El processador

Miquel Albert Orenga
Gerard Enrique Manonellas

PID_00218256

UOC
Universitat Oberta de Catalunya

Índex

Introducció

En aquest mòdul estudiarem el component principal d'un computador: el processador, unitat central de procés o CPU (sigles de l'expressió anglesa *central processing unit*).

La funció principal que té és processar les dades i transferir-les als altres elements del computador. Aquestes tasques es duen a terme mitjançant l'execució d'instruccions. Per aquest motiu l'objectiu principal a l'hora de dissenyar un processador és aconseguir que les instruccions s'executin de la manera més eficient possible.

En aquest mòdul ens centrarem a analitzar els elements principals del processador des del punt de vista funcional i no entrarem a analitzar els aspectes relacionats amb la millora de rendiment.

Els elements bàsics del processador que estudiarem són els següents:

- Conjunt de registres.
- Unitat aritmètica i lògica.
- Unitat de control.

Del conjunt de registres en descriurem la funció principal i el tipus d'informació que poden emmagatzemar.

De la unitat aritmètica i lògica (ALU) en veurem la funció principal i el tipus de dades amb què es treballa habitualment. No entrarem en més detall perquè les operacions aritmètiques i lògiques ja s'han estudiat àmpliament en assignatures prèvies d'aquests estudis.

Finalment farem un estudi detallat de la unitat de control en què veurem que és l'element clau per al funcionament correcte del processador i l'execució de les instruccions.

Objectius

Amb l'estudi d'aquest mòdul es pretén que l'estudiant assoleixi els objectius següents:

1. Entendre l'organització d'un processador, pel que fa les unitats funcionals que el componen: registres, ALU i unitat de control.
2. Entendre com executa una instrucció un processador.
3. Conèixer l'organització del conjunt de registres del processador.
4. Veure els conceptes bàsics relacionats amb l'ALU.
5. Entendre el funcionament de la unitat de control del processador.
6. Conèixer les idees clau per a dissenyar una unitat de control.
7. Saber les diferències principals entre computadors RISC i CISC.

1. Organització del processador

La funció principal d'un processador és executar instruccions i l'organització que té és condicionada per les tasques que cal fer i com es fan.

Els processadors estan dissenyats i operen segons un senyal de sincronització. Aquest senyal, conegut com a *senyal de rellotge*, és un senyal en forma d'ona quadrada periòdica amb una determinada freqüència. Totes les operacions fetes pel processador les governa aquest senyal de rellotge: un cicle de rellotge determina la unitat bàsica de temps, és a dir, la durada mínima d'una operació del processador.

Freqüència del senyal de rellotge

La freqüència del senyal de rellotge es defineix com el nombre d'impulsos per unitat de temps, es mesura en cicles per segon o hertzs (Hz) i determina la velocitat d'operació del processador.

Per a executar una instrucció són necessaris un cicle o més d'un cicle de rellotge, depenent del tipus d'instrucció i dels operands que tingui.

Les prestacions del processador no les determina només la freqüència de rellotge sinó altres característiques del processador, especialment del disseny del joc d'instruccions i la capacitat que té per a executar simultàniament múltiples instruccions.

Per a executar les instruccions, tot processador disposa de tres components principals:

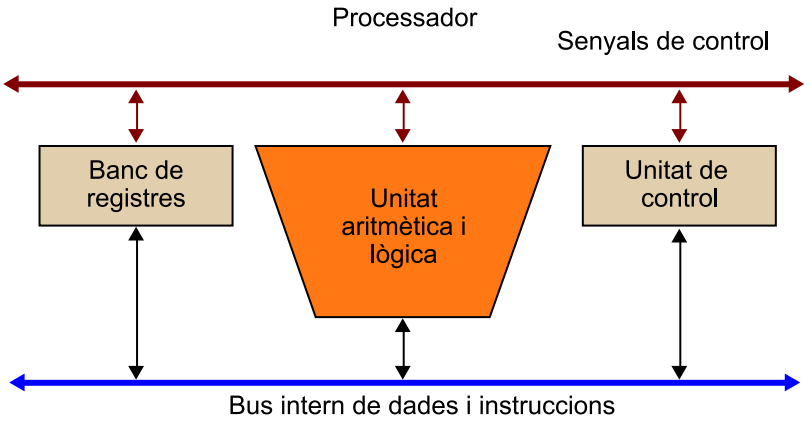
1) Un **conjunt de registres**: espai d'emmagatzematge temporal de dades i instruccions dins del processador.

2) Unitat aritmètica i lògica o ALU[1]: circuit que fa un conjunt d'operacions aritmètiques i lògiques amb les dades emmagatzemades dins del processador.

3) Unitat de control: circuit que controla el funcionament de tots els components del processador. Controla el moviment de dades i instruccions dins i fora del processador i també les operacions de l'ALU.

(1)*ALU* són les sigles de l'expressió anglesa *arithmetic logic unit*.

L'organització bàsica dels elements que componen un processador, i el flux d'informació entre els diferents elements, es veu en l'esquema següent:



Com s'observa, a part dels tres components principals, és necessari disposar d'un sistema que permeti interconnectar aquests components. Aquest sistema d'interconnexió és específic per a cada processador. Distingim dos tipus de línies d'interconnexió: línies de control que permeten governar el processador i línies de dades que permeten transferir les dades i les instruccions entre els diferents components del processador. Aquest sistema d'interconnexió ha de disposar d'una interfície amb el bus del sistema.

El terme *processador* actualment es pot entendre com a microprocessador perquè totes les unitats funcionals que formen el processador es troben dins d'un xip, però cal tenir present que, per l'augment de la capacitat del nivell d'integració, dins els microprocessadors s'hi poden trobar altres unitats funcionals del computador. Per exemple:

- **Unitats d'execució SIMD**: unitats especialitzades en l'execució d'instruccions SIMD (*single instruction, multiple data*), instruccions que treballen amb estructures de dades vectorials, com per exemple instruccions multimèdia.

- **Memòria cau**: pràcticament tots els processadors moderns incorporen dins del mateix xip del processador alguns nivells de memòria cau.

- **Unitat de gestió de memòria** o **memory management unit (MMU)**: gestiona l'espai d'adreces virtuals, traduint les adreces de memòria virtual a adreces de memòria físiques en temps d'execució. Aquesta traducció permet protegir l'espai d'adreces d'un programa de l'espai d'adreces d'altres programes i també permet separar l'espai de memòria del sistema operatiu de l'espai de memòria dels programes d'usuari.

- **Unitat de punt flotant** o **floating point unit (FPU)**: unitat especialitzada a fer operacions en punt flotant, pot funcionar de manera autònoma ja que disposa del seu propi conjunt de registres.

2. Cicle d'execució de les instruccions

El cicle d'execució és la seqüència d'operacions que es fa per a executar cadascuna de les instruccions. El dividim en quatre fases principals:

1) Lectura de la instrucció.

2) Lectura dels operands font.

3) Execució de la instrucció i emmagatzematge de l'operand de destinació.

4) Comprovació d'interrupcions.

Ordre d'operacions

Les fases principals del cicle d'execució són comunes a la majoria de computadors actuals i les diferències principals es troben en l'ordre en què es fan algunes de les operacions de cada fase o en què es fan algunes de les operacions en altres fases.

En cada fase s'inclou una operació o diverses operacions que cal fer. Per a dur a terme aquestes operacions, anomenades *microoperacions*, cal una complexa gestió dels senyals de control i de la disponibilitat dels diferents elements del processador, tasca que fa la unitat de control.

Vegem les operacions principals que es fan habitualment en cadascuna de les cinc fases del cicle d'execució de les instruccions:

Fase 1. Lectura de la instrucció. Les operacions que es fan en aquesta fase són les següents:

a) Llegir la instrucció. Quan llegim la instrucció, el registre comptador del programa (PC) ens indica l'adreça de memòria on hi ha la instrucció que hem de llegir. Si la mida de la instrucció és superior a la paraula de memòria, s'han de fer tants accessos a memòria com calgui per a llegir-la completament i carregar tota aquesta informació en el registre d'instrucció (IR).

b) Descodificar la instrucció. S'identifiquen les diferents parts de la instrucció per a determinar quines operacions cal fer en cada fase del cicle d'execució. Aquesta tasca la fa la unitat de control del processador llegint la informació que hem carregat en el registre d'instrucció (IR).

c) Actualitzar el comptador del programa. El comptador del programa s'actualitza segons la mida de la instrucció, és a dir, segons el nombre d'accessos a memòria que hem fet per llegir la instrucció.

Actualització del PC

L'actualització del comptador del programa és una operació que es pot trobar en altres fases de l'execució de la instrucció.

Les operacions d'aquesta fase són comunes als diferents tipus d'instruccions que trobem en el joc d'instruccions.

