UNIUNEA EUROPEANĂ
GUVERNUL ROMÂNIEI
Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Programare în limbaj de asamblare

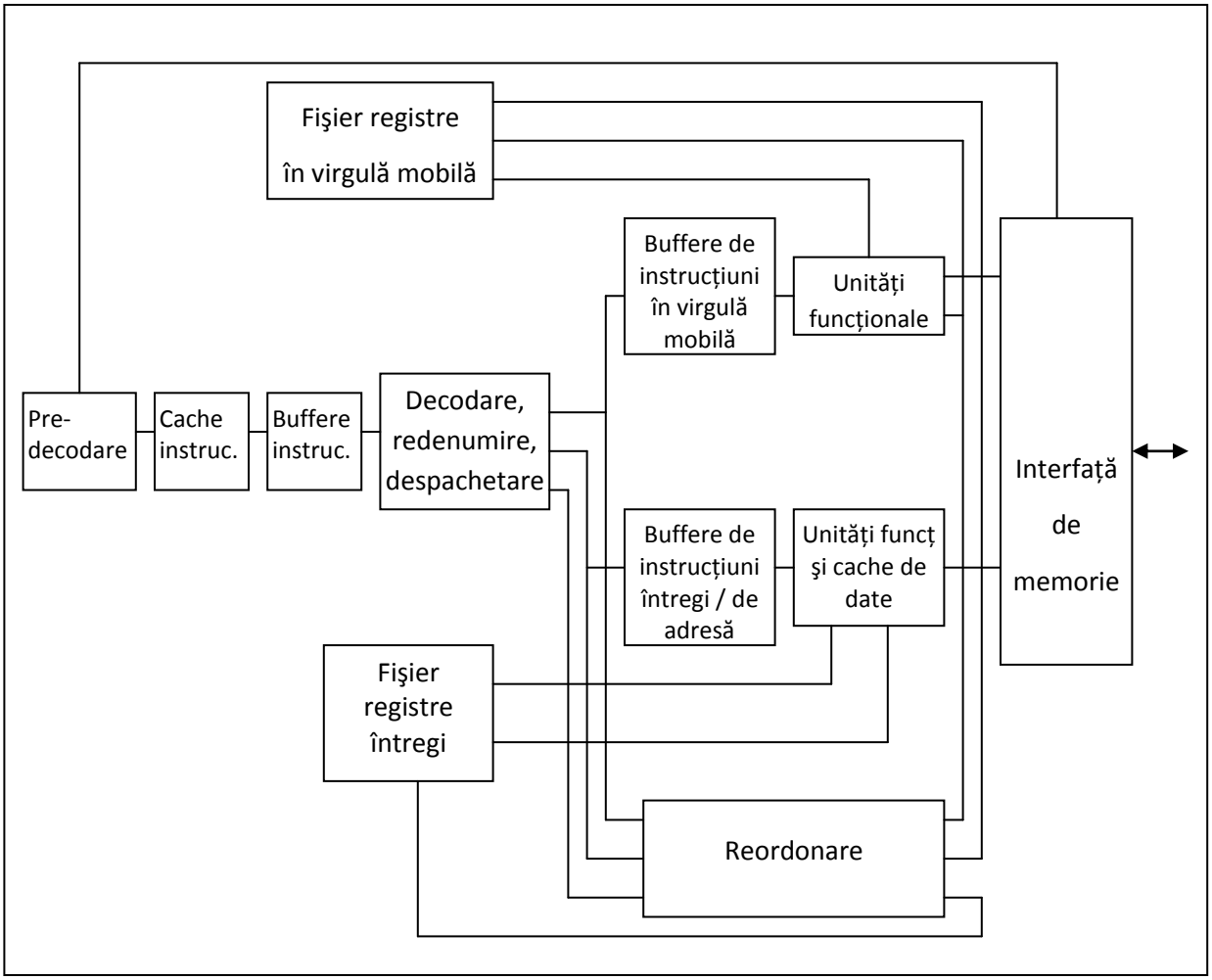
10. Procesoare superscalare.

Procesoare superscalare

Procesoarele superscalare, exploatând paralelismul arhitecturii, pot executa mai multe instrucțiuni într-un ciclu mașină. Metodele superscalare, rezultat al extinderii principiilor ce stau la baza procesoarelor RISC, au fost aplicate începând din anii 1990 unei game largi de arhitecturi, de la cele specifice RISC (DEC Alpha), până la unele evident non-RISC (Intel x86).

