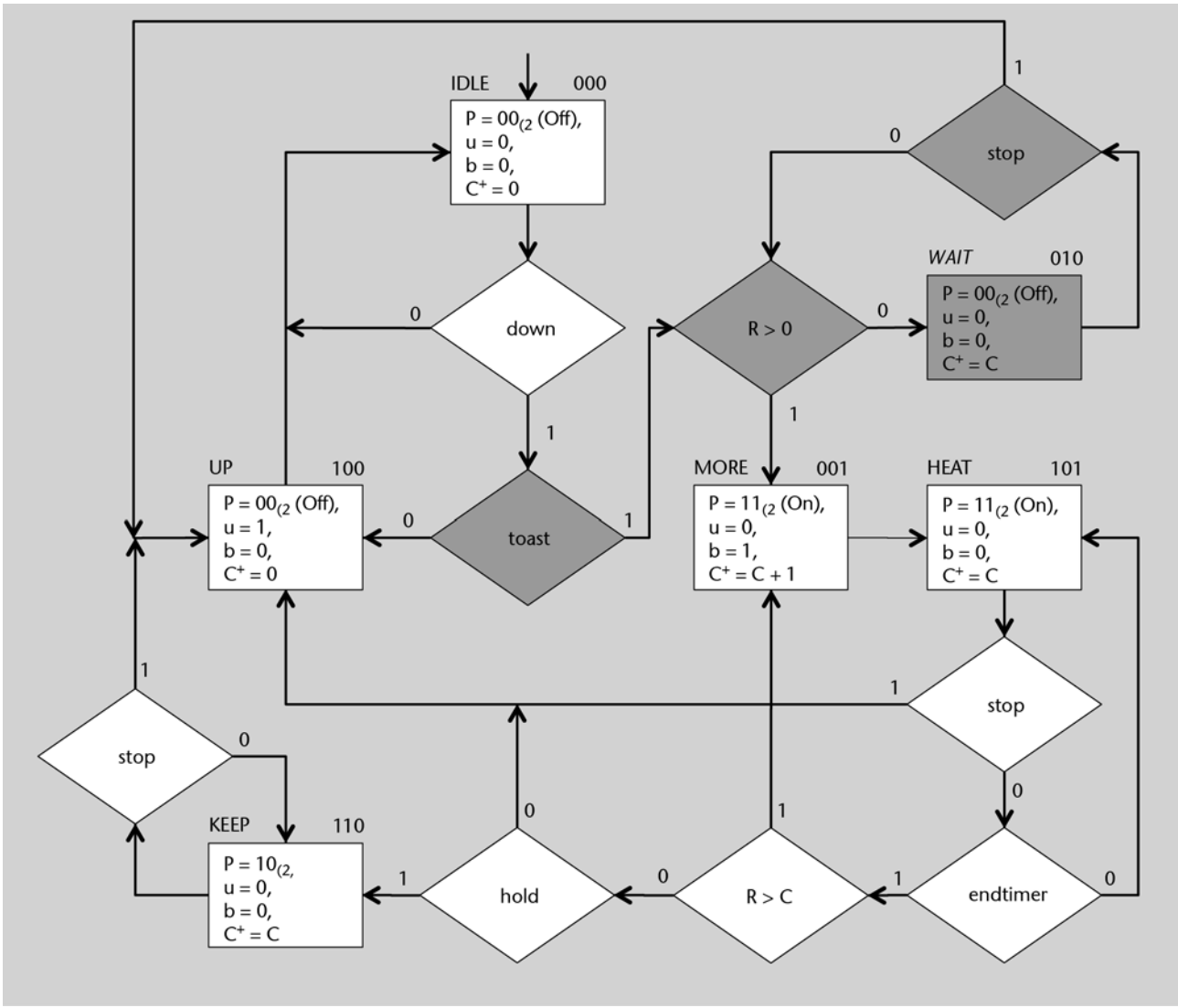
Es pot comprovar que hi ha molts de *don't-cares* que poden ajudar a obtenir expressions mínimes per a les funcions de càlcul de l'estat següent, però també que són difícils de manipular de manera manual.

9. La modificació implica dividir en dos la caixa de decisió sobre *toast* & ($R > 0$), de manera que primer es pregunti sobre si s'ha detectat torrada i, en cas afirmatiu, es comprovi si $R > 0$ per a passar a MORE. En cas que R sigui 0, es passarà a una nova caixa d'estat (WAIT) en la qual s'esperarà que es premi el botó d'aturada (*stop*) o es giri la roda a una posició diferent de 0.

En aquest cas, és més complicat trobar una codificació d'estats de manera que, entre estats veïns, hi hagi el mínim de canvis possible. Especialment perquè l'estat UP té quatre veïns i, en una codificació de 3 bits, com a molt, hi ha tres veïns possibles per a cada codi en què només es canvia 1 bit.

En l'ASM que s'ha modificat, s'ha aprofitat un dels codis lliures per a WAIT (010). Les accions associades són les de mantenir els elements calefactors apagats sense fer saltar la llesca de pa amunt ni activar el temporitzador.

Figura 78. ASM del controlador d'una torradora amb estat d'espera



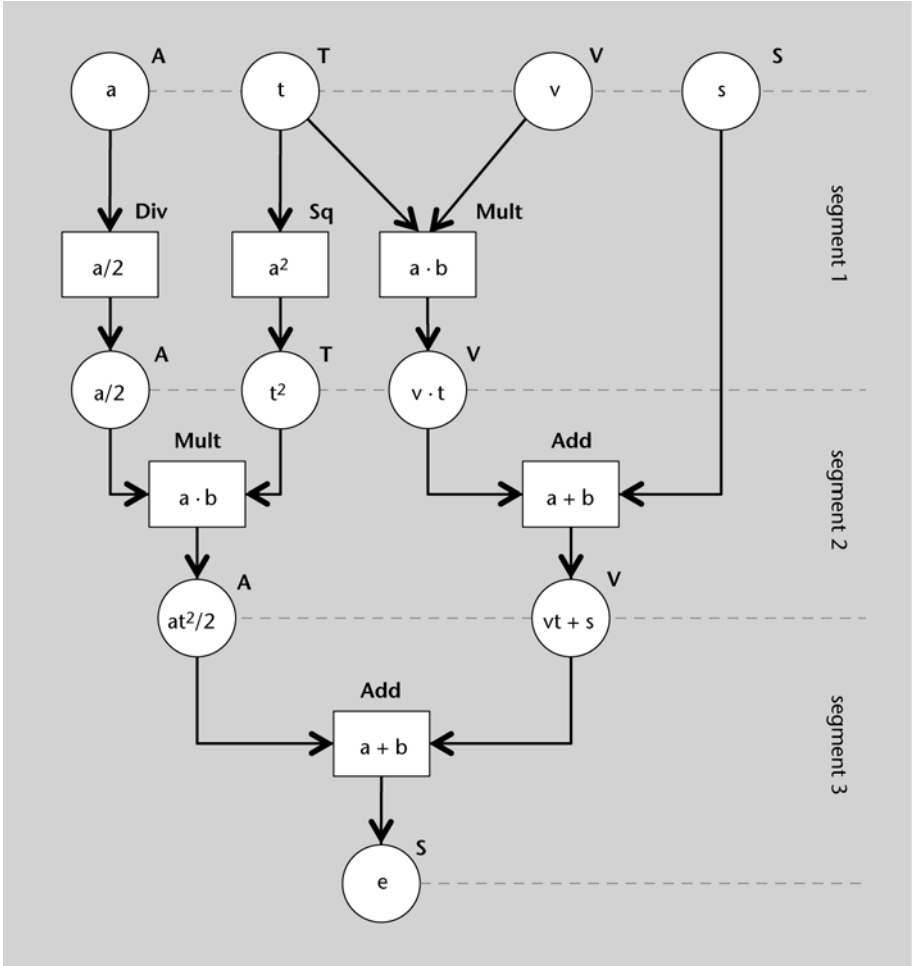
10. Abans de fer l'esquema de càlcul, és convenient de posar parèntesis a l'expressió, per a agrupar les operacions. Així doncs, l'expressió donada es pot transformar en:

$$(((a/2) \cdot (t^2)) + ((v \cdot t) + s))$$

A partir de l'expressió anterior es pot generar fàcilment el diagrama de l'esquema de càlcul corresponent. En aquest cas, s'ha optat per segmentar-lo fins a arribar a un bon compromís entre el nombre d'etapes (3) i el nombre de recursos de càlcul (4). Sense segmentació haurien calgut sis operadors, però el retard seria, a la pràctica, molt gran. Amb segmentació màxima i aprofitant el multiplicador per a fer el quadrat de t, caldria una etapa més, fins a quatre, i només tres recursos de càlcul. No obstant això, el nombre d'etapes també implica, finalment, un temps de procés llarg.

En la figura 79 es mostra la solució amb un esquema de càlcul segmentat. L'esquema també porta etiquetes per a les variables, associant-les a registres. En aquest cas, s'ha optat per un reaprofitament total. Per a permetre un funcionament progressiu, caldria que a cada etapa es fessin servir registres diferents. Així doncs, amb deu registres seria possible iniciar un nou càlcul a cada cicle de rellotge.