Fase 2. Lectura dels operands font. Aquesta fase s'ha de repetir per a cadascun dels operands font que tingui la instrucció.

Les operacions que cal fer en aquesta fase depenen del mode d'adreçament que tinguin els operands: per als més simples, com ara l'immediat o el directe a registre, no cal fer cap operació; per als indirectes o els relatius, cal fer càlculs i accessos a memòria. Si l'operand font és implícit, anem a buscar la dada en el lloc predeterminat per aquella instrucció.

Vegeu també

Els modes d'adreçament s'expliquen en l'apartat 2 del mòdul "Joc d'instruccions".

És fàcil adonar-se que donar molta flexibilitat en els modes d'adreçament que podem utilitzar en cada instrucció pot afectar molt el temps d'execució d'una instrucció.

Per exemple, si fem un ADD R3,3 no cal fer cap càlcul ni cap accés a memòria per a obtenir els operands; en canvi, si fem un ADD [R1], [R2+16], cal fer una suma i dos accessos a memòria.

Fase 3. Execució de la instrucció i emmagatzemament de l'operand de destinació (resultat)

Les operacions que cal fer en aquesta fase depenen del codi d'operació de la instrucció i del mode d'adreçament que tingui l'operand destinació.

Execució de la instrucció

Les operacions que es fan són diferents per a cada codi d'operació. Durant l'execució, a més a més d'obtenir el resultat de l'execució de la instrucció, es poden modificar els bits de resultat de la paraula d'estat del processador.

Emmagatzemament de l'operand de destinació (resultat)

La funció bàsica és recollir el resultat obtingut durant l'execució i guardar-lo en el lloc indicat per l'operand, llevat que l'operand sigui implícit, cas en què es guarda en el lloc predeterminat per aquella instrucció. L'operand de destinació pot ser un dels operands font; d'aquesta manera no cal repetir els càlculs per a obtenir l'adreça de l'operand.

Fase 4. Comprovació d'interrupcions

Les interrupcions són el mecanisme pel qual un dispositiu extern al processador pot interrompre el programa que està executant el processador, per tal d'executar un altre programa (una rutina de servei a la interrupció o RSI) per donar servei al dispositiu que ha produït la interrupció.

La petició d'interrupció es fa activant alguna de les línies de petició de què disposa el processador.

Vegeu també

El concepte d'*interrupció* s'explica en detall en l'apartat 3 dins del mòdul "Sistema d'entrada/sortida".

En aquesta fase es verifica si s'ha activat alguna línia de petició d'interrupció del processador en el transcurs de l'execució de la instrucció. Si no se n'ha activat cap, continuem el procés normalment; és a dir, s'acaba l'execució de la instrucció en curs i comença l'execució de la instrucció següent. En cas contrari, cal transferir el control del processador a la rutina de servei d'interrupció que ha d'atendre aquesta interrupció i fins que no s'acabi no podem continuar l'execució de la instrucció en curs.

Per a atendre la interrupció cal dur a terme algunes operacions (intentarem emmagatzemar la mínima informació perquè el procés sigui tan ràpid com es pugui) per a transferir el control del processador a la rutina de servei d'interrupcions de manera que, després, el puguem recuperar amb la garantia que el processador estarà en el mateix estat que estava abans de transferir-ne el control a la rutina de servei d'interrupció:

- Emmagatzemar el comptador de programa i la paraula d'estat (generalment, es guarden a la pila).
- Emmagatzemar l'adreça on comença la rutina per a atendre la interrupció al comptador de programa.
- Executar la rutina de servei d'interrupció.
- Recuperar el comptador de programa i la paraula d'estat.

Aquesta fase actualment és present en tots els computadors, ja que tots utilitzen E/S per a interrupcions i E/S per a DMA.

Vegeu també

En el mòdul didàctic "Sistema d'entrada/sortida" veurem que els ordinadors requereixen línies de petició d'interrupció que també es poden utilitzar per a altres tasques que no són específiques d'E/S.

2.1. Segmentació de les instruccions

La segmentació de les instruccions (*pipeline*) consisteix a dividir el cicle d'execució de les instruccions en un conjunt d'etapes. Aquestes etapes poden coincidir o no amb les fases del cicle d'execució de les instruccions.

L'objectiu de la segmentació és executar simultàniament diferents etapes de diverses instruccions, cosa que permet augmentar el rendiment del processador sense haver de fer més ràpides cadascuna de les unitats del processador (ALU, UC, busos, etc.) i sense haver-les de duplicar.

La divisió de l'execució d'una instrucció en diferents etapes s'ha de fer de manera que cada etapa tingui la mateixa durada, generalment un cicle de rellotge. És necessari afegir registres per a emmagatzemar els resultats intermedis entre les diferents etapes, de manera que la informació generada en una etapa sigui disponible per a l'etapa següent.

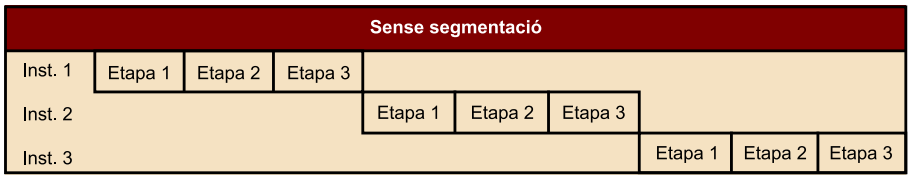
Exemple de segmentació d'instruccions

La segmentació és com una cadena de muntatge. En cada etapa de la cadena es du a terme una part de la feina total i quan s'acaba la feina en una etapa el producte passa a la següent i així successivament fins a arribar al final.

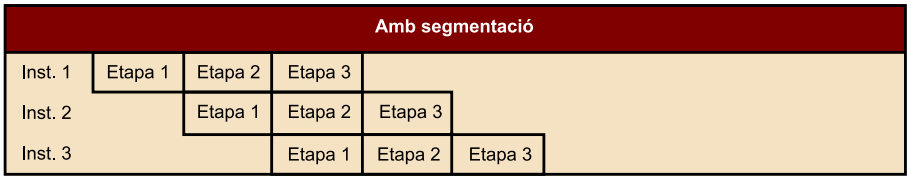
Si hi ha *N* etapes, es pot treballar sobre *N* productes al mateix temps i, si la cadena està ben equilibrada, sortirà un producte acabat en el temps que es triga a dur a terme una de les etapes: no es redueix el temps que es triga a fer un producte, sinó que es redueix el temps total necessari per a fer una determinada quantitat de productes perquè les operacions de cada etapa es fan simultàniament.

Si considerem que el producte és una instrucció i les etapes són cadascuna de les fases d'execució de la instrucció, que anomenem *etapa de segmentació*, hem identificat els elements i el funcionament de la segmentació.

Vegem-ho gràficament. Presentem un mateix procés amb instruccions de tres etapes, en què cada etapa triga el mateix temps a executar-se, resolt sense segmentació i amb segmentació:



Cal nou etapes per a executar les tres instruccions.



Cal cinc etapes per a executar les tres instruccions.

S'ha reduït el temps total que es triga a executar les tres instruccions però no el temps que es triga a executar cadascuna de les instruccions.

Si es vol implementar la segmentació per a millorar el rendiment d'un processador, cal tenir en compte algunes qüestions que implicaran fer canvis en el disseny del processador. Les més importants són les següents:

- Com s'ha de fer perquè les etapes estiguin ben equilibrades (que totes tinguin la mateixa durada)?

- Quins recursos són necessaris a cada etapa perquè totes es puguin executar simultàniament?

- Com cal tractar les instruccions de salt per a minimitzar els efectes en el rendiment de la segmentació?

Nota

Aquestes qüestions no són gens senzilles de contestar, però no és l'objectiu d'aquest curs entrar en el detall d'aquesta problemàtica. Només pretenem que entengueu el funcionament de l'execució de les instruccions utilitzant segmentació.

3. Registres

Els registres són bàsicament elements de memòria d'accés ràpid que hi ha dins el processador. Són un espai de treball pel processador, s'utilitzen com un espai d'emmagatzematge temporal. S'implementen utilitzant elements de memòria RAM estàtica (*static RAM*). Són imprescindibles per a executar les instruccions, entre altres motius, perquè l'ALU només treballa amb els registres interns del processador.

El conjunt de registres i l'organització que tenen és diferent d'un processador a un altre, i difereixen en el nombre de registres, en el tipus de registres i en la mida de cada registre.

Organització de registres

Per a veure com s'organitzen els registres i com s'utilitzen en una arquitectura concreta, consulteu els mòduls "Programació en assemblador (x86-64)" i "Arquitectura CISCA".

Una part dels registres poden ser visibles al programador d'aplicacions, altres només a instruccions privilegiades i altres només s'utilitzen en el funcionament intern del processador.

