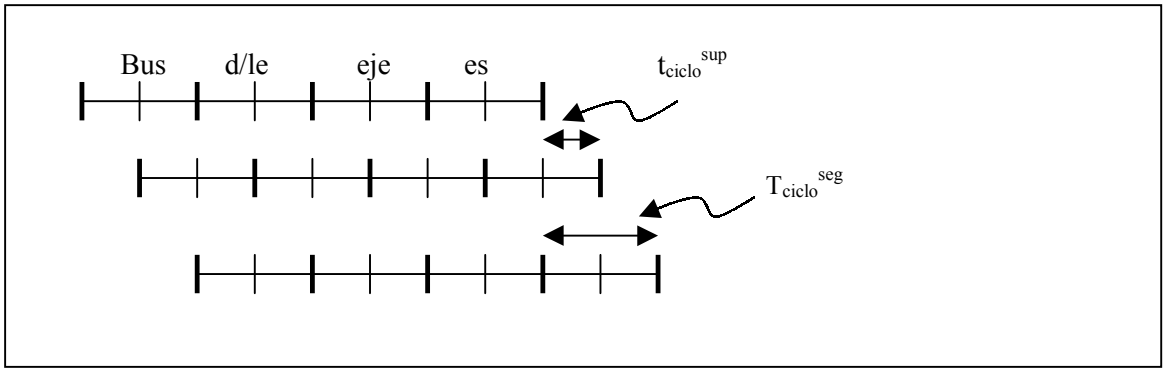
La supersegmentació

1. Introducció

El processador supersegmentat neix amb l'idea de dotar de major velocitat els processadors segmentats. Per fer això es diveix les una etapa en diferents subetapes més petites que es poden execurtar en paral.lel.

Per exemple:



Com es pot observar el temps de cicle en els segmentats correspon al temps de cada etapa, mentre que el processadors supersegmentats el temps de cicle correspondria al temps de cada subetapa. Si ho representem amb una fòrmula seria el següent:

$$t_{ciclo}^{\sup} = \frac{1}{m} * t_{ciclo}^{seg}$$

En els superescalars per tant el temps de busqueda s'adaptarà al temps de cicle del supersegmentat i la latència de les instruccions seran de "m" cicles.

Si les instruccions s'executessin de forma eficient es podria arribar a un paral.lelisme entre instruccions de "m" instruccions.

A nivell de processador els aventatges que té aquest sistema són els següents:

- No es requereix replicar les unitats funcionals donat a que es fa servir una sola unitat funcional que permeti l'execució per etapes.
- La unitat de control pot ésser simple ja que només s'inicia un 1 sola instrucció per cicle.
- Adaptar-nos al fet que la tecnologia actual ens permet una gran capacitat d'integració (quadràtica).

- És una alternativa viable en tecnologies d'alta velocitat a on el retard entre portes són baixos.

Pel que fa a la CPI resultant del sistema tindríem:

$$T = NxCPI^{\sup} xT_{ciclo}^{\sup}$$

Es equivalent a:

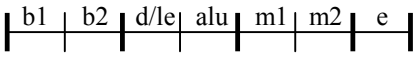
$$T = NxCPI^{\sup} x 1/m xT_{ciclo}^{seg}$$

2 Risc estructurals

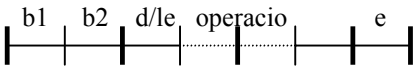
El fet de treballar amb etapes més segmentades ens implicarà que es puguin produir més conflictes a nivell d'accés a memòria, de seqüenciament. Per tant serà molt important com passava amb les màquines segmentades el fet d'introduir el màxim possible de curtcircuits i la introducció de registres entre unitats funcionals.

Per exemple en el següent model que proposem següent on hi ha dos pipelines un per aritmètica sencera i un altre per coma flotant trobem:

ARITMÈTICA SENCERA:



ARITMÈTICA EN COMA FLOTANT



A ON LES ETAPES QUE TROBEM SÓN:

Búsqueda	b1,b2
Dec/lectura	d/le
ALU	alu
Accés memor.	m1,m2
Escriptura registres	es

Com es pot observar en el cas òptim en l'aritmètica sencera es pot donar que 7 instruccions es trobin alhora dins de la unitat.