Figura 79. Esquema de càlcul de $at^2/2 + vt + s$



11. Quant als multiplexors lligats a les entrades dels recursos de càlcul, cal tenir present que s'ocupen de seleccionar les dades que necessiten segons el període de rellotge en què s'estigui. Així, les entrades en posició 0 dels multiplexors proporcionen als recursos associats les dades per al primer període de rellotge, les entrades 1 corresponen a les dades del segon període i així fins al màxim nombre de períodes de rellotge que calguin. Ara bé, no tots els recursos de càlcul es fan servir en tots els períodes de rellotge i, per tant, els resultats que generen en aquests períodes de temps no importen, de la mateixa manera que tampoc no importen quines entrades tinguin. Així doncs, es pot construir una taula que ajudi a aprofitar aquests casos irrellevants:

Estat	<<	+	CMP
S0	*x*	*x*	*C* = *n*
S1	*R* << 1	*C* + 1	*x*
S2	*x*	*R* + *M*	*x*
S3	*P* << 1	*x*	*C* = *n*

En el circuit de la figura 44 tots els *don't-cares* apareixien com a 0 en les entrades dels multiplexors. Amb vista a l'optimització, però, s'han de considerar com a tals. Així, es pot observar que el comparador pot estar connectat sempre a les mateixes entrades i el multiplexor es pot suprimir. En el cas de la suma, n'hi hauria prou amb un multiplexor de dues entrades. Una cosa similar passa amb el decalador. En aquest cas, però, resulta més convenient fer servir dos decaladors que no pas tenir-hi un multiplexor (els decaladors són molt més simples que els multiplexors, ja que no els calen portes lògiques).

En el cas dels registres, cal tenir en compte que la tria entre mantenir el valor i carregar-ne un de nou es pot fer amb el senyal de càrrega (*load*) corresponent, cosa que ajuda a reduir l'ordre dels multiplexors a la seva entrada. És convenient tenir una taula similar que ajudi a visualitzar les optimitzacions que es poden fer, com la que es mostra a continuació.

Estat	*R*	*M*	*P*	*C*
S0	0	*A*	*B*	0
S1	*R* << 1	*M*	*P*	*C* + 1
S2	*R* + *M*	*M*	*P*	*C*
S3	*R*	*M*	*P* << 1	*C*

El cas més senzill és el del registre *M*, que només ha de fer una càrrega en l'estat *S0* i, per tant, n'hi ha prou de fer servir *S0* connectat al senyal corresponent del registre associat. Una cosa similar es pot fer amb *C*, si la posada a 0 es fa aprofitant el *reset* del registre. En aquest cas, es faria un *reset* en l'estat *S0* i una càrrega a l'estat *S1*. Amb *P* és necessari fer servir un multiplexor de dues entrades per a distingir entre els dos valors que s'hi poden carregar (*B* o *P* << 1). Amb *R* es pot aprofitar la mateixa solució, si la posada a zero es fa amb *reset*.

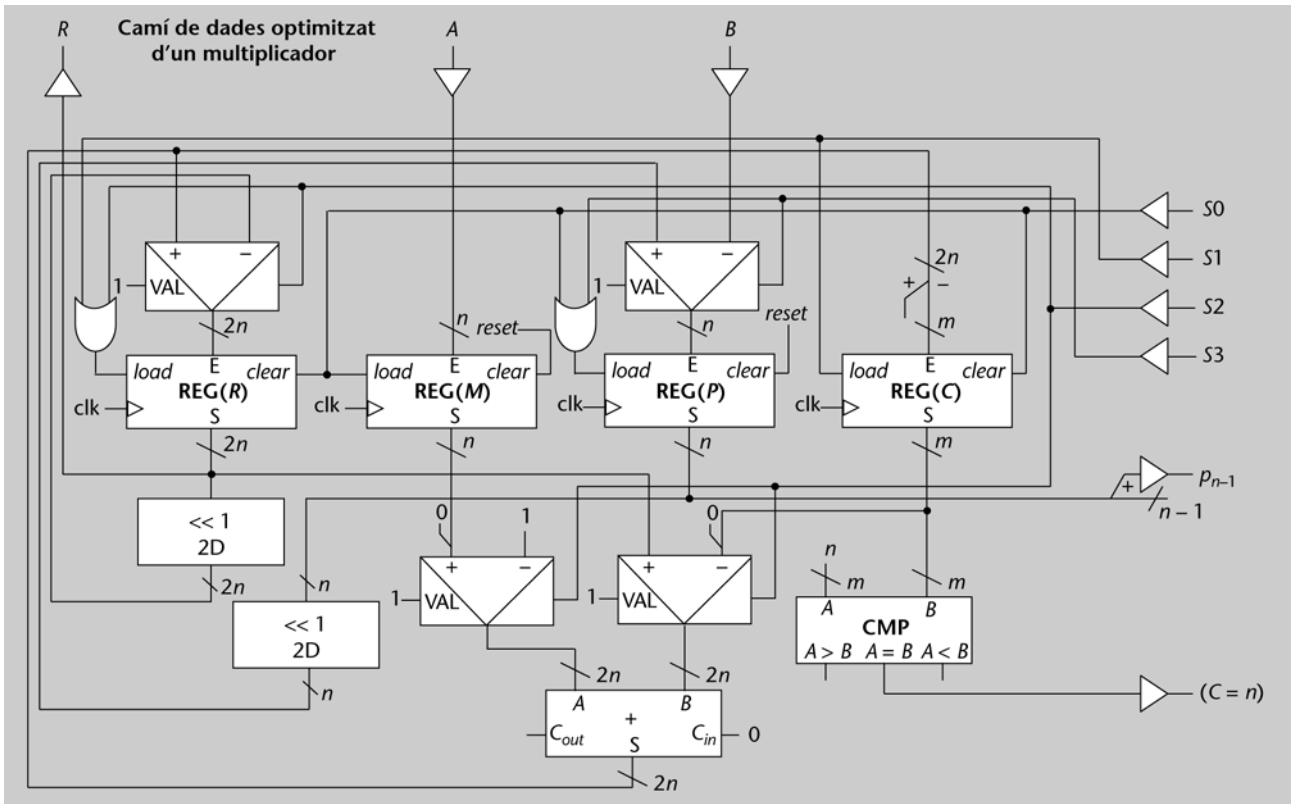
Finalment, cal tenir en compte que les funcions lògiques que generen els valors de control dels multiplexors, dels *loads* i dels *resets* solen ser més simples amb la codificació *one-hot bit* que amb la binària. De fet, en aquest cas no hi ha gaire diferència, però fer servir la codificació *one-hot bit* estalvia el codificador de la unitat de control.

En la taula següent es mostren els senyals anteriors en funció dels bits d'estat. En la taula, les operacions en què es carrega un valor estan separades per punts suspensius, de manera que l'activació del senyal de càrrega es mostra com a "*P* = ..." o "*R* = ..." i la selecció del valor a carregar com a "... *B*", "... *P* << 1", "... *R* << 1" i "... *R* + *M*".

Estat	+	*loadM*	*loadP*	*selP*	*loadC*	*resetC*	*loadR*	*selR*	*resetR*
S0	(*C* + 1)	*M* = *A*	*P* = ...	... *B*	—	*C* = 0	—	... *R* << 1	*R* = 0
S1	*C* + 1	(*M* = *M*)	(*P* = *P*)	... *B*	*C* = *C* + 1	—	*R* = ...	... *R* << 1	—
S2	*R* + *M*	(*M* = *M*)	(*P* = *P*)	... *B*	(*C* = *C*)	—	*R* = ...	... *R* + *M*	—
S3	(*C* + 1)	(*M* = *M*)	*P* = ...	... *P* << 1	(*C* = *C*)	—	(*R* = *R*)	... *R* << 1	—

El circuit optimitzat es mostra en la figura 80.

Figura 80. Unitat de processament d'un multiplicador en sèrie



12. Per a completar-la s'ha de fer la traducció dels símbols del programa als codis binaris corresponents. De fet, la codificació en binari d'aquestes dades és prou directa, seguint el format de les instruccions que interpreta el Femtoproc.

Per a obtenir la codificació en binari d'una instrucció, es pot traduir a binari, d'esquerra a dreta, primer el símbol de l'operació a fer (ADD, AND, NOT, JZ) i, després, els que representen els operands, que poden ser registres o adreces. Si són registres, n'hi ha prou d'obtenir el nombre binari equivalent al número de registre que s'especifica. Si són adreces, es "desempaqueta" el nombre hexadecimal en el binari equivalent.

En la taula següent es completa la codificació del programa del MCD.

Adreça	Instrucció	Codificació	Comentari
...	...	...	...
0Bh	`AND R0, R1`	01 000 001	Força que el resultat sigui 0.
0Ch	`JZ 04h (sub)`	11 000100	Torna a fer una altra resta.
pos,0Dh	`NOT R3, R4`	10 011 100	
0Eh	`NOT R3, R3`	10 011 011	*R3* = *R4* = *R3* – *R2*
0Fh	`JZ 12h (end)`	11 010010	
10h	`AND R0, R1`	01 000 001	
11h	`JZ 04h (sub)`	11 000100	Torna a fer una altra resta.
end,12h	`ADD R5, R2`	00 101 010	
13h	`ADD R5, R3`	00 101 011	*R5* = *R2* + *R3*, però un dels dos és 0.
14h	`AND R0, R1`	01 000 001	En espera, fins que se li faci *reset*.
hlt,15h	`JZ 15h (hlt)`	11 010101	

13. Amb el repertori d'instruccions del Femtoproc no és possible copiar el valor d'un registre a un altre de diferent de manera directa. Ara bé, es pot aprofitar la instrucció NOT per a desar en un registre el complement del contingut d'un altre registre font. Després, n'hi ha prou de fer una operació NOT del primer registre, de manera que aquest registre en sigui font i destinació alhora.