Una possible classificació dels registres del processador és la següent:

- Registres de propòsit general.
- Registres d'instrucció.
- Registres d'accés a memòria.
- Registres d'estat i de control.

3.1. Registres de propòsit general

Els registres de propòsit general són els registres que s'acostumen a utilitzar com a operands en les instruccions assemblador. Aquests registres es poden assignar a funcions concretes: dades o adreçament. En alguns processadors tots els registres es poden utilitzar per a totes les funcions.

Els registres de dades es poden diferenciar pel format i mida de les dades que emmagatzemen; per exemple, hi pot haver registres per nombres enters i per nombres en punt flotant.

Els registres d'adreçament s'utilitzen per a accedir a memòria i poden emmagatzemar adreces o índexs. Alguns d'aquests registres s'utilitzen de manera implícita per a diferents funcions, com per exemple accedir a la pila, adreçar segments de memòria o fer de suport a la memòria virtual.

3.2. Registres d'instrucció

Els dos registres principals relacionats amb l'accés a les instruccions són:

- ***Program counter*** **(PC)**: registre comptador de programa, conté l'adreça de la instrucció següent que s'ha de llegir de la memòria.

- ***Instruction register*** **(IR)**: registre d'instrucció, conté la instrucció que s'ha d'executar.

3.3. Registres d'accés a memòria

Hi ha dos registres necessaris per a qualsevol operació de lectura o escriptura a memòria:

- ***Memory address register*** **(MAR)**: registre d'adreces de memòria, on posem l'adreça de memòria a què volem accedir.

- ***Memory buffer register*** **(MBR)**: registre de dades de memòria, registre on la memòria diposita la dada llegida o la dada que volem escriure.

La manera de fer un accés a memòria utilitzant aquests registres és la següent:

1) En una **operació de lectura**, es fa la seqüència d'operacions següent:

a) El processador carrega en el registre MAR l'adreça de la posició de memòria que es vol llegir.

b) El processador col·loca en les línies d'adreces del bus el contingut del MAR i activa el senyal de lectura de la memòria.

c) El MBR es carrega amb la dada obtinguda de la memòria.

2) En una **operació d'escriptura**, es fa la seqüència d'operacions següent:

a) El processador carrega en el registre MBR la paraula que vol escriure en la memòria.

b) El processador carrega en el registre MAR l'adreça de la posició de memòria on es vol escriure la dada.

c) El processador col·loca en les línies d'adreces del bus el contingut del MAR, i en les línies de dades del bus, el contingut del MBR, i activa el senyal d'escriptura de la memòria.

3.4. Registres d'estat i de control

La informació sobre l'estat del processador pot estar emmagatzemada en un registre o en més d'un, tot i que habitualment sol ser un únic registre anomenat *registre d'estat*.

Els bits del registre d'estat són modificats pel processador com a resultat de l'execució d'alguns tipus d'instruccions, com per exemple instruccions aritmètiques o lògiques, o com a conseqüència d'algun esdeveniment, com les peticions d'interrupció. Aquests bits són parcialment visibles al programador, en alguns casos mitjançant l'execució d'instruccions específiques.

Cada bit o conjunt de bits del registre d'estat indica una informació concreta. Els més habituals són:

- **Bit de zero**: s'activa si el resultat obtingut és 0.
- **Bit de transport**: s'activa si en el darrer bit que operem en una operació aritmètica es produeix transport; també pot ser degut a una operació de desplaçament.
- **Bit de sobreeiximent**: s'activa si l'última operació ha produït un resultat que no es pot representar en el format que estem utilitzant.
- **Bit de signe**: s'activa si el resultat obtingut és negatiu.
- **Bit d'interrupció**: indica si les interrupcions estan habilitades o inhibides.
- **Bit de mode d'operació**: indica si la instrucció s'executa en mode supervisor o en mode usuari. Hi ha instruccions que només s'executen en mode supervisor.
- **Nivell d'execució**: indica el nivell de privilegi d'un programa en execució. Un programa pot desallotjar el programa que s'executa actualment si el nivell de privilegi que té és superior.

Els *registres de control* són els que depenen més de l'organització del processador. En aquests registres s'emmagatzema la informació generada per la unitat de control i també informació específica per al sistema operatiu. La informació emmagatzemada en aquests registres no és mai visible al programador d'aplicacions.

4. Unitat aritmètica i lògica

La unitat aritmètica i lògica o ALU[2] és un circuit combinacional capaç de fer operacions aritmètiques i lògiques amb nombres enters i també amb nombres reals. Les operacions que pot fer són definides pel conjunt d'instruccions aritmètiques i lògiques de què disposa el joc d'instruccions del computador.

[(2)] *ALU* és l'abreviatura d'*arithmetic logic unit*.

Vegem primer com es representen els valors dels nombres enters i reals amb què pot treballar l'ALU i a continuació quines són les operacions que pot fer.

1) Nombres enters. Els nombres enters es poden representar utilitzant diferents notacions, entre les quals hi ha signe magnitud, complement a 1 i complement a 2. La notació més habitual dels computadors actuals és el complement a 2 (Ca2).

Representació de la informació

En aquest mòdul no analitzarem en detall la representació de nombres ni la manera d'operar-hi. Aquest tema l'hem tractat en profunditat en el mòdul "Representació de la informació" de l'assignatura de *Fonaments de computadors*.

Totes les notacions representen els nombres enters en binari. Segons la capacitat de representació de cada computador s'utilitzen més o menys bits. El nombre de bits més habitual en els computadors actuals és de 32 i 64.

2) Nombres reals. Els nombres reals es poden representar bàsicament de dues maneres diferents: en punt fix i en punt flotant. El computador és capaç de treballar amb nombres reals representats en qualsevol de les dues maneres.

En la **notació en punt fix** la posició de la coma binària és fixa i s'utilitza un nombre concret de bits tant per a la part entera com per a la part decimal.

En la **notació en punt flotant** es representen utilitzant tres camps, això és, signe, mantissa i exponent, on el valor del nombre és:

$$\pm \text{ mantissa} \cdot 2^{\text{exponent}}$$

L'IEEE ha definit una norma per a representar nombres reals en punt flotant: IEEE-754. La norma defineix diferents formats de representació de nombres binaris en punt flotant. Els més habituals són els següents:

- **Precisió simple**: nombres binaris en punt flotant de 32 bits, utilitzen un bit de signe, 8 bits per a l'exponent i 23 per a la mantissa.

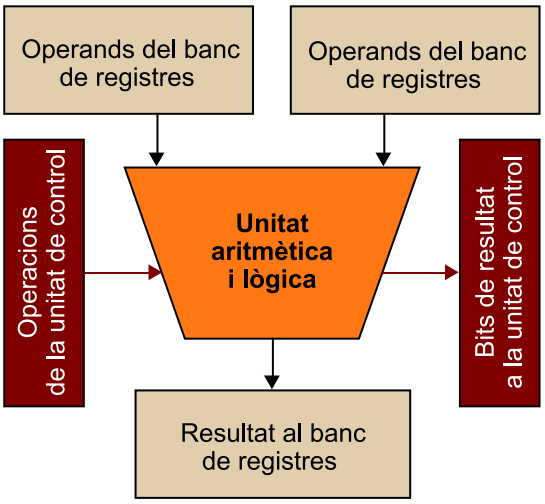
- **Doble precisió**: nombres binaris en punt flotant de 64 bits, utilitzen un bit de signe, 11 bits per a l'exponent i 52 per a la mantissa.

- **Quàdruple precisió**: nombres binaris en punt flotant de 128 bits, utilitzen un bit de signe, 15 bits per a l'exponent i 112 per a la mantissa.

La norma defineix també la representació del zero i de valors especials com infinit i NaN (*not a number*).

Les operacions aritmètiques habituals que pot fer una ALU inclouen suma, resta, multiplicació i divisió. A més a més s'hi poden incloure operacions específiques d'increment positiu (+1) o negatiu (–1).

Dins les operacions lògiques s'inclouen operacions AND, OR, NOT, XOR, operacions de desplaçament de bits a l'esquerra i a la dreta i operacions de rotació de bits.



En els primers computadors s'implementava l'ALU com una única unitat funcional capaç de fer les operacions descrites anteriorment sobre nombres enters. Aquesta unitat tenia accés als registres on s'emmagatzemaven els operands i els resultats de cada operació.

Per a fer operacions en punt flotant, s'utilitzava una unitat específica anomenada *unitat de punt flotant*[3] o *coprocessador matemàtic*, que disposava dels seus propis registres i estava separada del processador.

[3] En anglès, *floating point unit* (FPU).