Per intentar reduir en el cas d'accessos a memòria les etapes que ho poden realitzar requeriríem quatre accessos a memòria i podem realitzar segmentació dels mateixos (b1,b2,m1,m2). Pel que fa als registres per a la unitat sencera es requeriran dos accessos per a lectura i un per escriptura, mentre que la unitat de punt flotant necessitarà de 2 de lectura i dos d'escriptura (un per a la memòria i l'altre per a la unitat funcional).

3 Risc de seqüenciament

Pel que fa al risc de seqüenciament (RAW,WAR,WAW) en front a les arquitectures segmentades augmenta degut el major nombre d'instruccions que es troben en execució concurrent i per tant s'ha d'evitar mitjançant delays tot els possibles conflictes que existeixen.

De la mateixa forma el risc de que es produeixin salt augmenta considerablement i per tant la dificultat augmenta alhora d'establir possibles delays per corregir el problema (major nombre d'instruccions).

En el cas dels salts condicionals la latència que implica el conèixer la següent instrucció a executar implica haver avaluat la condició i per tant les etapes són:

- Búsqueda.
- Decodificació.
- Càlcul del pc+x
- Avaluació condició.

Per intentar millorar el rendiment s'introdueix el concepte de ***predicció del salt***. La predicció consisteix en intentar suposar quin resultat donarà el salt i continuar la seqüència segons la predicció feta (saltar o no saltar). En el cas d'equivocar-nos en la predicció s'anul.larà les instruccions de la seqüència no correctes i s'executarà la seqüència correcta.

Com es pot observar els avantatges que dóna aquest mètode són més grans que els incovenients que pot produir.

Com podrem observar hi ha algunes variacions del mateix per intentar optimitzar-lo.

3.1 Predicció fixa

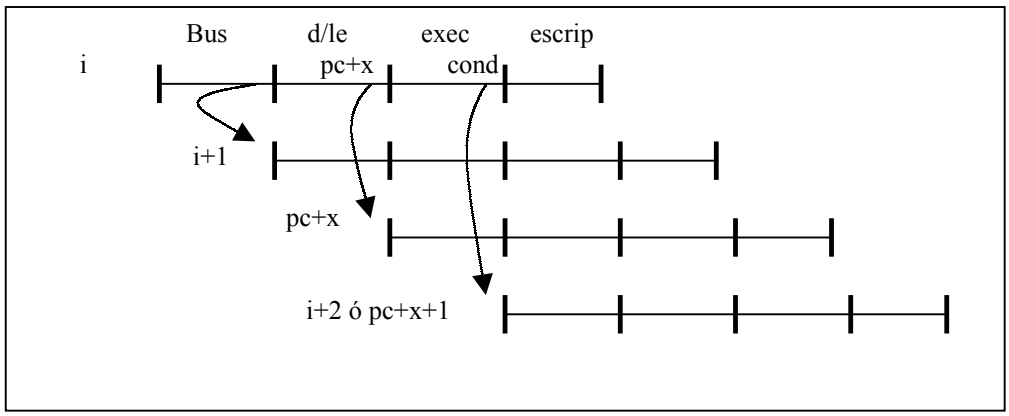
Consisteix en assumir sempre la mateixa predicció quan es produeixi el salt, és a dir el codi sempre suposarà o que la seqüència a seguir és la del salt o que correspon a la de seguir en seqüència sense realitzar el salt.

Aquest mètode es fonamenta en el fet de que s'ha observat que casi sempre un salt pren el mateix sentit cada vegada que s'executa.

Per poder desenvolupar aquest mètode es requereix de poder tenir un sistema per anul·lar totes aquelles instruccions que s'han predit i que no havien d'haver-se executat.

També es requereix que el càlcul de l'adreça de salt (pc+x) sigui anticipada és a dir que es pugui realitzar amb el menor de les latències per evitar el fet d'introduir delays per culpa de la modificació de la seqüència d'execució de les instruccions. De la mateixa manera quan més aviat es faci l'avaluació més petit serà el cos de la recuperació (menys instruccions executant-se en paral.lel) dels salts en el cas d'equivocació en la predicció del mateix.