Aquesta situació es dóna en les instruccions de les posicions 0Dh i 0Eh del programa anterior:

Adreça	Instrucció	Codificació	Comentari
...	...	...	...
0Bh	`AND R0, R1`	01 000 001	Força que el resultat sigui 0.
0Ch	`JZ 04h (sub)`	11 000100	Torna a fer una altra resta.
pos,0Dh	`NOT R3, R4`	10 011 100	
0Eh	`NOT R3, R3`	10 011 011	*R3* = *R4* = *R3* – *R2*
0Fh	`JZ 12h (end)`	11 010010	
10h	`AND R0, R1`	01 000 001	
...	...	...	...

14. Fer un desplaçament a l'esquerra d'1 bit d'un nombre equival a multiplicar-lo per 2. La multiplicació per 2 es pot fer amb una suma. Així doncs, si es volen desplaçar els bits del registre *R3* un bit a l'esquerra n'hi ha prou de fer `ADD R3, R3`.

15. Seguint el patró d'aquests materials i de l'exemple que es dóna, cal una microordre de càrrega per a cada registre i biestable: *ld_PC*, *ld_IR*, *ld_MB*, *ld_MAR*, *ld_A*, *ld_X*, *ld_C* i *ld_Z*. Com que tots els registres només tenen una entrada, no cal cap selector per a controlar cap multiplexor addicional.

Ara bé, els registres que carreguen la dada provinent del *BusW* carreguen el resultat de fer alguna operació amb l'ALU. Aquesta operació pot tenir dos operands, un dels quals és el registre MB, i l'altre pot ser PC, A o X. Cal, doncs, una microordre de selecció de quin d'aquests tres registres transfereix el contingut al *BusR*. Aquesta microordre seria un senyal de 2 bits que es podria denominar *selBusR*.

En resum, doncs, calen deu microordres, una de les quals (*selBusR*) és de 2 bits i l'altra, de selecció d'operació de l'ALU, de 5. En total, 15 bits. Per tant, seria compatible amb el format de microinstruccions vist en l'apartat 3.2.2 i, consegüentment, es podria implementar amb seqüenciador de la figura 53.

16. En aquest cas, es tractaria d'una instrucció d'un únic byte, ja que no hi ha cap operand en la memòria. Així doncs, només caldria llegir la instrucció, comprovar que sigui d'increment de X (suposem que el seu símbol és INX), fer l'operació i passar a la instrucció següent. Això últim es pot fer simultàniament amb la lectura del codi d'operació, tal com s'ha vist.

Etiqueta	Codi d'operació	μ–instrucció	Comentari
START:	EXEC	`MAR = PC`	**Fase 1**. Lectura de la instrucció
	EXEC	`MB = M[MAR], PC = PC + 1`	Càrrega del *buffer* de memòria i increment del *PC*
	EXEC	`IR = MB`	
	JMPIF	`INX?, X_INX`	**Fase 2**. Descodificació: Si és INX, salta a X_INX
...			
X_INX:	EXEC	`X = X + 1`	**Fase 3**. Execució de l'operació
	JMPIF	`Inconditional, START`	Bucle infinit d'interpretació
...			

17. El *pipe* s'omplirà fins que s'acabi l'execució de la segona instrucció. Havent acabat de la fase d'execució de l'operació (EO), el *pipe* es "buida" de manera que les fases ja iniciades de les instruccions número 3, 4 i 5 no es continuen. Per tant, el *pipeline* tindrà una latència de quatre períodes de rellotge més fins no executar l'operació d'una nova instrucció.

Figura 81. *Pipeline* de quatre etapes amb salt de seqüència a la segona instrucció

Instrucció	Etapa del *pipeline* (fase del cicle)								
1	LI	LA	CO	EO					
2		LI	LA	CO	EO				
3			LI	LA	CO	X			
4				LI	LA	X			
5					LI	X			
11						LI	LA	CO	EO
12							LI	LA	CO
Període	1	2	3	4	5	6	7	8	9

Les "X" del *pipeline* indiquen que els valors dels registres a la sortida de les etapes corresponents no importen, ja que s'han de calcular de nou.

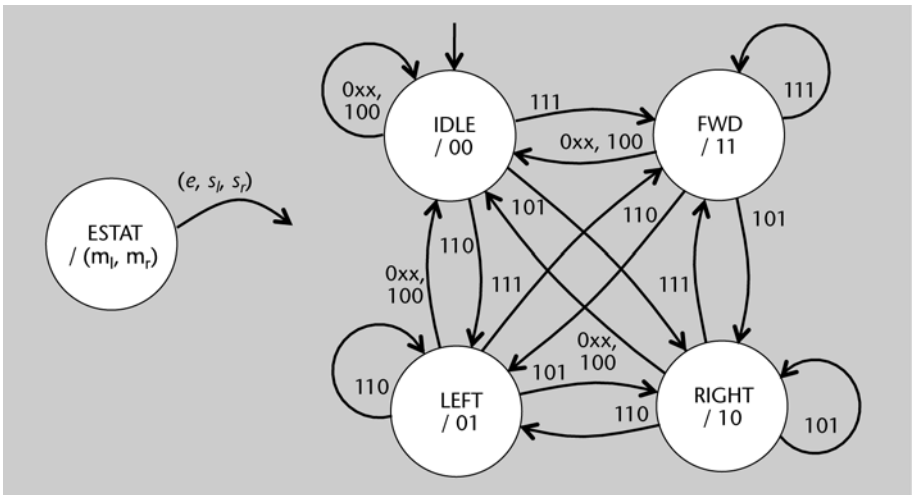
Exercicis d'autoavaluació

1. Per a fer el graf d'estats que representi el comportament que s'ha indicat en l'enunciat cal començar per l'estat inicial, que es denominarà IDLE perquè, inicialment, el robot no ha de fer res. En aquest estat, la sortida ha de ser $(m_l, m_r) = (0, 0)$, ja que el vehicle ha de restar immòbil. Tant si l'entrada e és 0 com si és 1 i no es detecta línia en cap dels sensors que donen els senyals d'entrada, el robot s'ha de mantenir en aquest estat. De fet, des de qualsevol altre estat que tingui aquesta màquina d'estats s'ha de passar a IDLE quan es detecti alguna d'aquestes condicions. És a dir, quan (e, s_l, s_r) siguin (0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1) o (1, 0, 0).

Des de l'estat d'IDLE, amb $e = 1$, es passa a l'estat d'anar endavant (FWD) si es detecta línia en els dos sensors o als de girar a l'esquerra (LEFT) o a la dreta (RIGHT) si la línia només es detecta en l'entrada del sensor esquerre o dret, respectivament. De manera similar, cadascun d'aquests tres estats ha de tenir un arc que vagi als altres dos i un de reentrant per al cas corresponent al mateix estat.

El dibuix del graf corresponent és un xic complicat perquè tots els nodes estan connectats entre ells.

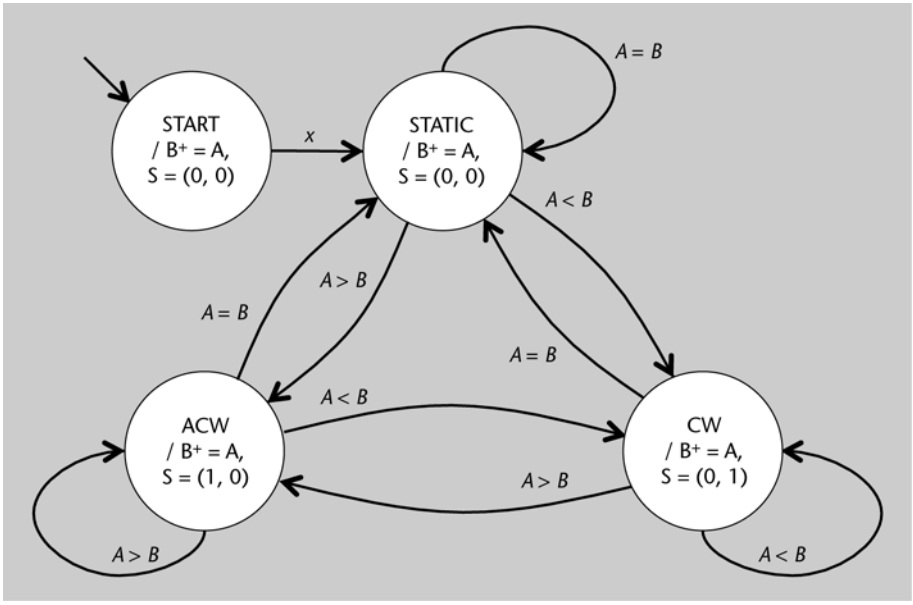
Figura 82. Graf d'estats del controlador d'un seguidor de línies



2. L'enunciat del problema defineix tant el comportament que ha seguit el detector com les entrades i sortides que té, com també la seva codificació. Ara bé, s'ha de tenir en compte que les EFSM poden fer servir variables. De fet, com que la detecció del sentit del gir es fa comparant la posició angular actual (A) amb l'anterior, l'EFSM corresponent n'ha de tenir una que l'emmagatzemi, que es denominarà B. Així doncs, si $A > B$ l'eix gira en sentit contrari a les agulles del rellotge i la sortida ha de ser (1, 0).