L'evolució dels processadors que poden executar les instruccions encavalcadament ha fet que el disseny de l'ALU sigui més complex, de manera que ha fet necessari replicar les unitats de treball amb enters per a permetre executar diverses operacions aritmètiques en paral·lel i, d'altra banda, les unitats de punt flotants continuen implementant en unitats separades però ara dins del processador.

5. Unitat de control

La unitat de control es pot considerar el cervell del computador. Com el cervell, està connectada a la resta de components del computador mitjançant els senyals de control (el sistema nerviós del computador). Amb aquest símil no es pretén humanitzar els computadors, sinó il·lustrar que la unitat de control és imprescindible per a coordinar els diferents elements que té el computador i fer-ne un bon ús.

És molt important que un computador tingui unitats funcionals molt eficients i ràpides, però si no es coordinen i es controlen correctament, no es poden aprofitar totes les potencialitats que s'havien previst en el disseny.

Conseqüentment, moltes vegades, en implementar una unitat de control, es fan evidents les relacions que hi ha entre les diferents unitats del computador, ens adonem que cal redissenyar-les, no per a millorar el funcionament concret de cadascuna d'aquestes unitats, sinó per a millorar el funcionament global del computador.

La funció bàsica de la unitat de control és l'execució de les instruccions, però la complexitat del disseny que té no es deu a la complexitat d'aquestes tasques (que en general són molt senzilles), sinó a la sincronització que se n'ha de fer.

A banda de veure les maneres més habituals d'implementar una unitat de control, n'analitzarem el comportament dinàmic, que és clau en l'eficiència i la rapidesa d'un computador.

5.1. Microoperacions

Com ja sabem, executar un programa consisteix a executar una seqüència d'instruccions, i cada instrucció es duu a terme mitjançant un cicle d'execució que consta de les fases principals següents:

1) Lectura de la instrucció.
2) Lectura dels operands font.
3) Execució de la instrucció i emmagatzematge de l'operand de destinació.
4) Comprovació d'interrupcions.

Cadascuna de les operacions que fem durant l'execució d'una instrucció l'anomenem *microoperació*, i aquestes microoperacions són la base per a dissenyar la unitat de control.

5.1.1. Tipus de microoperacions

La funció bàsica de les microoperacions és la transferència d'informació d'un lloc a l'altre del computador, generalment d'un registre a un altre, tant si són interns al processador com externs. Aquest procés de transferència pot implicar només moure la informació, i també transformar-la. Identifiquem tres tipus bàsics de microoperacions:

1) Transferència interna: operacions de transferència entre registres interns del processador.

2) Transferència interna amb transformació: operacions aritmètiques o lògiques utilitzant registres interns del processador.

3) Transferència externa: operacions de transferència entre registres interns del processador i registres externs al processador o mòduls externs al processador (com el bus del sistema o la memòria principal).

Exemples de transferència

Una transferència interna pot ser carregar el contingut del registre PC al registre MAR per obtenir la instrucció següent que hem d'executar; una transferència interna amb transformació d'informació pot ser incrementar un registre, portant el contingut del registre a l'ALU i recollir el resultat per guardar-lo a un altre registre; i una transferència externa pot ser portar el contingut d'un registre d'estat d'un dispositiu d'E/S a un registre del processador.

La nomenclatura que utilitzarem per a denotar les microoperacions és la següent:

Registre de destinació ← Registre d'origen

Registre de destinació ← Registre d'origen <operació> Registre d'origen / Valor

Nota

Els registres, tant d'origen com de destinació, poden ser també de mòduls externs al processador.

5.1.2. Cicle d'execució

Les microoperacions serveixen de guia per a dissenyar la unitat de control, però abans d'entrar en el detall de la implementació, analitzarem la seqüència de microoperacions que habitualment es produeixen en cada fase del cicle d'execució de les instruccions.

Aquesta seqüència pot variar d'una arquitectura a una altra i, fins i tot, hi pot haver microoperacions que estiguin en fases diferents. Això depèn en bona part de les característiques de l'arquitectura: nombre de busos, a quins busos tenen accés els diferents registres, si hi ha unitats funcionals específiques com a registres que es puguin autoincrementar sense fer ús de l'ALU, manera d'accedir als elements externs al processador, etc.

A continuació veurem les microoperacions que es duen a terme en cadascuna de les fases del cicle d'execució per a una arquitectura genèrica des del punt de vista funcional: quines s'han de fer i en quin ordre. En el proper apartat analitzarem amb més detall la dependència temporal entre les microoperacions per raó dels recursos que ha utilitzat cadascuna.

Lectura de la instrucció

La fase de lectura de la instrucció consta bàsicament de quatre passos:

1) MAR ← PC: es posa el contingut del registre PC al registre MAR.

2) MBR ← Memòria: es llegeix la instrucció.

3) PC ← PC + Δ: s'incrementa el PC tantes posicions de memòria com s'han llegit (Δ posicions).

4) IR ← MBR: es carrega la instrucció en el registre IR.

Cal tenir present que si la instrucció té una mida superior a una paraula de memòria aquest procés s'ha de repetir tantes vegades com calgui.

Les diferències principals que trobem entre diferents màquines en aquesta fase són com i quan s'incrementa el PC, ja que en algunes màquines s'utilitza l'ALU i en d'altres es pot fer servir un circuit incrementador específic per al PC.

La informació emmagatzemada en el registre IR es descodifica per a identificar les diferents parts de la instrucció i determinar les operacions necessàries que cal fer en les fases següents.

Lectura dels operands font

El nombre de passos que cal fer en aquesta fase depèn del nombre d'operands font i dels modes d'adreçament utilitzats en cada operand. Si hi ha més d'un operand cal repetir el procés per a cadascun d'aquests operands.

El mode d'adreçament indica el lloc on hi ha la dada:

- Si la dada és a la instrucció mateixa, no cal fer res perquè ja la tenim en la mateixa instrucció.
- Si la dada és en un registre, no cal fer res perquè ja la tenim disponible en un registre dins del processador.
- Si la dada és a la memòria, cal portar la dada al registre MBR.

Exemple

Vegem-ne ara algun exemple:

- Immediat: la dada és en la mateixa instrucció i, per tant, no cal fer res: IR(operand).
- Directe a registre: la dada és en un registre i, per tant, no cal fer res.
- Relatiu a registre índex:
  - **MAR ← IR(Adreça operand) + Contingut IR(registre índex)**
  - **MBR ← Memòria**: llegim la dada.

IR(camp)

Camp, a IR(camp), és un dels camps de la instrucció que acabem de llegir i que tenim guardat en el registre IR.

La major part de jocs d'instruccions no deixen especificar dos operands font amb accés a la memòria, ja que la dada obtinguda es deixa en el MBR i, per tant, si s'especifiquessin dos operands de memòria, s'hauria de guardar el primer temporalment en un altre registre, cosa que introduiria un retard considerable en l'execució de la instrucció.

En arquitectures amb un sol bus intern (o de més d'un bus però amb determinades configuracions d'accés als busos) també cal afegir microoperacions per a guardar temporalment informació en registres quan es treballa amb més d'una dada alhora, com per exemple quan es fa una suma.

Execució de la instrucció i emmagatzematge de l'operand de destinació

El nombre de passos que cal fer en aquesta fase depèn del codi d'operació de la instrucció i del mode d'adreçament utilitzat per a especificar l'operand de destinació. Cal, per tant, una descodificació per a obtenir aquesta informació.

Per a executar algunes instruccions cal l'ALU. Per a operar amb l'ALU cal tenir disponibles al mateix temps tots els operands que utilitza, però l'ALU no disposa d'elements per a emmagatzemar-los; per tant, s'han d'emmagatzemar en registres del processador. Si no hi ha un bus diferent des d'on es pugui captar cadascun dels operands font i on es pugui deixar l'operand de destinació cal registres temporals (transparents al programador) connectats directament a l'ALU (entrada i sortida de l'ALU) per a tenir-los disponibles al mateix temps, cosa que comporta l'ús de microoperacions addicionals per a portar els operands a aquests registres temporals.

Si els operands font són en registres disponibles per l'ALU, la microoperació per a fer la fase d'execució és de la manera següent:

Registre de destinació ← Registre d'origen <operació> Registre d'origen o Valor

Si l'operand de destinació no és un registre, cal resoldre abans el mode d'adreçament per a emmagatzemar la dada, de manera molt semblant a la lectura de l'operand font.

Vegem com es resoldrien alguns d'aquests modes d'adreçament.

Directe a memòria:

- MBR ← <Resultat execució>
- MAR ← IR(Adreça operand)
- Memòria ← MBR

Relatiu a registre base:

- MBR ← <Resultat execució>
- MAR ← Contingut IR(RB) + IR(Desplaçament operand)
- Memòria ← MBR

Cal tenir present que en moltes arquitectures l'operand de destinació és el mateix que un dels operands font i, per tant, els càlculs conseqüència del mode d'adreçament ja estan resolts, és a dir, ja se sap on es guarda la dada i no cal repetir-ho.