Per exemple si tenim una estructuració d'etapes com la següent:



Com es pot observar el càlcul de l'adreça de salt es realitza en la segona etapa i per tant això implicarà realitzar un delay entre la instrucció de salt i el salt en el cas de voler fer el salt. També es pot observar que l'avaluació de la condició es realitzarà en la tercera etapa i per tant en el cas d'equivocar-nos perdríem dues instruccions.

La predicció estàtica es pot implementar de tres formes:

- Sempre que es produeixi una instrucció de salt es realitzarà en el mateix sentit.

- El compilador inclourà en el codi generat uns bits on indicarà si el salt es produieix o no. La unitat de control cada vegada que es trobin amb una instrucció de salt optarà a l'hora d'escollir la seqüència per allò que l'indiquin els bits.

- En funció del tipus de desplaçament es realitzarà la seqüencialització, és a dir en els bucles sempre seguirà en seqüència, etc...

Altres tècniques que s'utilitzan seria el fet de la reorganització del software per part del compilador. Això es basa en que els salt cap al davant tenen la tendència marcada en modificar el seqüenciament i després seguir en seqüència. Utilitzant la informació d'una execució anterior el compilador pot reordenar els blocs dins d'un procediment per a que la majoria dels salts cap al davant segueixin en seqüència.

3.2 Predicció dinàmica.

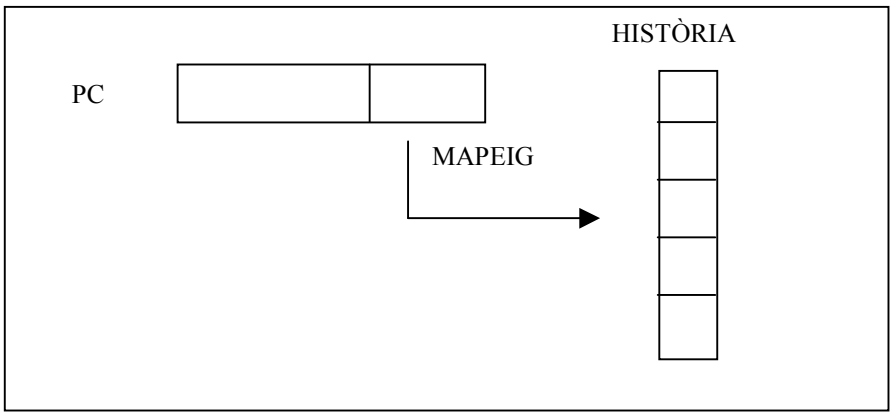
El mètode anterior té la problemàtica de no adaptar-se en temps d'execució al programa en curs. Per millorar aquest aspecte es creen la predicció dinàmica que ens permet guardar una informació històrica del l'execució.

Aquest mètode consisteix en tenir una estructura en forma de llista d'un bit la qual es mapejarà amb els bits menys significatius de l'adreça del PC.

Aquesta llista podrà tenir dos valors (per exemple 0 i 1) que ens indicarà si el salt s'ha de produir o no.

Cada vegada que es descodifica una instrucció de salt automàticament es mapeja els bits menys significatius del "PC" d'aquesta instrucció amb la llista. Segons el valor de la llista es procedirà a fer o no el salt. Aquest mètode implica que cada vegada que s'obtingui l'avaluació del salt es modifiqui el bit de la llista per a tenir-ho com històric.

La estructura seria la següent:

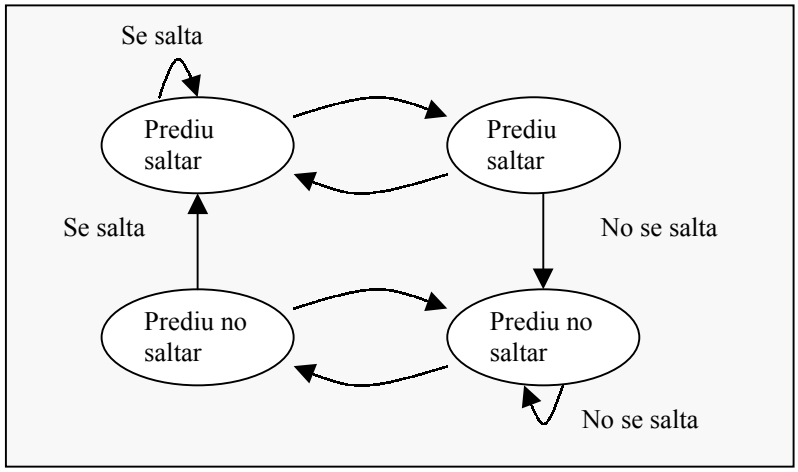


Aquest mètode pot comportar una sèrie de problema com són els següents:

- Es pot arribar a coincidir que dos salts puguin tenir el mateixos bits menys significatius.