Amb aquesta informació ja es pot establir el graf d'estats, que començarà en l'estat inicial START. En aquest estat la sortida ha de ser (0, 0) i s'ha de donar el valor inicial a B, que ha de ser el de l'entrada A, és a dir, $B^+ = A$. Com que no és possible comparar el valor de la posició angular actual amb l'anterior, d'aquest estat s'ha de passar a un altre estat en què es pugui fer una segona lectura. Així, es pot passar de START a STATIC amb la mateixa sortida. Des de STATIC s'haurà de passar a ACW (de l'anglès *anticlockwise*, en el sentit contrari a les agulles del rellotge) si $A > B$, a CW (de l'anglès *clockwise*, en el sentit de les agulles del rellotge) si $A < B$, o restar a STATIC si $A = B$. En tots els estats, cal emmagatzemar en B la posició angular actual. El graf corresponent es mostra en la figura 83.

Figura 83. EFSM del detector del sentit de gir



Atès que la codificació binària de les entrades i sortides ja queda definida en l'enunciat del problema, només queda codificar la variable *B* i els estats: *B* ha de tenir el mateix format que *A*, és a dir, ha de ser un nombre natural de 3 bits, i els estats es codifiquen segons la numeració binària, de manera que START = 00, ja que és l'estat al qual s'ha de passar en cas de *reset*. Com es pot veure en la taula de transicions següent, s'ha fet que els estats *ACW* i *CW* tinguin una codificació igual a la de les sortides *S* corresponents.

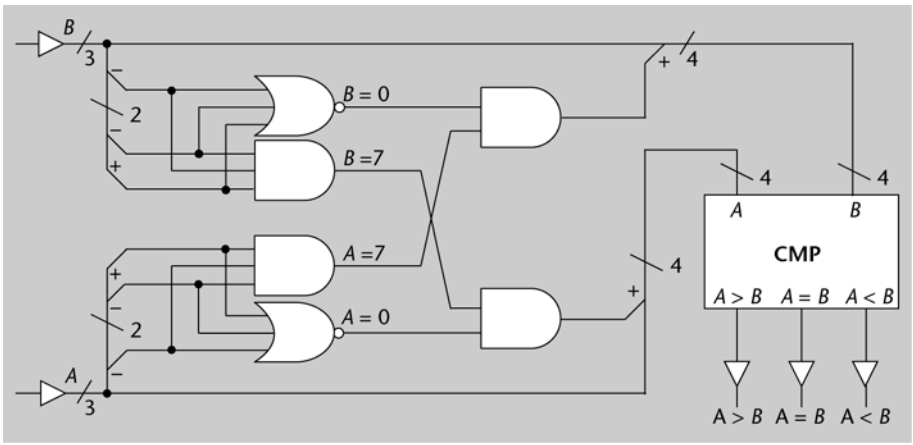
Estat actual			**Entrades**			**Estat següent**		
Identificació	q_1	q_0	$A > B$	$A = B$	$A < B$	**Identificació**	q_1^+	q_0^+
START	0	0	x	x	x	STATIC	1	1
CW	0	1	0	0	1	CW	0	1
CW	0	1	0	1	0	STATIC	1	1
CW	0	1	1	0	0	ACW	1	0
ACW	1	0	0	0	1	CW	0	1
ACW	1	0	0	1	0	STATIC	1	1
ACW	1	0	1	0	0	ACW	1	0
STATIC	1	1	0	0	1	CW	0	1
STATIC	1	1	0	1	0	STATIC	1	1
STATIC	1	1	1	0	0	ACW	1	0

Quant a les accions lligades a cada estat, l'assignació $B^+ = A$ s'ha de fer en tots i, conseqüentment, es fa de manera independent de l'estat en què s'estigui. La funció de sortida *S* és molt senzilla:

$$S = (s_1, s_0) = (ACW, CW) = (q_1 \cdot q_0', q_1' \cdot q_0)$$

Amb vista a la implementació cal tenir en compte una cosa quant al comparador: ha de ser un "comparador circular". És a dir, ha de tenir en compte que del sector 7 es passa, en el sentit contrari de les agulles del rellotge, al 0 i que del 0 es passa al 7. Per tant, s'ha de complir que {(0 < 1), (1 < 2), (2 < 3), ..., (5 < 6), (6 < 7), (7 < 0)}. Perquè això passi, una de les opcions és fer servir un comparador convencional de 4 bits en el qual el quart bit sigui 0 excepte en el cas en què el nombre corresponent sigui 000 i l'altre sigui 111. D'aquesta manera, el 0 passaria a ser 8 quan es comparés amb el 7. Aquesta solució és vàlida perquè se suposa que no hi pot haver salts de dos o més sectors entre dues lectures consecutives. El circuit corresponent a aquest comparador circular es mostra en la figura 84.

Figura 84. Circuit d'un comparador circular de 3 bits



Per a la materialització de l'EFSM amb arquitectura d'FSMD se separa la part de control de la de processament de dades. La unitat de control corresponent pren les entrades ($A > B$), ($A = B$) i ($A < B$) de la unitat de processament, que conté un comparador circular com el que s'ha vist. La unitat de control s'ocupa de decidir l'estat següent de l'EFSM i les accions associades a cada estat. En aquest cas, han d'implementar les funcions de transició d'estat i les de sortida *S*, que ja són la mateixa sortida de l'EFSM i que ja s'han definit amb anterioritat.

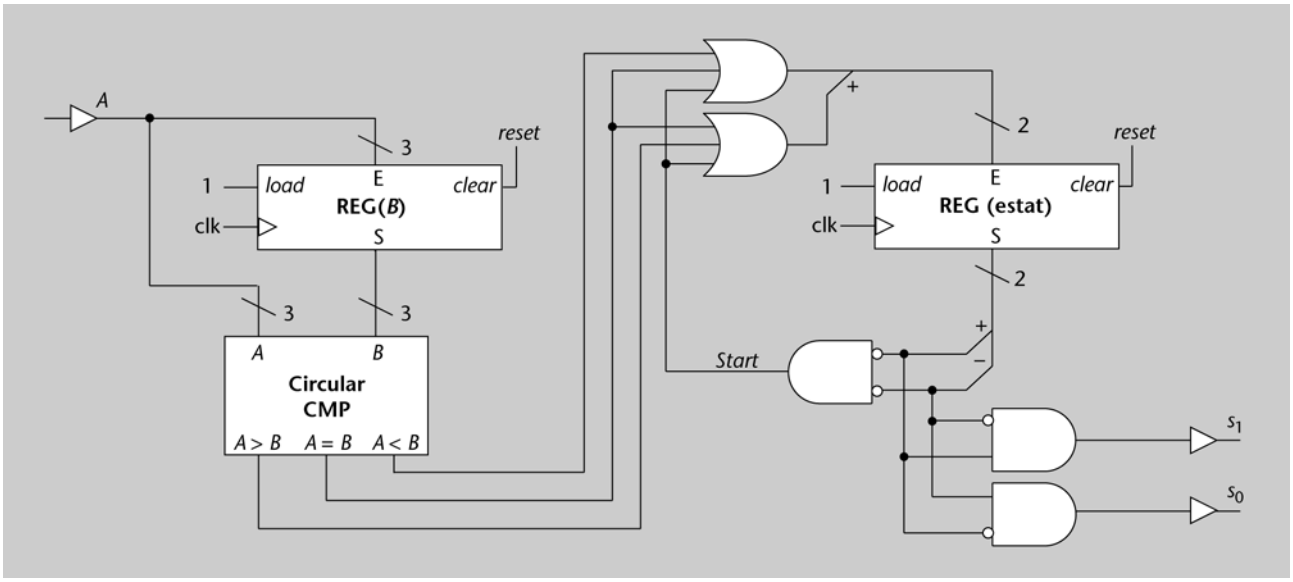
Les funcions de transició s'obtenen de la taula de veritat corresponent. En aquest cas, n'hi ha prou d'observar que l'estat de destinació depèn exclusivament de les entrades i que només hi ha una entrada activa a cada moment, amb l'excepció de la transició de sortida de l'estat START.

$$q_1^+ = q_1' \cdot q_0' + (A = B) + (A > B)$$
$$q_0^+ = q_1' \cdot q_0' + (A = B) + (A < B)$$

En la unitat de processament s'ha d'efectuar l'operació de càrrega de la variable B i s'hi han de generar els senyals $(A < B)$, $(A = B)$ i $(A > B)$, cosa que es fa amb un comparador circular.

El circuit de tipus FSMD que materialitza l'EFSM del detector del sentit del gir és el que es mostra en la figura 85.

Figura 85. Circuit seqüencial corresponent al detector del sentit del gir



3. Per al disseny de la PSM cal tenir present que, com a mínim, hi haurà un estat inicial previ al moment en què el sensor de nivell hagi enviat cap dada en què no s'haurà d'activar l'alarma. En altres paraules, mentre no arribi aquesta informació, el controlador estarà apagat o en *off*. En aquest estat, que es pot anomenar OFF, s'hi ha de romandre en tant que no arribi 1 bit de START per l'entrada s, és a dir, mentre $(s = 0)$.