Comprovació d'interrupcions

En aquesta fase, si no s'ha produït cap petició d'interrupció, no cal executar cap microoperació i es continua amb l'execució de la instrucció següent; si se n'ha produït una, s'ha de fer un canvi de context. Per a fer un canvi de context cal guardar l'estat del processador (generalment a la pila del sistema) i posar en el PC l'adreça de la rutina que dóna servei a aquesta interrupció. Aquest procés pot variar molt d'una màquina a una altra. Aquí només presentem la seqüència de microoperacions per a actualitzar el PC.

- **MBR ← PC**: es posa el contingut del PC en el registre MBR.
- **MAR ← Adreça de salvaguarda**: s'indica on es guarda el PC.
- **Memòria ← MBR**: es guarda el PC a la memòria.
- **PC ← Adreça de la rutina**: es posiciona el PC a l'inici de la rutina de servei de la interrupció.

Vegeu també

Aquest punt el veurem amb més detall en el mòdul "Sistema d'entrada/sortida" d'aquesta mateixa assignatura.

5.2. Senyals de control i temporització

Hem vist que cada microoperació fa una tasca determinada dins del cicle d'execució d'una instrucció. Tot i la simplicitat d'aquestes microoperacions, dur-les a terme comporta l'activació d'un conjunt de senyals de control.

De manera general entenem un **senyal de control** com una línia física que surt de la unitat de control i va cap a un dispositiu o més d'un dispositiu del computador per on circula un senyal elèctric que representa un valor lògic 0 o 1 i segons quins siguin els dispositius on està connectat és actiu per flanc o per nivell.

Vegeu també

Els conceptes relacionants amb els senyals elèctrics els hem treballat en el mòdul de circuits lògics de l'assignatura de *Fonaments de computadors* i no els analitzarem en més detall.

La major part dels computadors tenen un funcionament síncron, és a dir, la seqüència d'operacions és governada per un senyal de rellotge. El període d'aquest senyal de rellotge (el temps que triga a fer una oscil·lació sencera), anomenat també *cicle de rellotge*, determina el temps mínim necessari per a fer una operació elemental en el computador. Aquesta operació elemental considerarem que és una microoperació.

Operació elemental i microoperació

La identificació "operació elemental – microoperació", en els computadors actuals, no sempre és tan fàcilment generalitzable, en gran manera per conceptes més avançats d'arquitectura del computador que no hem tractat i que, per tant, no considerarem.

Per a dur a terme el control de l'execució d'una instrucció, cal fer les quatre fases següents:

- F_1: lectura de la instrucció.
- F_2: lectura dels operands font.
- F_3: execució de la instrucció i emmagatzematge de l'operand de destinació.
- F_4: comprovació d'interrupcions.

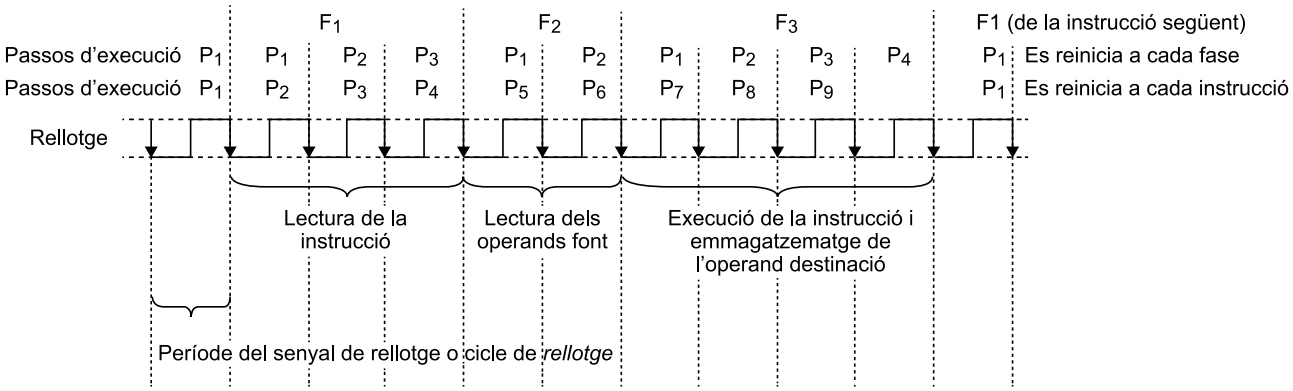
En cadascuna d'aquestes fases es fan un conjunt de microoperacions i, per tant, requereixen un cicle o més d'un cicle de rellotge per a executar-les (o cap si no es fa cap microoperació en aquella fase). Per a controlar l'execució de les microoperacions en cada fase, es divideix cada fase en una seqüència de passos d'execució.

Un pas d'execució (P_i) és el conjunt de microoperacions que es poden executar simultàniament en un cicle de rellotge. El nombre de passos d'execució que es fan en cada fase pot ser diferent.

El fet de dividir el cicle d'execució en fases i passos permet sistematitzar el funcionament de la unitat de control i simplificar-ne el disseny. Fent-ho d'aquesta manera, per a controlar les fases i els passos d'execució només és necessari tenir un comptador per a les fases que es reinicia en començar cada instrucció i un comptador de passos que, en algunes màquines, es reinicia en començar cada instrucció i en d'altres es reinicia en començar cada fase.

CISCA

En el mòdul "Arquitectura CISCA" expliquem en detall la seqüència de microoperacions de l'execució d'una instrucció concreta.



Per a optimitzar el temps d'execució de cada fase i, per tant, de la instrucció, s'han d'intentar executar simultàniament dues microoperacions o més en un mateix pas d'execució. Fins ara, hem vist per una banda que l'execució de les microoperacions implica la utilització d'uns determinats recursos del computador, i per l'altra que aquestes microoperacions s'executen durant cert temps, generalment un cicle de rellotge. Per a assegurar-nos que dues microoperacions o més es poden executar al mateix temps hem de tenir en compte quins recursos s'utilitzen i durant quants cicles de rellotge s'utilitzen.

L'agrupació de microoperacions ha de seguir bàsicament dues regles:

1) Ha de seguir la seqüència correcta d'esdeveniments. No es pot executar una microoperació que genera o obté una dada al mateix temps que una altra microoperació que l'ha d'utilitzar.

> Per exemple, no es pot fer una suma al mateix temps que es llegeix un dels operands de memòria, ja que no es pot assegurar que aquest operand sigui disponible fins al final del cicle i, per tant, no es pot garantir que la suma es faci amb la dada esperada.

2) S'han d'evitar els conflictes. No es pot utilitzar el mateix recurs per a dues microoperacions diferents.

> Per exemple, no es pot fer servir el mateix bus per a transferir dues dades o adreces diferents al mateix temps.

5.3. Unitat de control cablejada

Fins ara, hem vist la unitat de control des d'un punt de vista funcional: entrades, sortides, tasques que fa i com interacciona amb la resta d'elements del computador. A continuació ens centrarem en les tècniques de disseny i implementació de la unitat de control.

Aquestes tècniques es poden classificar en dues categories:

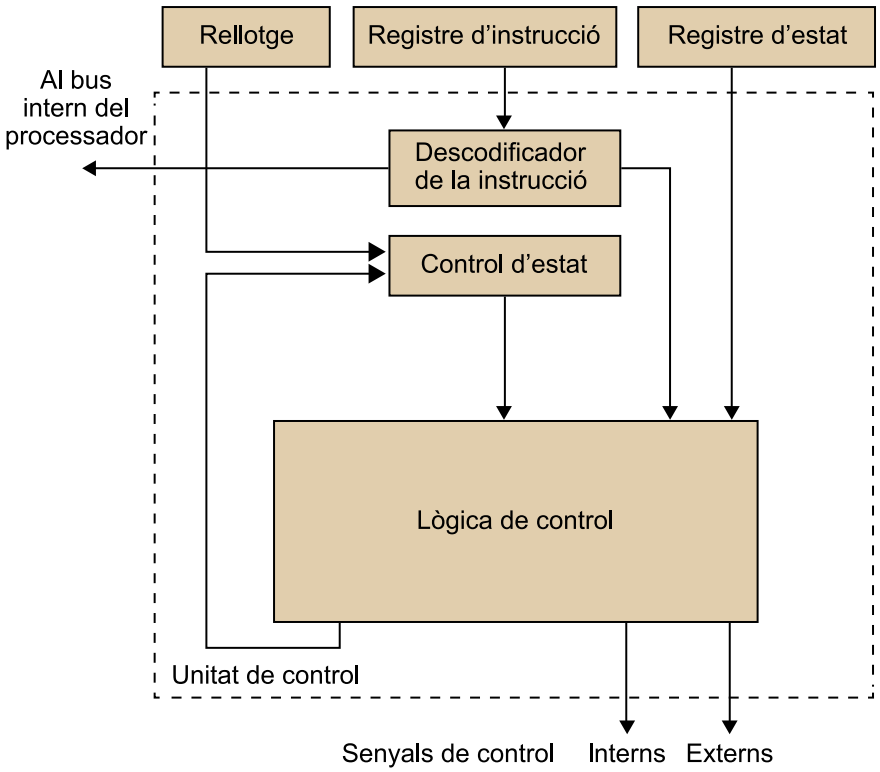
- Control cablejat.
- Control microprogramat.