- La història que es guarda només es de la instrucció darrera i per tant això no pot arribar a donar una versió del tot correcta del comportament del salt.

Per corregir el darrer problema es va optar per la utilització de dos bits en la predicció del salt i l'esquema d'estats resultant d'aquesta modificació seria el següent:



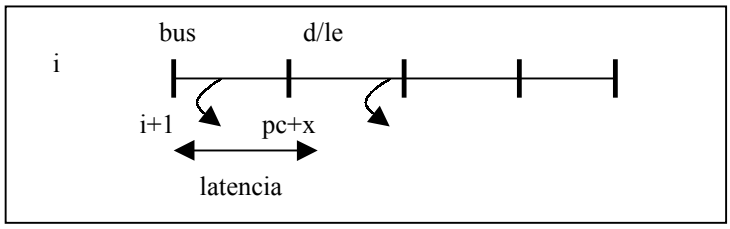
Com es pot observa en aquestes esquema s'utilitzaran 2 bits i per a poder canviar la predicció s'hauran de produir dues prediccions incorrectes.

3.3 Predicció especulativa

El problema dels tractaments de salts vists fins ara és el fet de que no són molt perfectes per sistemes com són els supersegmentats o els escalars a on el nombre d'instruccions que s'executen en paral.lel és força gran.

Per intentar obtenir el màxim rendiment en l'execució dels programes es va optar pel fet de la predicció, és a dir fer una suposició de per on continuaria l'execució del programa. En el cas de que la suposició feta fos errònia s'hauria de refer tot allò que s'ha fet malament.

En els models de predicció especulatives s'han de tenir en compte que la latència del càlcul del PC+X pot afectar de forma important el funcionament del sistema.

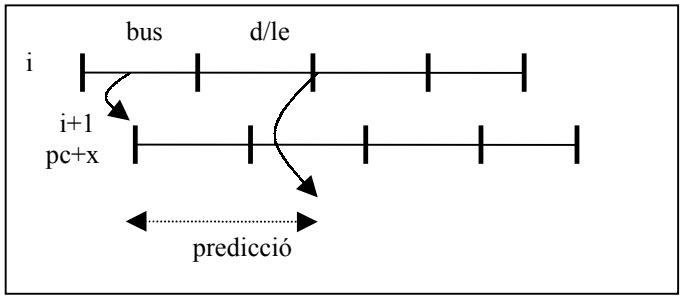


El Pc+x s'hauria de calcular immediatament després de la búsqueda de la instrucció, per saber en cas d'escollir fer el salt l'adreça de la instrucció següent. Per tant la latència es pot definir com:

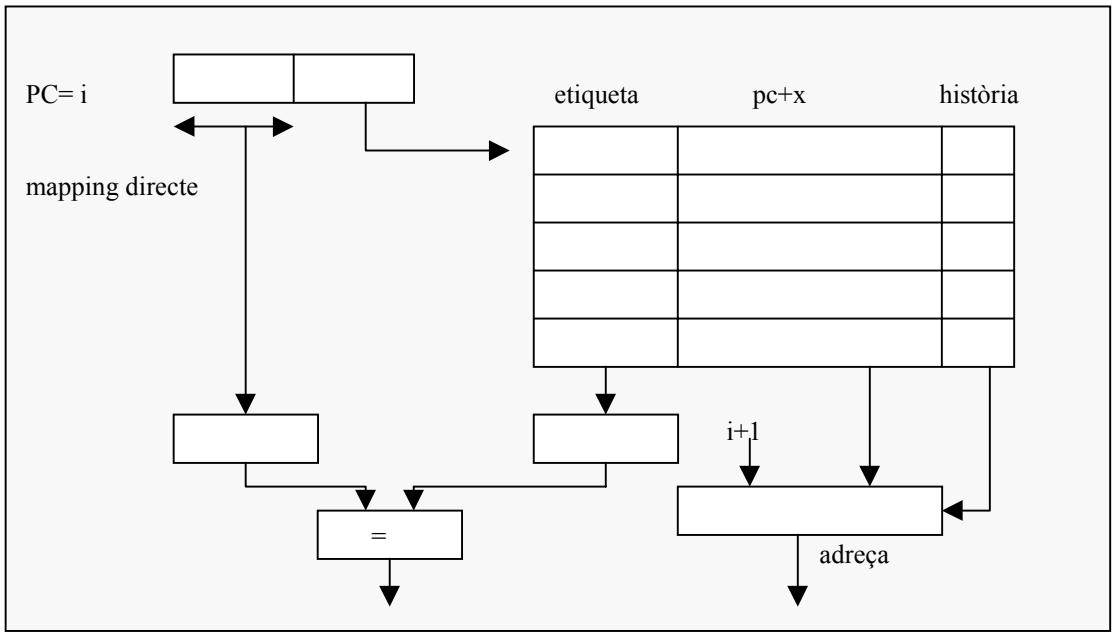
busqueda+dec./càlcul pc+x

Per intentar reduir el màxim possible partirem de la idea de que en els bucles es calcula sempre la mateixa adreça en cada iteració, és a dir, que una vegada calculada l'adreça per primer cop ja ens servirà per a la resta de vegades.

Disposarem d'una cache per guardar l'adreça el primer cop i a partir d'aquí la resta de vegades accedirem a la cache per consultar l'adreça del PC+x d'aquest salt. Com a conseqüència ens avançarem al càlcul de la mateixa i la latència obtinguda serà de 1.



Un exemple d'aquest sistema seria l'anomenat BTB (Branch Target Buffer).



En el funcionament d'aquest mètode s'ha de diferenciar dues fases diferents que afecten al treball amb el BTB:

- La predicció del salt.
- Una vegada l'avaluació del mateix s'ha dut a terme.

En la primera part el BTB ens servirà per a predir l'adreça de la següent instrucció que s'haurà de descodificar. Aquesta fase comprèn els següent passos:

1) Quan descodificà la unitat de control una instrucció automàticament envia el PC a la memòria i al BTB.

2) Accedeix al BTB utilitza els bits menys significatius del PC i mira si hi ha una entrada amb aquest valor. En el cas de que es trobi aquests bits en el BTB per a comprovar que l'adreça és la mateixa, es comparen els bits més significatius del PC amb els de l'etiqueta del BTB. En l'accés al BTB poden donar-se tres casos:

   a. Que es trobi l'adreça i per tant vol dir que el salt ja s'havia produït abans. Llavors el que s'ha de fer per saber si continuar en seqüència o no seria consultar l'històric que es troba en la casella de la taula del BTB (mirar apartat 3.2).