En el moment en què $(s = 1)$, s'ha de fer la lectura dels 4 bits següents, que contenen el percentatge d'ompliment del dipòsit que ha mesurat el sensor. Per a això la màquina passarà a un estat de lectura, READ. En aquest estat s'executarà un programa que acumularà la informació dels bits de dades en una variable que contindrà, finalment, el valor del nivell. El primer pas d'aquest programa consisteix a inicialitzar aquesta variable amb 3 bits a 0 i el bit més significatiu del valor del percentatge (DATA3 de la figura 64) posat com a valor d'unitat, és a dir a fer:

$$L^+ = 000s$$

Els altres passos són similars, ja que consisteixen a decalar a l'esquerra (multiplicar per 2) el valor de L i sumar-hi el bit corresponent:

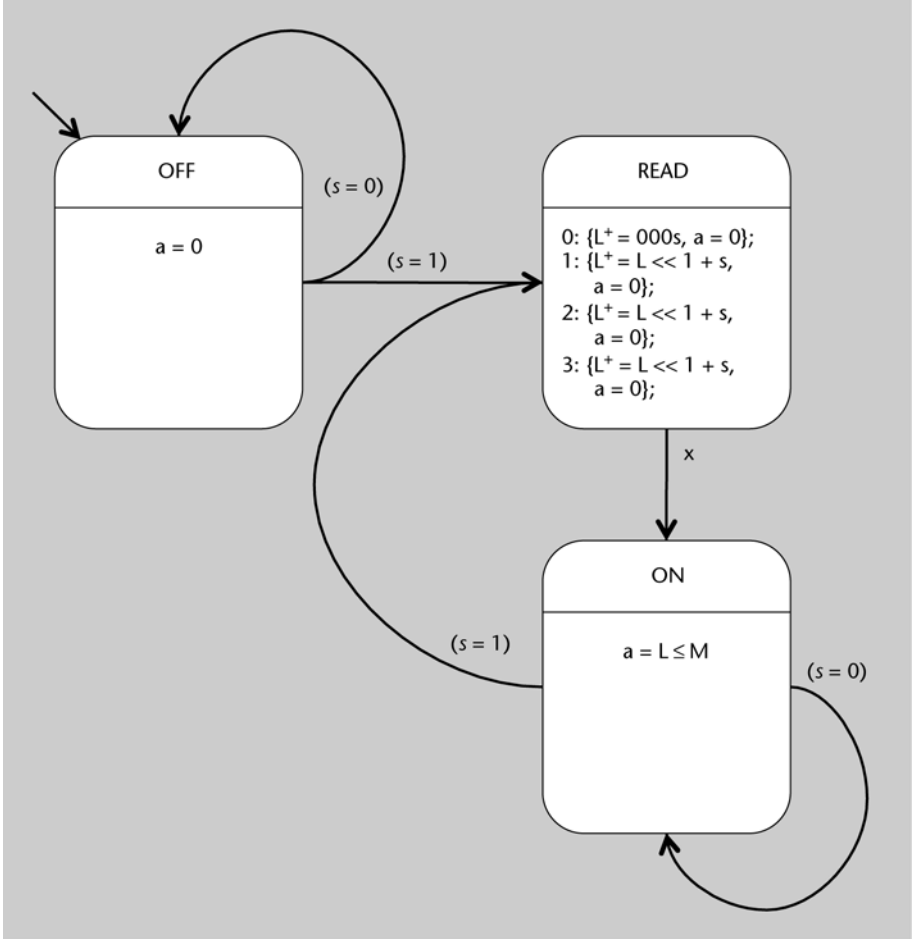
$$L^+ = (L << 1) + s$$

D'aquesta manera, després de tres passos, el bit més significatiu ja és a la posició de més a l'esquerra i el menys significatiu, a la posició de les unitats. Per simplicitat, s'ha optat perquè, durant l'execució del programa de lectura, l'alarma s'aturi. En la pràctica, això no hauria de ser gaire problemàtic, atesa la freqüència de treball dels circuits.

Des de l'estat-programa de lectura, READ, s'ha de passar de manera incondicional a un altre estat en què el controlador estigui actiu i tingui, com a sortida, un 1 o un 0, segons si $L \leq M$ o no. Per aquest motiu, se'l pot anomenar ON. La PSM s'ha de mantenir en aquest estat fins que es rebi una nova dada, és a dir, fins que $(s = 1)$.

Com que hi ha 4 bits de parada a 0 (STOP3, ..., STOP0), no és possible perdre cap lectura o fer una lectura parcial de les sèries de bits que provenen del sensor. De fet, només cal un cicle de rellotge entre l'acabament del programa de READ i un possible nou inici, cosa que evita de posar estats intermedis entre READ i ON. En la figura 86 es mostra la PSM resultant.

Figura 86. PSM del controlador de l'alarma d'avís de nivell mínim



Amb vista a la implementació en un circuit, caldria codificar els tres estats de manera que OFF fos 00 i, a tall d'exemple, READ podria ser 01 i ON, 10. També caldria tenir un comptador intern de 2 bits activat amb un senyal intern, *inc*, que s'hauria de posar a 1 per als valors 0, 1 i 2 de l'estat-programa READ. El senyal de sortida *a* depèn exclusivament de l'estat en què es troba la PSM:

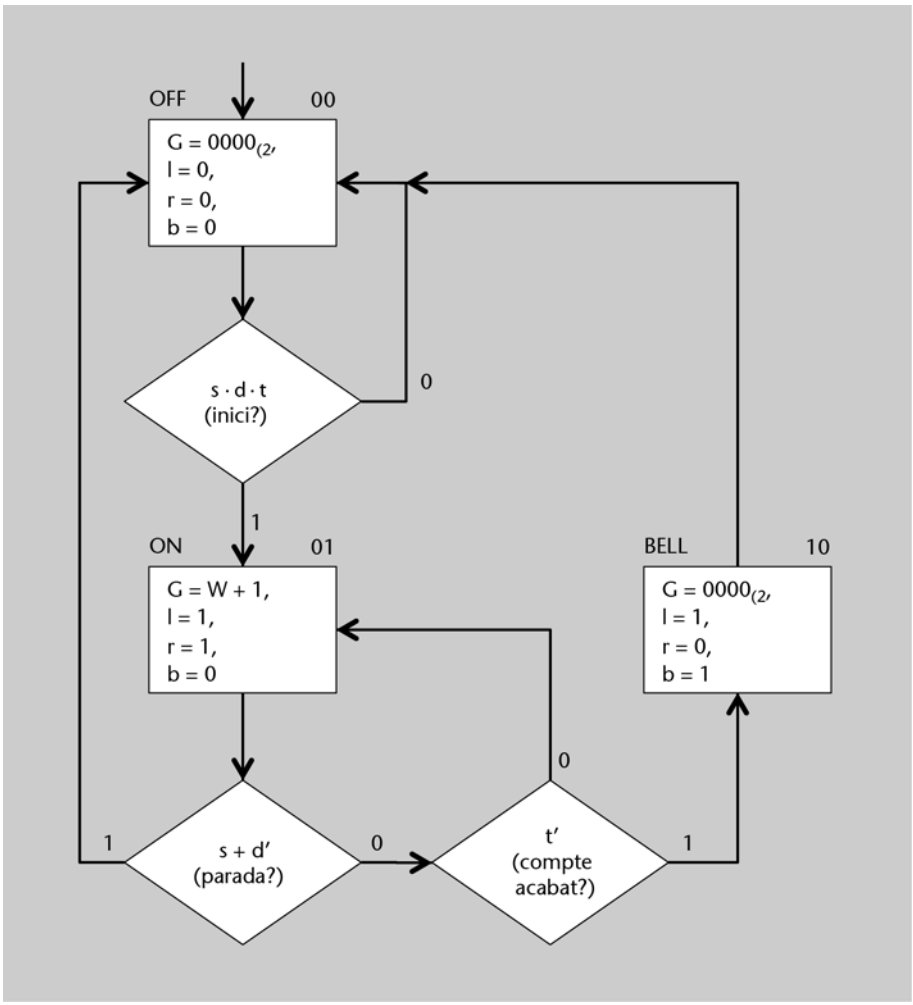
$$a = \text{ON} \cdot (L \leq M)$$

A la unitat de processament hi hauria d'haver el registre *L* i els mòduls combinacionals necessaris per a fer les operacions de decalatge ($L << 1$), suma i comparació. Cal tenir present que s'hauria de seleccionar quin valor s'assigna a *L* amb un multiplexor i, per tant, cal una altra sortida de la unitat de control, que es pot denominar *sL*, per *selector del valor d'entrada de* L.

4. De manera similar al disseny de les màquines d'estats, el procediment per a dissenyar les ASM inclou l'elaboració del graf corresponent a partir d'una caixa d'estat inicial. En aquest cas, el forn ha d'estar apagat (caixa d'estat OFF). Tal com s'indica en l'enunciat, el forn ha d'estar en OFF fins el moment en què l'usuari premi el botó d'inici/aturada ($s = 1$). En aquest moment, si la porta és tancada ($d = 1$) i el temporitzador està en funcionament ($t = 1$), es pot passar a un estat d'activació del forn (caixa d'estat ON).