La unitat de control cablejada és, bàsicament, un circuit combinacional que rep un conjunt de senyals d'entrada i els transforma en un conjunt de senyals de sortida que són els senyals de control. Aquesta tècnica és la que s'utilitza típicament en màquines RISC.

Per a analitzar en més detall la unitat de control cablejada estudiarem els elements bàsics que en formen part.

5.3.1. Organització de la unitat de control cablejada

En el diagrama següent es mostren els mòduls que formen la unitat de control i la interconnexió entre uns i altres.



A continuació, es descriuen en detall cadascun dels mòduls que componen aquesta unitat de control:

1) Descodificador de la instrucció. La funció bàsica d'aquest mòdul és descodificar la instrucció. Obté la informació del registre IR i proporciona una sortida independent per a cada instrucció de màquina, i també s'encarrega de subministrar els valors immediats (especificats a la instrucció mateixa) al bus intern del processador per a treballar amb aquests valors.

2) Control d'estat. La funció d'aquest mòdul és comptar els passos dins de cada fase del cicle d'execució de la instrucció. Rep el senyal de rellotge (seqüència repetitiva d'impulsos que s'utilitza per a delimitar la durada de les microope-

racions) i els senyals de la lògica de control necessaris per a controlar les fases del cicle d'execució, i genera un senyal diferent per a cada pas dins de cada fase del cicle d'execució i els senyals per a identificar la fase dins el cicle d'execució.

3) Lògica de control. Aquest mòdul és un circuit combinacional que fa les operacions lògiques necessàries per a obtenir els senyals de control corresponents a una microoperació. Rep els senyals dels altres dos mòduls de la unitat de control cablejada i del registre d'estat i subministra informació del resultat d'aquesta microoperació al mòdul que fa el control d'estat.

Aquest circuit es pot implementar mitjançant dos nivells de portes lògiques, però a causa de diferents limitacions dels circuits lògics es fa amb més nivells. Tot i això continua essent molt ràpid. Cal tenir present que les tècniques modernes de disseny de circuits VLSI fan molt més fàcil el disseny d'aquests circuits.

Exemple

Utilitzant de referència els senyals de control de l'arquitectura CISCA, i la seqüència de microoperacions que tenen per a executar les instruccions:

$$\text{MBRoutA} = F_1 \cdot P_3 + I_k \cdot F_3 \cdot P_1 + \ldots$$

On I_k: instruccions de transferència del joc d'instruccions on es transfereixi una dada que s'ha llegit de memòria a un registre.

D'aquesta expressió n'interpretem que el senyal de control MBRoutA és actiu quan som en el pas 3 de la fase 1 de qualsevol instrucció (no n'especifiquen cap, ja que la fase de lectura de la instrucció és comuna a totes les instruccions), o en el pas 1 de la fase 3 de la instrucció I_k, o en el pas ..., i es fa de la mateixa manera per a la resta de casos en què és necessari activar el senyal.

Com es veu, és una feina molt complexa i laboriosa i el fet de redissenyar el computador pot implicar canviar moltes d'aquestes expressions.

Nota

No analitzarem amb més detall aquesta implementació cablejada, ja que hauríem d'analitzar amb més detall la problemàtica del disseny de circuits lògics i aquest no és l'objectiu d'aquests materials. Trobareu informació sobre aquesta tècnica de disseny en bibliografia específica sobre aquest tema.

5.4. Unitat de control microprogramada

La microprogramació és una metodologia per a dissenyar la unitat de control, proposada per M. V. Wilkes el 1951, que pretén ser un mètode de disseny organitzat i sistemàtic que eviti les complexitats de les implementacions cablejades.

Aquesta metodologia es basa en la utilització d'una memòria de control que emmagatzema les microoperacions necessàries per a executar cadascuna de les instruccions i en el concepte de **microinstrucció** que veurem més endavant.

En els anys cinquanta la proposta va semblar poc viable ja que feia necessari disposar d'una memòria de control que fos, alhora, ràpida i econòmica. Amb la millora en velocitat dels circuits de memòria i l'abaratiment d'aquests circuits, va esdevenir una metodologia molt utilitzada, especialment en el disseny d'unitats de control complexes com és el cas de les arquitectures CISC.

Com ja sabem, l'execució d'una instrucció comporta realitzar una sèrie de microoperacions, i cada microoperació implica l'activació d'un conjunt de senyals de control, cadascun dels quals pot ser representat per un bit.

La representació del conjunt de tots els senyals de control (combinació de zeros i uns) dóna lloc al que anomenem **paraula de control**.

Paraula de control

Cal tenir present que una paraula de control determina tots els senyals de control que genera la unitat de control, ja que es fan servir tant per a indicar quins recursos s'utilitzen com per a assegurar que la resta de recursos no interfereixen en l'execució de les microoperacions en curs.

Si dues microoperacions o més es poden executar en un mateix període, es poden activar els senyals de control necessaris en una mateixa paraula de control.

5.4.1. Microinstruccions

Anomenem **microinstrucció** la notació utilitzada per a descriure el conjunt de microoperacions que es fa simultàniament en un mateix període i es representa amb una paraula de control.

L'execució d'una microinstrucció implica activar els senyals de control corresponents a les microoperacions que fa i determinar l'adreça de la microinstrucció següent.

Cada microinstrucció s'emmagatzema en una memòria de control i, per tant, cadascuna té assignada una adreça de memòria diferent.

Per a especificar la seqüència d'execució de les microinstruccions, cal més informació que la mateixa paraula de control. Aquesta informació addicional dóna lloc a dos tipus bàsics de microinstruccions:

1) Microinstrucció horitzontal. S'afegeix a la paraula de control un camp per a expressar una condició i un camp per a especificar l'adreça de la microinstrucció següent que s'ha d'executar quan es compleixi la condició.

Els bits de la paraula de control representen directament els senyals de control.

2) Microinstruccions verticals. Per a reduir la mida de les microinstruccions horitzontals, es codifiquen els bits de la paraula de control, de manera que se'n redueix el nombre de bits necessaris.

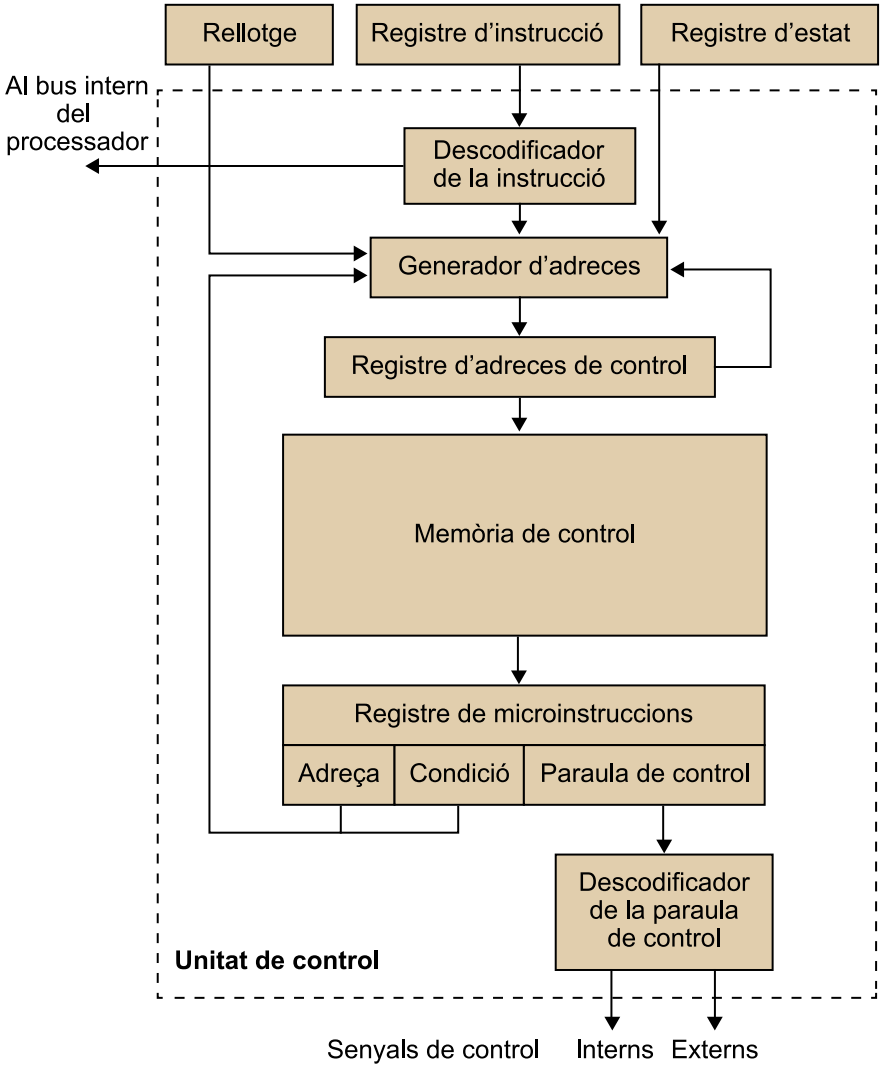
Si tenim una paraula de control, on tenim grups de 2^N bits que no s'activen simultàniament, es poden codificar utilitzant N bits. Per exemple, un grup de $2^4 = 16$ bits de la paraula de control es pot codificar utilitzant només 4 bits.