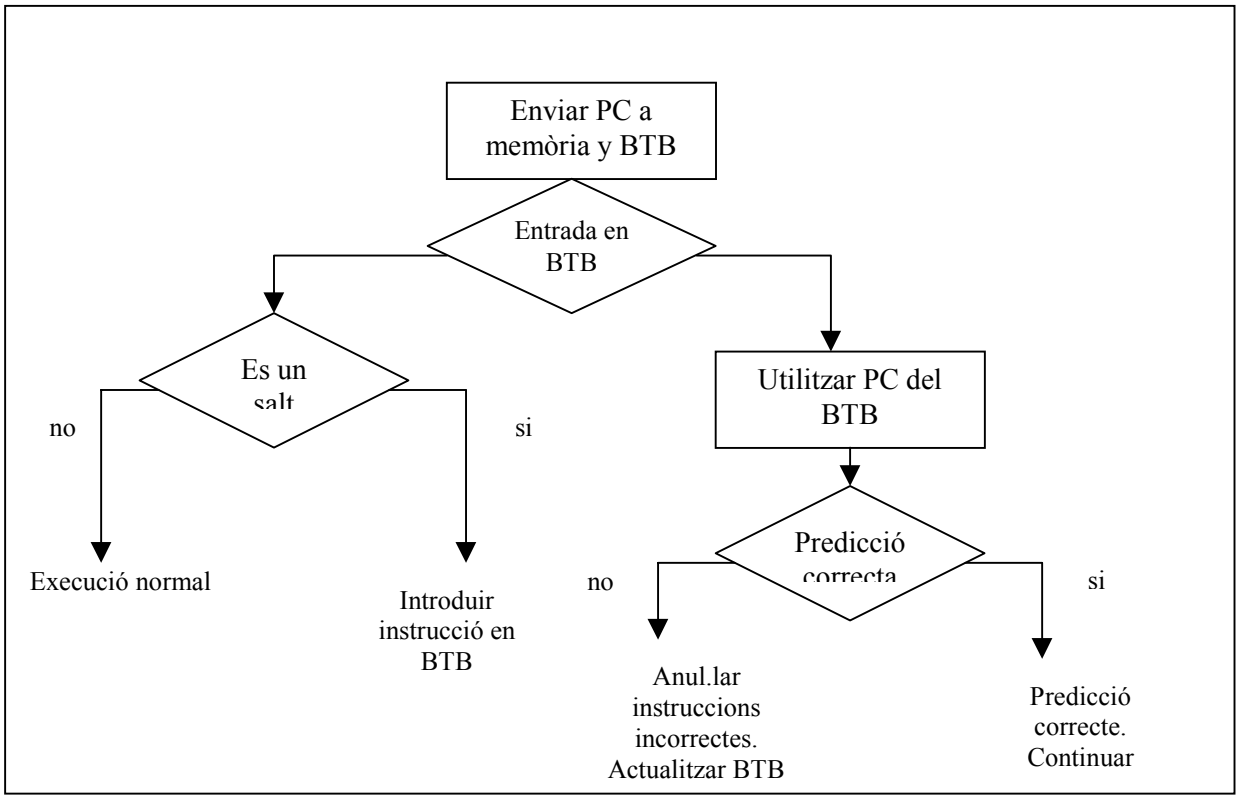
b. Que l'adreça no es trobi en el BTB i la instrucció no sigui de salt amb el que la unitat de control continuaria en seqüència l'execució.

c. Que l'adreça no es trobi en el BTB i la instrucció sigui de salt s'executaria també en seqüència per que no es disposa del PC+X.

En la segona part una vegada s'hagi avaluat la condició s'haurà de tornar a accedir al BTB en dos cassos:

a. Quan la predicció no es trobava en el BTB s'haurà d'accedir en el mateix introduint una nova casella en el mateix amb l'etiqueta de la instrucció, l'adreça del PC que hauria de tenir la instrucció i la informació de l'històric.

b. Quan la predicció ha sigut errònia s'haurà d'accedir a la casella del BTB corresponent aquesta instrucció i modificar la predicció de la mateixa juntament amb la informació de l'històric.



Respecta a la penalització que pot tenir aquest mètode, en general, es fa la suposició que l'accés al BTB per buscar la instrucció no comporta cap pèrdua, mentre que la pèrdua es pot produir en el cas de que la predicció fos incorrecte o que el PC no es trobi en el BTB.

En el primer cas el nombre de cicles perduts serà igual al nombre d'instruccions que s'han d'anul.lar i en el segon cas serà igual a un cicle que és el temps que es perd en actualitzar el BTB.

També cal tenir en compte a la hora de fer el càlculs dels cicles perduts és el fet de l'existència dels delays.

Un exemple seria el següent exercici:

En un supersegmentat amb el pipeline de la figura es vol decidir quina és la millor política de disseny dels salts:

- Utilitzar delay-branch
- Utilitzar un BTB més predicció dinàmica. No utilitzar cap delay-branch. En cas de no estar en el BTB utilitzar la predicció estàtica de sempre saltar. L'actualització del BTB costa un cicle.

Fetch	Fetch	Desc./lect PC+X	ALU Avaluació	Mem	Mem	Escipt.

Contesta:

a) Quants cicles costa l'execució d'un salt? Comenta-ho per tots els casos.