En l'estat ON, s'ha d'indicar al generador de microones la potència amb què ha de treballar, que és el valor de la potència indicada per l'usuari (de 0 a 3) incrementat en una unitat. El forn ha de restar en aquest estat fins que el temporitzador acabi el compte o se n'interrompi el funcionament obrint la porta o prement el botó d'inici/aturada. En els darrers casos, s'ha de passar directament a la caixa d'estat OFF. En cas que el funcionament s'acabi normalment perquè el temporitzador indiqui la finalització del període de temps ($t = 0$), s'ha de passar a una caixa d'estat (BELL) que ajudi a crear un pols a 1 en la sortida *b* per a fer sonar una alarma d'avís. Els valors que s'atribueixen a la resta de sortides en aquesta caixa d'estat són relativament irrellevants: només seran actius durant un cicle de rellotge. L'ASM corresponent es pot veure en la figura 87.

Figura 87. ASM del controlador d'un forn de microones



En l'esquema de la figura 87 es poden veure les expressions lògiques associades a cada caixa de decisió. En aquest cas, no es fa servir cap variable i, consegüentment, la relació entre caixes de decisió i d'estat és flexible.

Quant a la implementació d'aquest controlador, val a dir que té una unitat de processament molt simple, atès que només hi ha un càlcul $(W + 1)$ i una alternativa de valors per a G: $0000_{(2}$ o $W + 1$. Així doncs, el problema es redueix a implementar la unitat de control que, a més del càlcul de l'estat següent i de les sortides de bit (l, r, b), hauria de tenir un senyal de sortida addicional, sG, que hauria de servir per a controlar el valor a la sortida G.

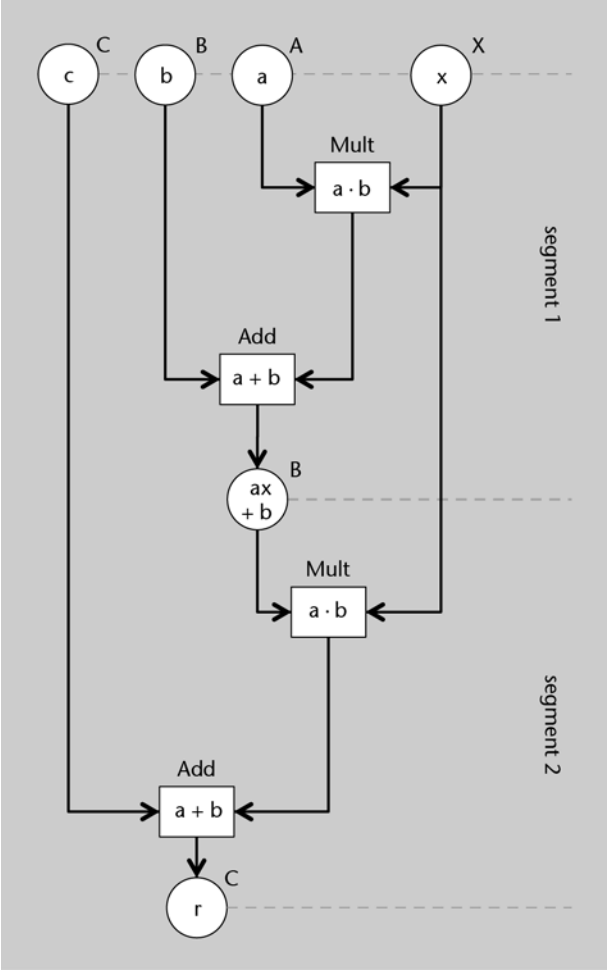
5. Com que es tracta de resoldre el càlcul amb el mínim nombre de recursos, cal reduir el nombre d'operacions al mínim i aprofitar els registres que emmagatzemen les variables d'entrada per a resultats intermedis i per al final. Quant al primer punt, es pot reescriure l'expressió en la forma:

$$(ax + b)x + c$$

D'aquesta manera, no es fa el producte per a calcular x^2 i el nombre d'operacions és mínim. Només cal fer-les gradualment de manera que només en calgui una de cada tipus a cada cicle de rellotge. Això anirà en contra del nombre de cicles de rellotge que seran necessaris per a fer el càlcul, que serà més elevat.

El resultat és l'esquema de càlcul que es mostra en la figura 88, que necessita dos cicles de rellotge. L'esquema s'ha etiquetat de manera que es pot veure una possible assignació de cada node als recursos d'emmagatzematge (registres) i de càlcul corresponents.

Figura 88. Esquema de càlcul per a $ax^2 + bx + c$



6. En l'estat *S*3 es redueix *A* en una unitat perquè l'actualització dels registres *A* i *P* que es fa en l'estat *S*2 té efecte en acabar el cicle en què es calculen els nous valors. Per tant, la condició es fa amb els valors que tenien els registres al principi del cicle, els de la dreta de les expressions que hi ha en el node de processament *S*2. Per tant, en arribar a *S*3, *A* s'actualitza a $A + 1$, essent una unitat més gran que la corresponent al fet de satisfer la condició de ser l'arrel quadrada entera de *C*.

La implementació del circuit es farà separant les parts de control i de processament, segons el model d'FSMD. La unitat de control es pot materialitzar amb una codificació de tipus *one-hot bit*, cosa que permet reproduir directament el diagrama de flux de l'algorisme corresponent: els nodes de processament són biestables i els de decisió, demultiplexors.

El camí de dades segueix l'arquitectura amb multiplexors controlats pel número d'estat a l'entrada dels recursos, tant si són de memòria (registres) com de càlcul. En el cas de la solució que es presenta en la figura 89, hi ha algunes optimitzacions per a reduir les dimensions de l'esquema del circuit:

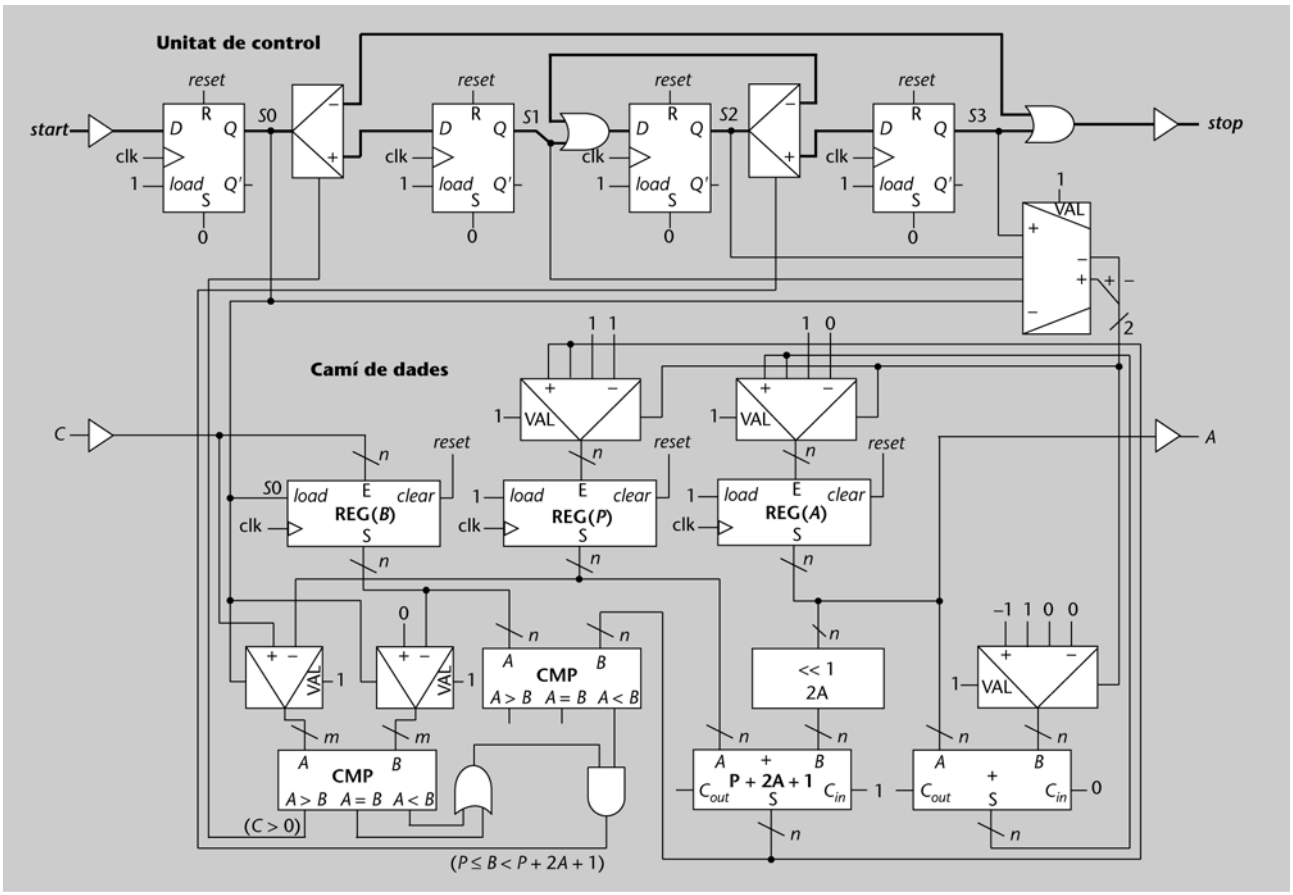
- S'ha eliminat el multiplexor a l'entrada del registre *B* perquè aquest registre només carrega un valor a *S*0 (el contingut del senyal d'entrada *C*).