L'inconvenient d'aquest tipus de microinstruccions és que els bits de la paraula de control no representen directament els senyals de control, i cal un descodificador per a traduir els codis que apareixen en la microinstrucció a senyals de control individuals.

El conjunt de microinstruccions necessàries per a executar una instrucció dóna lloc a un **microprograma** (microcodi o *firmware*). Com que cada instrucció fa tasques diferents, cal un microprograma diferent per a cada instrucció del joc d'instruccions de l'arquitectura.

5.4.2. Organització d'una unitat de control microprogramada

En el diagrama següent es mostren els mòduls que formen la unitat de control i la interconnexió entre uns i altres.



A continuació, es descriuen en detall cadascun dels mòduls que componen aquesta unitat de control:

1) Descodificador de la instrucció. Aquest mòdul identifica els diferents camps de la instrucció i envia la informació necessària al generador d'adreces per al control dels microprogrames (generalment, l'adreça d'inici del microprograma de la instrucció en curs) i subministra els valors immediats (especificats en la mateixa instrucció) al bus intern del processador perquè pugui treballar amb aquests valors.

2) Generador d'adreces. Aquest mòdul genera l'adreça de la microinstrucció següent i la carrega al registre d'adreces de control.

3) Registre d'adreces de control. En aquest registre es carrega l'adreça de memòria de la microinstrucció següent que s'executarà.

4) Memòria de control. Emmagatzema el conjunt de microinstruccions necessàries per a executar cada instrucció o microprograma.

5) Registre de microinstruccions. Aquest registre emmagatzema la microinstrucció que s'acaba de llegir de la memòria de control.

6) Descodificador de la paraula de control. Aquest mòdul, quan s'utilitzen microinstruccions verticals, tradueix la paraula de control inclosa en la microinstrucció als senyals de control individuals que representa.

5.4.3. Funcionament de la unitat de control microprogramada

Les dues tasques que fa la unitat de control microprogramada són la seqüenciació de les microinstruccions i la generació dels senyals de control.

Seqüenciació de les microinstruccions

Una de les tasques més complexes que fa la unitat de control microprogramada és determinar l'adreça de la microinstrucció següent que s'ha d'executar, tasca que duu a terme el generador d'adreces.

L'adreça de la microinstrucció següent la determinen els factors següents:

- **El registre d'instrucció**, que només s'utilitza per a determinar el microprograma propi de cada instrucció.

- **El registre d'estat**, que només s'utilitza quan cal trencar la seqüència normal d'execució d'un programa, com passa quan s'han d'avaluar les condicions en una instrucció de salt condicional.

- **El camp de condició de la microinstrucció**. És el factor que s'utilitza més freqüentment. Aquest és el factor que cal avaluar per a saber si s'ha de continuar en seqüència o fer una bifurcació:
  - Si la condició indicada és falsa, la microinstrucció següent que s'executarà és la que segueix en seqüència i l'adreça d'aquesta microinstrucció pot estar especificada en la mateixa microinstrucció o es pot obtenir incrementant el valor actual del registre d'adreces de control.

  - Si la condició indicada és certa, la microinstrucció següent que s'executarà és la que indica l'adreça de bifurcació i aquesta adreça s'obté de la mateixa microinstrucció.

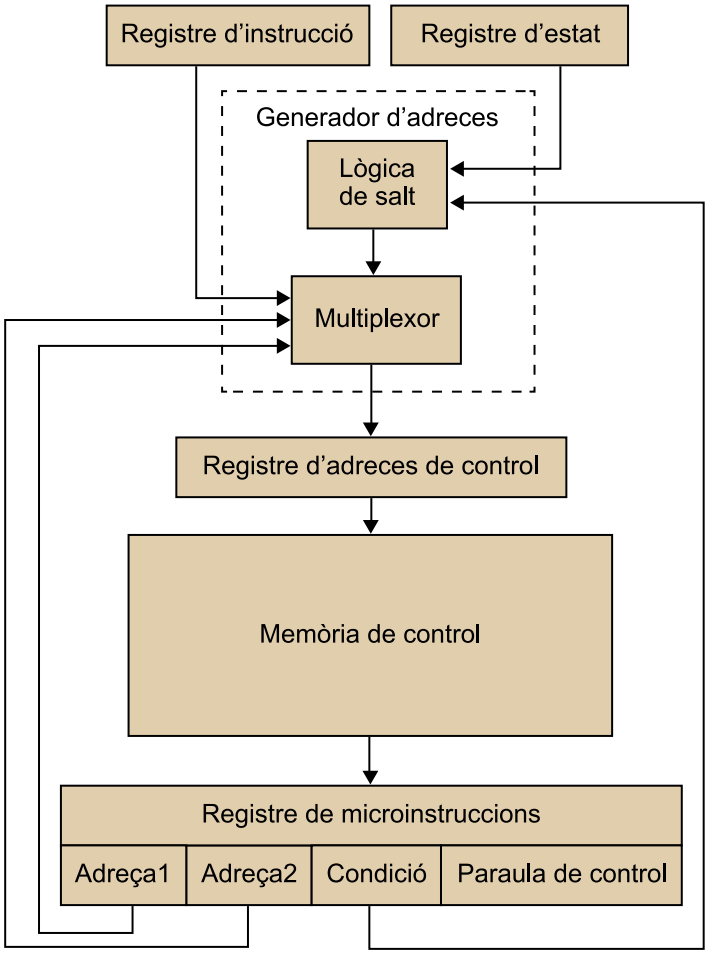
Formats del camp d'adreça i condició de la microinstrucció

El camp d'adreça i condició de les microinstruccions pot tenir els formats següents:

1) Dos camps explícits. En la microinstrucció s'especifiquen dues adreces explícites: l'adreça de la microinstrucció que segueix en seqüència i una adreça de bifurcació. Segons si es compleix o no la condició, es carrega una adreça o l'altra en el registre d'adreces de control.

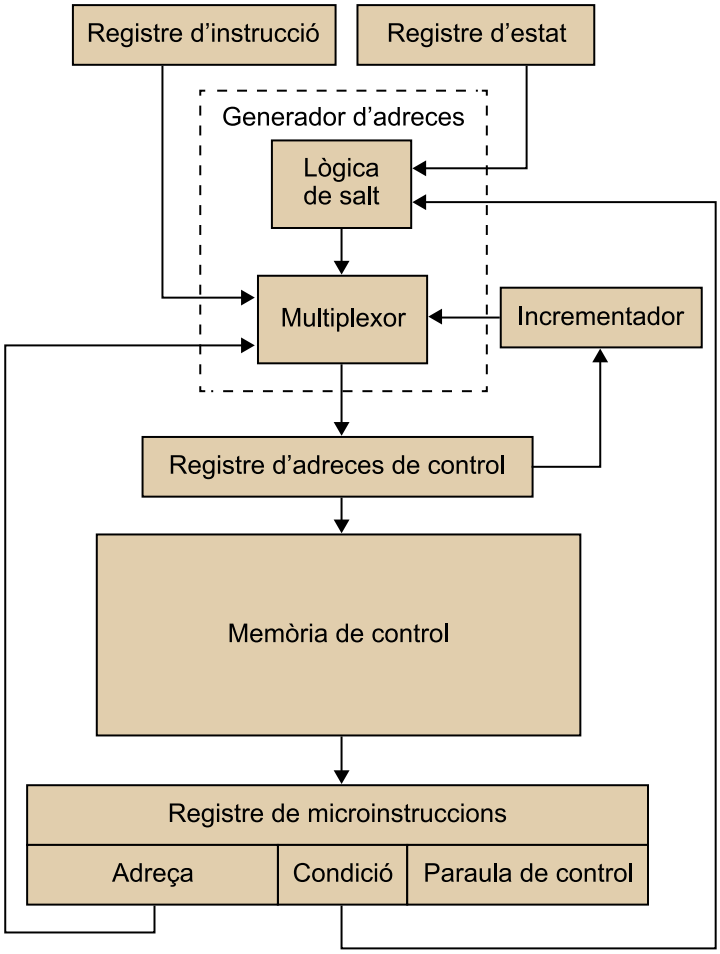
Amb aquesta tècnica no cal una lògica per a incrementar l'adreça actual i passar a la microinstrucció següent que segueix en seqüència però requereix més bits que altres formats.