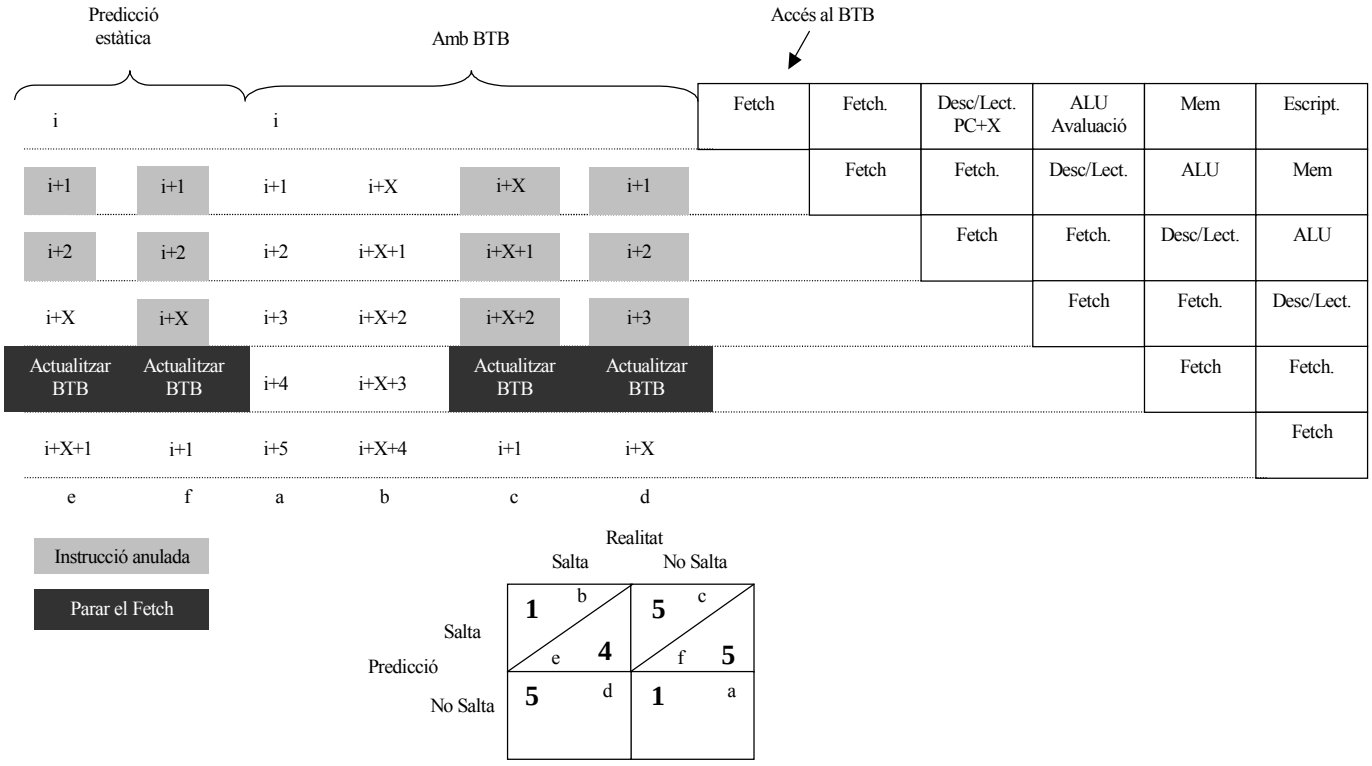
Si en l'execució d'un programa el 30% són salts, d'aquests el 70% salten. En el primer cas, el compilador es capaç d'optimitzar el 40% dels delay-branches. En el segon cas, el BTB té una tassa d'encert del 90%, amb una predicció dinàmica correcte del 85%. Digues quina de les dues opcions dona un nombre de CPI més petit.

SOLUCIÓ

a)

- Delay branch: EN aquest cas fan falta 3 delay branches. Per tant el cost d'un salt va des de 1 cicle quan el compilador és capaç d'optimitzar tots els delays, fins a 4 cicles quan tots els delays estan omplerts amb NOPs.