- S'ha reduït l'ordre dels multiplexors a l'entrada del comparador que s'ocupa de calcular ($C > 0$) i ($P \leq B$), perquè la primera comparació només es fa a *S*0.

- S'han eliminat els multiplexors del càlcul de $P + 2A + 1$ i de $((P \leq B)$ AND $(B < P + 2A + 1))$, perquè només s'aprofiten a *S*2.

Encara hi ha d'altres optimitzacions possibles que es deixen com a ampliació a l'exercici.

Figura 89. Circuit per al càlcul de l'arrel quadrada entera

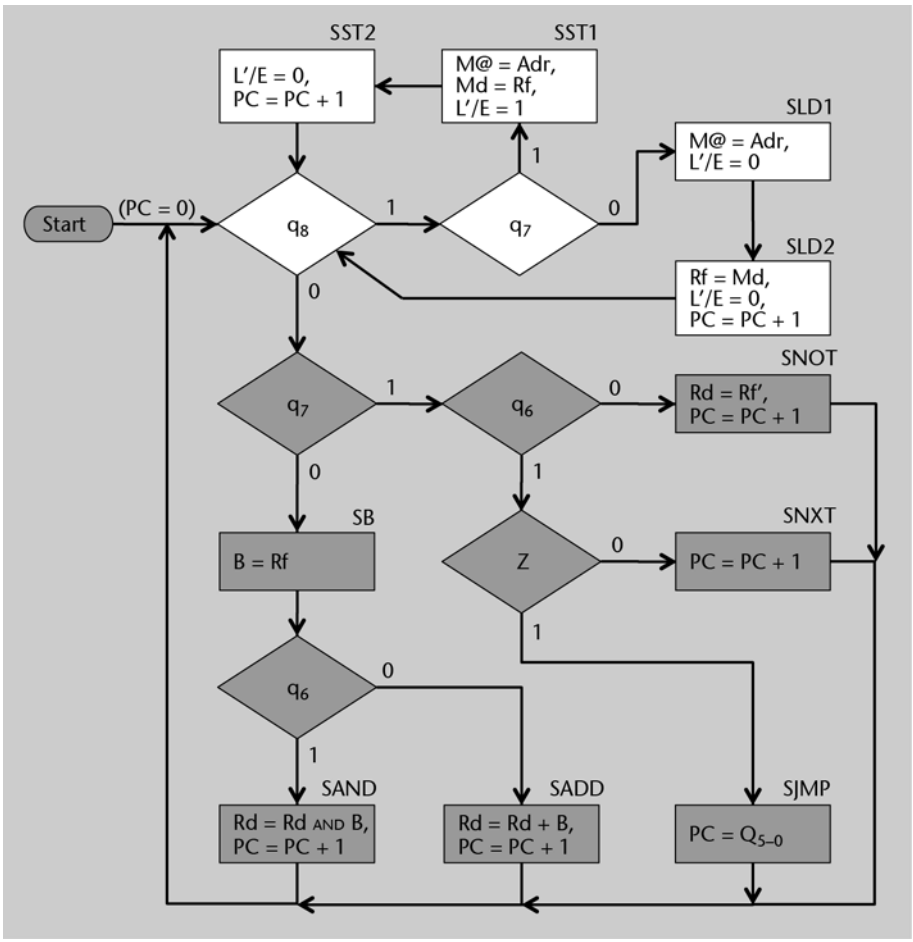


7. L'algorisme d'interpretació no canvia per a les instruccions que ja hi ha en el repertori del Femtoproc, només cal afegir la part corresponent a la interpretació de les noves instruccions. Per a fer-ho, en el diagrama de flux corresponent s'ha de posar un node de decisió just al principi que comprovi el valor del bit més significatiu del codi de la instrucció (q_8). Si és 0, s'enllaça amb el diagrama anterior, els nodes del qual apareixen ombrejats en la figura 90. Si és 1, es passa a interpretar les noves instruccions.

El bit q_7 identifica si la instrucció és de lectura (LOAD) o escriptura (STORE) d'una dada en memòria. Si es tracta d'una lectura, cal que la memòria rebi l'adreça de la posició de memòria que es vol llegir ($M@ = Adr$, en què $Adr = Q_{6-3}$) i que el senyal d'operació de memòria L'/E sigui a 0. Atès que la memòria necessita un cicle de rellotge per a poder obtenir la dada de la posició de memòria donada, cal un estat addicional (SLD2) en el qual ja serà possible llegir de dada de la memòria i emmagatzemar-la en el registre indicat per les posicions 2 a 0 del codi d'instrucció (Rf = Md). La sortida L'/E s'ha de mantenir a 0 en tots els estats per a evitar fer escriptures que puguin alterar el contingut de la memòria de manera indesitjada, per això, s'ha posat que $L'/E = 0$ també a SLD2. En aquest estat, cal incrementar el comptador de programa per a passar a la instrucció següent ($PC = PC + 1$).

Si s'executa una instrucció STORE, $Q_{8-7} = 11_{(2}$, s'ha de passar a un primer estat en què s'indiqui a la memòria a quina adreça s'ha d'accedir ($M@ = Adr$) per a escriure-hi ($L'/E = 1$) i quin contingut s'hi ha de posar (Md = Rf). El node següent (SST2) proporciona l'espera necessària perquè la memòria faci l'escriptura i, a més, serveix per a calcular l'adreça de la instrucció següent ($PC = PC + 1$) a la memòria de programa. De forma similar al que s'ha fet a SLD2, en SST2 s'ha de posar $L'/E = 0$. En aquest sentit, tots els estats del diagrama anterior (SAND, SADD, SJMP, SNOT i SNXT) també han d'incloure $L'/E = 0$, tot i que no es representi en la figura 90 per qüestions de llegibilitat.

Figura 90. Diagrama de flux del Femtoproc ampliat



8. Al camí de dades, el registre d'adreces de memòria (MAR) hauria de tenir 16 bits. Per a no modificar res més, s'hi podria afegir un segon registre de 8 bits, igualment connectat al *BusW*. Així, des del punt de vista del camí de dades, una adreça és formada per 2 bytes, el menys significatiu (el byte baix o *low byte*) s'hauria d'emmagatzemar en el registre MAR_L (l'actual MAR) i el més significatiu, en el nou registre MAR_H. Des del punt de vista de la memòria, els dos registres (MAR_L i MAR_H) es veurien com una unitat, que li proporcionaria una adreça de 16 bits.

La unitat de control hauria d'implementar un cicle d'execució d'instruccions en què les que treballen amb adreces completes de memòria necessitarien una lectura addicional per a llegir el segon byte de l'adreça. En el cas del mode indirecte, serien dues lectures addicionals, ja que cal llegir una segona adreça de memòria. Per tant, l'execució de les instruccions afectades seria més lenta (una solució seria augmentar el nombre de registres del camí de dades per poder reduir el nombre d'accessos a dades en memòria).

Glossari

accés directe a memòria *m* Mecanisme que permet als dispositius perifèrics d'un computador accedir directament a la memòria principal del processador.
en direct memory access
sigla **DMA**

algorisme *m* Mètode per a resoldre un problema que s'especifica com una seqüència de passos finita. Cada pas consisteix o bé en una decisió o bé en un procés que, al seu torn, pot ser un altre algorisme.

algorithmic state-machine *f* Vegeu **màquina d'estats algorísmica**.

ALU *f* Vegeu **unitat aritmeticològica**.

arithmetic logic unit *f* Vegeu **unitat aritmeticològica**.

arquitectura *f* Model de construcció que descriu els components (recursos) d'un sistema i les relacions que han de tenir.

arquitectura Hardvard *f* Arquitectura d'un processador en què les instruccions d'un programa es guarden en una memòria diferenciada de la memòria de les dades que manipula.

arquitectura del repertori d'instruccions *f* Model de codificació de les instruccions en un repertori donat.
en instruction-set architecture
sigla **ISA**

arquitectura de Von Neumann *f* Arquitectura d'un processador en què el programa és emmagatzemat en la memòria.

ASM *f* Vegeu **màquina d'estats algorísmica**.

bus *m* Sistema de comunicació entre dos o més components d'un computador constituït per un canal de comunicació (línies d'interconnexió), un protocol de transmissió (emissió i recepció) de dades i mòduls que el materialitzen.

bus d'adreces *m* Part d'un bus dedicada a la identificació de la ubicació de les dades que es transmeten en el dispositiu o dispositius receptors.

bus de control *m* Part d'un bus que conté senyals de control de la transmissió o, si es vol, dels senyals per a l'execució dels protocols d'intercanvi d'informació.

bus de dades *m* Part d'un bus que s'ocupa de la transmissió de les dades.

cache memory *f* Vegeu **memòria cau**.

camí de dades *m* Circuit compost per una munió de recursos de memòria, que emmagatzemen les dades; de recursos de càlcul, que fan operacions amb les dades, i d'interconnexions, que permeten que les dades es transmetin d'un component a l'altre. S'anomena així perquè el processament de les dades s'hi efectua de manera que les dades segueixen un determinat camí des de les entrades fins a convertir-se en resultats a les sortides.

canonada *f* Vegeu ***pipeline***.

central processing unit *f* Vegeu **unitat central de processament**.

computador *m* Màquina capaç de processar informació de manera automàtica. Per a fer-ho disposa de, com a mínim, un processador i diversos dispositius d'entrada i sortida d'informació.

controlador *m* Circuit que pot actuar sobre una altra entitat (habitualment, un altre circuit) mitjançant canvis en els senyals de sortida. Normalment, aquests canvis són en funció de l'estat intern i dels senyals d'entrada, que solen incloure informació sobre l'entitat que controlen.

core *m* Vegeu **nucli**.