Exemple de circuit generador d'adreces amb dos camps explícits



2) Un camp explícit. En la microinstrucció s'especifica explícitament una adreça: quan es compleix la condició indicada es carrega el registre d'adreces de control amb l'adreça explícita. Quan no es compleix la condició, es carrega el registre d'adreces de control amb el valor incrementat del registre d'adreces de control.

Exemple de circuit generador d'adreces amb un camp explícit



3) Microinstruccions de format variable. En aquest format només hi ha el camp de condició i el camp de paraula de control. En les microinstruccions de salt, s'utilitza una part del camp de la paraula de control com a adreça de bifurcació, cosa que obliga a utilitzar un bit de la microinstrucció per a indicar si cal interpretar aquesta part del camp de la paraula de control com l'adreça de la microinstrucció següent o com a part de la paraula de control.

L'avantatge principal d'aquest mètode és que ens permet reduir la mida de les microinstruccions i no complica excessivament la lògica de control de la unitat de control.

Generació dels senyals de control

Generalment les unitats de control microprogramades no utilitzen un format totalment horitzontal de la paraula de control, sinó un cert nivell de codificació per a estalviar espai en la memòria de control.

El més habitual és agrupar els bits de la paraula de control en camps; cada camp conté un codi que, un cop descodificat, representa un conjunt de senyals de control.

Un camp amb k bits pot contenir 2^k codis diferents. Cada codi representa un estat diferent dels senyals de control que representa i n'indica, per a cadascun, l'activació o la desactivació.

Considerem que els senyals de control són actius si el bit que els representa val 1 i inactius si aquest bit val 0.

A l'hora de triar una codificació cal tenir present quines són les diferents combinacions de senyals de control que han de ser actius o no en un mateix instant.

La codificació pot ser *directa* (un sol nivell de codificació) o *indirecta* (dos nivells de codificació). En aquest segon cas el valor descodificat que pren un camp serveix per a saber quins senyals de control representa un segon camp ja que poden ser diferents segons el valor que prengui el primer camp.

L'agrupació dels bits de la paraula de control en camps pot respondre a diferents criteris; el més habitual és agrupar-los segons els recursos que representen: es tracten tots els recursos de la màquina de manera independent i s'assigna un camp a cadascun: un camp per a l'ALU, un camp per a l'entrada/sortida, un camp per a la memòria, etc.

5.5. Comparació: unitat de control microprogramada i cablejada

A continuació presentem els avantatges i inconvenients de la unitat de control microprogramada respecte de la unitat de control cablejada.

Els avantatges són:

- Simplifica el disseny.
- És més econòmica.
- És més flexible:
  - Adaptable a la incorporació de noves instruccions.
  - Adaptable a canvis d'organització, tecnològics...
- Permet disposar d'un joc d'instruccions amb gran nombre d'instruccions. Només cal disposar d'una memòria de control de gran capacitat.
- Permet disposar de diversos jocs d'instruccions en una mateixa màquina (emulació de màquines).
- Permet ser compatible amb màquines anteriors de la mateixa família.

- Permet construir màquines amb diferents organitzacions però amb un mateix joc d'instruccions.

Intel i AMD

Intel i AMD utilitzen dissenys diferents per als seus processadors amb organitzacions internes diferents però gràcies a la utilització d'unitats de control microprogramades tots dos treballen amb el mateix joc d'instruccions.

Els inconvenients són:

- És més lenta: implica accedir a una memòria i interpretar les microinstruccions necessàries per a executar cada instrucció.
- És necessari un entorn de desenvolupament per als microprogrames.
- És necessari un compilador de microprogrames.

6. Computadors CISC i RISC

En el disseny del processador cal tenir en compte diferents principis i regles, amb vista a obtenir un processador amb les millors prestacions possibles.

Les dues alternatives principals de disseny de l'arquitectura del processador són les següents:

- Computadors CISC[4], que tenen les característiques següents:
  - El format d'instrucció és de longitud variable.
  - Disposa d'un gran joc d'instruccions, habitualment més de cent, per a donar resposta a la majoria de necessitats dels programadors.
  - Disposa d'un nombre molt elevat de modes d'adreçament.
  - És una família anterior a la dels processadors RISC.
  - La unitat de control és microprogramada; és a dir, l'execució d'instruccions es fa descomponent la instrucció en una seqüència de microinstruccions molt simples.
  - Processa instruccions llargues i de mida variable, cosa que dificulta el processament simultani d'instruccions.

(4) *CISC* és l'abreviatura de *complex instruction set computer*; en català, computador amb un joc d'instruccions complex.

- Computadors RISC[5], que tenen les característiques següents:
  - El format d'instrucció és de mida fixa i curta, cosa que permet un processament més fàcil i ràpid.
  - El joc d'instruccions es redueix a instruccions bàsiques i simples, amb les quals s'han d'implementar totes les operacions complexes. Una instrucció d'un processador CISC s'ha d'escriure com un conjunt d'instruccions RISC.
  - Disposa d'un nombre molt reduït de modes d'adreçament.
  - L'arquitectura és de tipus *load-store* (carrega i emmagatzema) o registre-registre. Les úniques instruccions que tenen accés a memòria son LOAD i STORE, i la resta d'instruccions utilitzen registres com a operands.
  - Disposa d'un ampli banc de registres de propòsit general.

(5) *RISC* és l'abreviatura de *reduced instruction set computer*; en català, computador amb un joc d'instruccions reduït.

- Gairebé totes les instruccions es poden executar en pocs cicles de rellotge.

- Aquest tipus de joc d'instrucció facilita la segmentació del cicle d'execució, cosa que permet l'execució simultània d'instruccions.

- La unitat de control és cablejada i no microprogramada.

La segmentació permet que l'execució d'una instrucció comenci abans d'acabar la de l'anterior (s'encavalquen les fases del cicle d'execució de diverses instruccions), cosa que redueix el temps d'execució de les instruccions.

Els processadors actuals no són completament CISC o RISC. Els nous dissenys d'una família de processadors amb característiques típicament CISC incorporen característiques RISC, de la mateixa manera que les famílies amb característiques típicament RISC incorporen característiques CISC.

Exemple

PowerPC, que pot considerar-se un processador RISC, incorpora característiques CISC, com ara un joc d'instruccions molt ampli (més de dues-centes instruccions).

Els darrers processadors d'Intel (fabricant de processadors típicament CISC) també incorporen característiques RISC.

Resum

La funció principal del processador és processar les dades i transferir-les als altres elements del computador. Està format pels elements bàsics següents:

- Conjunt de registres.
- Unitat aritmètica i lògica.
- Unitat de control.

El cicle d'execució de la instrucció es divideix en quatre fases. Vegeu en la taula de manera esquemàtica les operacions que es fan en cada fase:

Inici del cicle d'execució	
Fase 1: lectura de la instrucció	Llegir la instrucció.
	Descodificar la instrucció.
	Actualitzar el PC.
Fase 2: lectura dels operands font	Calcular l'adreça i llegir el primer operand font.
	Calcular l'adreça i llegir el segon operand font.
Fase 3: execució de la instrucció i emmagatzematge de l'operand de destinació	Executar la instrucció.
Fase 4: comprovació d'interrupcions	Comprovar si algun dispositiu ha sol·licitat una interrupció.

Els registres s'han classificat en els tipus següents:

- Registres de propòsit general.
- Registres d'instrucció.
- Registres d'accés a memòria.
- Registres d'estat i de control.

La unitat aritmètica i lògica fa operacions aritmètiques i lògiques amb nombres enters i nombres reals en punt fix i en punt flotant.

Hem estudiat en detall la unitat de control, i n'hem vist els aspectes següents:

- Microoperacions.
- Senyals de control i temporització.
- Unitat de control cablejada.
- Unitat de control microprogramada.
- Avantatges i inconvenients de la unitat de control microprogramada respecte a la implementació cablejada.

Finalment, hem parlat de les característiques principals dels computadors CISC i RISC.