- BTB: Suposo que el BTB es llegeix durant el primer cicle de Fetch



Com es pot observar hi ha dues possibilitats en el cas de que el salt no es trobi en el BTB i s'hagi de fer una predicció estàtica que són els casos "e" i "f". També cal observar que en aquest exemple el fet de que el càlcul del BTB no és disponible fins el tercer cicle s'haurà de seguir en seqüència l'execució.

b) CPI
   - Delay-branch:

```
0,7*1 + 0,3*(0,6*4 + 0,4*1)
```

   - BTB

```
0,7*1+0,3*(0.9*(0,7*(0,85*1+0,15*5) +; BTB i salta
                 0,3*(0,85*1 + 0,15*5); BTB i no salta
               ) +
            0,1*(0,7*4+0,3*5)          ; No BTB
           )
```

4. Exercicis

PROBLEMA 1:

Suposem un processador supersegmentat amb el següent pipeline:

fetch1	fetch2	descodifi-cació	lectura PC+X	execució avaluació	execució (opcional)	execució (opcional)	escriptura

Contesta:

a) Si suposem un sistema de salts basat en la tècnica del delay, quants cicles costa l'execució d'una instrucció de salt? Justifica la resposta.

En cas d'utilitzar un BTB accedit amb l'adreça del salt, amb 2 cicles d'accés per lectura i 1 cicle per l'escriptura, quants cicles costa l'execució de les instruccions de salt en el cas de tenir la informació en el BTB i fallar la predicció de continuar en seqüència? Justifica la resposta.

SOLUCIO:

a) Es necessiten 4 retards. El cost del salt depèn de la possibilitat d'optimitzar aquests retards. En millor dels casos el salt costa 1 cicle i en el pitjor 5 cicles.

Salt	F	F	D	PC+X	Avalu.						
i+1 (delay)		F	F	D	L	X	X	X	W		
i+2 (delay)			F	F	D	L	X	X	X	W	
i+3 (delay)				F	F	D	L	X	X	X	W
i+4 (delay)					F	F	D	L	X	X	X
i+5 o PC+X						F	F	D	L	X	X

b) El salt és en el BTB i es fa una predicció dinàmica de continuar en seqüència.

Suposo que la instrucció i+1 és un retard (l'altre possibilitat és que es faci una predicció estàtica de continuar en seqüència, i en cas de canviar la predicció dinàmicament, s'anul·la la instrucció)

Salt	F	F	D	PC+X	Avalu.						
i+1 (delay)		F	F	D	L	X	X	X	W		
i+2			F	F	D	L	X	X	X	W	
i+3				F	F	D	L	X	X	X	W
i+4					F	F	D	L	X	X	X
Anul·lar i+2 fins a i+4 Actualitzar el BTB											
PC+X							F	F	D	L	X

Suposo que l'anul·lació de les instruccions i+2 fins a i+4 es poden fer en un cicle (per exemple utilitzant un Reorder Buffer) i simultàniament amb l'actualització del BTB.

El cost del salt en aquestes circumstancies és de 5 o 6 cicles, en funció de si es pot optimitzar o no el delay.

PROBLEMA2:

Suposem un processador supersegmentat amb el següent pipeline:

Fetch1	fetch2	descodifi-cació	lectura	execució	execució (opcional)	execució (opcional)	escriptura

A on poden haver-hi una, dues o tres etapes d'execució.

Sobre aquest processador executem el següent codi:

```
1:    A=B op C    ; 3 cicles d'execució
2:    D=B op E    ; 1 cicle d'execució
3:    C=A op E    ; 2 cicles d'execució
4:    D=D op A    ; 2 cicles d'execució
```

Contesta:

a) Si suposem una execució amb inici i finalització amb ordre, com s'executarà el codi sobre el processador?, Fes el cronograma i digues quins conflictes de dades poden haver-hi, i com els solucionaries.

b) En cas que la finalització sigui en desordre, torna a respondre les preguntes del punt b).

RESPOSTA:

a) Només hi poden haver conflictes RAW. Els solucionaré amb curt-circuits i, en cas necessari, parant el processador.

A=BopC	F	F	D	L	X	X	X	W				
D=BopE		F	F	D	L	X			W			
C=AopE			F	F	D	L		X	X	W		
D=DopA				F	F	D		L	X	X	W	

b) Només hi poden haver conflictes RAW i WAW. Els solucionaré amb curt-circuits i, en cas necessari, parant el processador.

A=BopC	F	F	D	L	X	X	X	W				
D=BopE		F	F	D	L	X	W					
C=AopE			F	F	D	L		X	X	W		
D=DopA				F	F	D	L		X	X	W	

conflicte estructural