CPU *f* Vegeu **unitat central de processament**.

diagrama de flux *m* Esquema gràfic amb diversos elements que representen les diverses successions d'accions (i, per tant, d'estats) possibles d'un determinat algorisme.

digital-signal processor *m* Vegeu **processador digital de senyal**.

direct memory access *m* Vegeu **accés directe a memòria**.

dispositiu perifèric *m* Component d'un computador que no s'inclou dins del processador.
sin. **perifèric**

dispositiu perifèric d'entrada *m* Component d'un computador que hi permet introduir informació. Habitualment, és format per una part externa al mateix computador (teclat, pantalla tàctil, ratolí, etc.) i una de més interna (el controlador del dispositiu o el mòdul d'entrada corresponent), encara que perifèrica respecte al processador.

dispositiu perifèric d'entrada/sortida *m* Component d'un computador que permet introduir-hi i extraure'n informació. Normalment, és una memòria secundària a la del processador i presenta una arquitectura amb dues parts: una d'externa, com unitats de disc dur o òptic o com els connectors dels mòduls per a memòries USB, i una d'interna, que inclou el controlador del dispositiu o el mòdul d'entrada/sortida corresponent, però que és fora del processador.

dispositiu perifèric de sortida *m* Component d'un computador que permet extraure'n informació. Normalment, la informació és observable en una part externa del mateix computador (pantalla, impressora, altaveu, etc.), tot i que n'hi ha una de més interna (el controlador del dispositiu o el mòdul de sortida corresponent), encara que perifèrica respecte al processador.

DMA *m* Vegeu **accés directe a memòria**.

DSP *m* Vegeu **processador digital de senyal**.

EFSM *f* Vegeu **màquina d'estats finits estesa**.

esquema de càlcul *m* Representació gràfica d'un càlcul en funció dels operands i operadors que conté.

estructura *f* Conjunt de components d'un sistema i manera en què estan relacionats.

extended finite-state machine *f* Vegeu **màquina d'estats finits estesa**.

finite-state machine *f* Vegeu **màquina d'estats finits**.

finite-state machine with data-path *f* Vegeu **màquina d'estats finits amb camí de dades**.

FSM *f* Vegeu **màquina d'estats finits**.

FSMD *f* Vegeu **màquina d'estats finits amb camí de dades**.

GPU *f* Vegeu **unitat de processament de gràfics**.

graphics processing unit *f* Vegeu **unitat de processament de gràfics**.

Hardvard (arquitectura) *f* Vegeu **arquitectura Hardvard.**

instruction-set architecture *f* Vegeu **arquitectura del repertori d'instruccions**.

ISA *f* Vegeu **arquitectura del repertori d'instruccions**.

llenguatge de màquina *m* Llenguatge binari intel·ligible per a una màquina. Normalment, consisteix en una sèrie de paraules binàries que codifiquen les instruccions d'un programa.

màquina algorísmica *f* Model de materialització del maquinari corresponent a un algorisme, habitualment representat amb un diagrama de flux.

màquina d'estats algorísmica *f* Model de representació del comportament d'un circuit seqüencial amb elements diferenciats per als estats i per a les transicions. Permet treballar amb sistemes amb un gran nombre d'entrades.
en algorithmic state-machine
sigla **ASM**

màquina d'estats finits *f* Model de representació d'un comportament basat en un conjunt finit d'estats i de les transicions que s'hi defineixen per a passar d'un a l'altre.
en finite-state machine
sigla **FSM**

màquina d'estats finits estesa *f* Model de representació de comportaments que consisteix en una màquina d'estats finits que inclou càlculs amb dades tant en les transicions com en les sortides associades a cada estat.
en extended finite-state machine
sigla **EFSM**

màquina d'estats finits amb camí de dades *f* Arquitectura que separa la part de càlcul de la de control, que és una màquina d'estats finits definida en termes de funcions lògiques per a les transicions i les sortides associades a cada estat.
en finite-state machine with data-path
sigla **FSMD**

màquina d'estats-programa *f* Model de representació de comportaments de sistemes seqüencials en què cada estat pot implicar l'execució d'un programa.
en program-state machine
sigla **PSM**

MCU *m* Vegeu **microcontrolador**.

memòria cau *f* Memòria interposada en el canal de comunicació de dos components per a fer més efectiva la transmissió de dades. Normalment, n'hi ha entre la CPU i la memòria principal i entre el processador i els perifèrics. En el primer cas, n'hi pot haver unes quantes d'interposades en cascada per a una adaptació més progressiva dels paràmetres de velocitat de treball i de capacitat de memòria.
en cache memory

memòria principal *f* Memòria del processador.

microarquitectura *f* Arquitectura d'un determinat processador.

microcontrolador *m* Processador amb una microarquitectura específica per a executar programes de control. Normalment, es tracta de processadors amb mòduls d'entrada/sortida de dades diversos i nombrosos.
en micro-controller unit
sigla **MCU**

micro-controller unit *m* Vegeu **microcontrolador**.

microinstrucció *f* Cadascuna de les possibles instruccions en un microprograma.

microprograma *m* Programa que executa la unitat de control d'un processador.

nucli *m* Bloc d'un processador (d'un computador) amb capacitat d'executar un procés. Habitualment es correspon amb una CPU.
en core

computador *m* Màquina capaç de processar informació de manera automàtica. Per a això, disposa, al menys, d'un processador i de diversos dispositius d'entrada i sortida d'informació.

perifèric Vegeu **dispositiu perifèric**.

pipeline *m* Encadenament en cascada de diversos segments d'un mateix càlcul per a poder-lo fer en paral·lel.
sin. **canonada**

processador *m* Element capaç de processar informació.

processador digital de senyal *m* Processador construït amb una microarquitectura específica per al processament intensiu de dades.
en digital-signal processor
sigla **DSP**

programa *m* Conjunt d'accions executades en seqüència.

program-state machine *f* Vegeu **màquina d'estats-programa**.

PSM *f* Vegeu **màquina d'estats-programa**.

segment *m* Part d'un càlcul que es du a terme en un mateix període en una cascada de càlculs parcials que porta a un determinat càlcul final.

seqüenciador *m* Màquina algorísmica que s'ocupa d'executar microinstruccions en seqüència, segons un microprograma determinat.

unitat aritmeticològica *f* Recurs de càlcul programable que fa tant operacions de tipus aritmètic, com la suma i la resta, com de tipus lògic, com el producte (conjunció) i la suma (disjunció) lògics.
en arithmetic logic unit
sigla **ALU**

unitat central de processament *f* Part d'un processador que fa el processament de la informació que té una microarquitectura amb memòria segregada. Aquesta unitat normalment té una arquitectura d'FSMD.
en central processing unit
sigla **CPU**

unitat de control *f* Part d'un circuit seqüencial que en controla les operacions. Habitualment, s'ocupa de calcular l'estat següent de la màquina d'estats de la qual en representa el funcionament i els senyals de control per a la resta del circuit.

unitat de procés *f* Part d'un circuit seqüencial que s'ocupa del processament de les dades. Habitualment, és la part que fa les operacions amb les dades tant per a determinar condicions de transició de la part de control com resultats de càlculs de sortida del circuit en conjunt.
sin. **unitat operacional**

unitat de processament de gràfics *f* DSP adaptat al processament de gràfics. Normalment, amb microarquitectura paral·lela.
en graphics processing unit
sigla **GPU**

unitat operacional *f* Vegeu **unitat de procés**.

variable *f* Element dels models de màquines d'estats que emmagatzema informació complementària als estats. Normalment, hi ha una variable per dada no directament relacionada amb l'estat i cada variable s'associa amb un registre a l'hora de materialitzar la màquina corresponent.

Von Neumann (arquitectura de) *f* Vegeu **arquitectura de Von Neumann**.

Bibliografia

Lloris Ruiz, A.; Prieto Espinosa, A.; Parrilla Roure, L. (2003). *Sistemas Digitales.* Madrid: McGraw-Hill.

Morris Mano, M. (2003). *Diseño Digital*. Madrid: Pearson-Education.

Ribas i Xirgo, Ll. (2000). *Pràctiques de fonaments de computadors*. Bellaterra (Cerdanyola del Vallès): Servei de publicacions de la UAB.

Ribas i Xirgo, Ll. i altres (2010). "La robótica como elemento motivador para un proyecto de asignatura en Fundamentos de Computadores" A: *Actas de las XVI Jornadas de Enseñanza Universitaria de la Informática (JENUI)* (pàg. 171-178). Santiago de Compostella.

Ribas i Xirgo, Ll. (2010, octubre). "Yet Another Simple Processor (YASP) for Introductory Courses on Computer Architecture". *IEEE Trans. on Industrial Electronics* (vol. 57, núm. 10, 3317-3323). Piscataway (Nova Jersey, EUA): IEEE.

Roth, Jr., Ch. H. (2004). *Fundamentos de diseño lógico*. Madrid: Thomson-Paraninfo.