Arhitectura internă

Elementele (unitățile) principale ale arhitecturii, prezentate în figura de mai jos, realizează, în ordinea în care sunt parcurse de fluxul de instrucțiuni, următoarele operații: citirea instrucțiunilor (fetch) şi predicția salturilor, decodificarea instrucțiunilor şi analiza dependențelor de date, alocarea unităților de execuție, analiza şi execuția operațiilor cu memoria, reordonarea instrucțiunilor şi înscrierea rezultatelor. La baza acestei arhitecturi stă o implementare a unei „benzi de asamblare“ (pipeline), ale cărei etape se suprapun, într-o oarecare măsură, peste fazele de procesare.



Organizarea hardware a unui procesor superscalar

Un procesor superscalar aduce din memorie şi decodifică, în mod obişnuit, mai multe instrucţiuni simultan. Ca parte a acestui proces, efectele salturilor condiţionate sunt anticipate pentru a asigura un flux neîntrerupt de instrucţiuni. Fluxul de instrucţiuni, adus anticipat în bufferul de prefetch (citire anticipată a instrucţiunilor), este analizat pentru determinarea dependenţelor de date, iar instrucţiunile independente sunt distribuite unităţilor funcţionale, în conformitate cu tipul lor. Aici începe, în paralel, execuţia instrucţiunilor, în funcţie, mai ales, de disponibilitatea operanzilor şi mai puţin de ordinea în care se află aceştia în programul secvenţial. Terminarea execuţiei instrucţiunilor şi înscrierea rezultatelor are loc în aşa fel încât starea logică a procesului este actualizată în ordinea secvenţială a programului, pentru a da posibilitatea tratării precise a unei eventuale întreruperi.

Dependenţele de date apar între instrucţiuni care obţin acces la acelaşi registru sau locaţie de memorie. În acest moment se spune că există un hazard datorat posibilităţii ca instrucţiunile să se execute într-o ordine incorectă. Aceste hazarduri pot fi de tipurile WAR, WAW şi RAW.

Hazardul WAR (Write After Read), adică „scriere după citire“, apare când instrucţiunile trebuie să modifice o locaţie de memorie, dar trebuie să aştepte ca toate instrucţiunile anterioare ce necesită vechea valoare să o citească.

Hazardul WAW (Write After Write), adică „scriere după scriere“, apare când mai multe instrucţiuni modifică aceeaşi locaţie; modificările trebuie realizate în ordinea precizată de programul secvenţial, pentru ca la locaţia respectivă să rămână, în final, valoarea corectă.

Aceste două tipuri de hazarduri sunt artificiale, fiind cauzate de un cod neoptimizat, de numărul limitat de registre, de necesitatea de a economisi memorie sau de ciclurile programului (în care o instrucţiune poate fi dependentă de ea însăşi). În general, aceste dependenţe pot fi eliminate prin redenumirea resurselor respective.

Hazardurile RAW (Read After Write), adică „citire după scriere“ sunt cele mai frecvente, deoarece o instrucţiune poate citi o valoare numai după ce valoarea a fost furnizată de instrucţiunea care o scrie.

Mai toate procesoarele superscalare utilizează o memorie rapidă, de mici dimensiuni, de tip cache, care conţine instrucţiunile cel mai recent executate, pentru a reduce întârzierile, datorate accesului la memoria principală, care este mai lentă, şi pentru a mări numărul de instrucţiuni aflate, la un moment dat, la dispoziţia procesorului. Cache-ul de instrucţiuni este organizat în blocuri conţinând instrucţiuni consecutive. Dacă instrucţiunea ce urmează a fi executată nu este în cache, atunci blocul care o conţine este adus, în întregime, din memoria principală, şi scris în cache.

Pentru a executa mai multe instrucţiuni în paralel, faza fetch trebuie să aducă, din memoria cache, mai multe instrucţiuni într-un ciclu. Din acest motiv, există două cache-uri: unul pentru date şi unul pentru instrucţiuni. Numărul maxim de instrucţiuni adus într-un ciclu ar trebui să fie egal cu rata de decodificare şi execuţie; el este, însă, ceva mai mare, pentru a compensa „miss“-urile în cache (adică lipsa acelei instrucţiuni din cache) şi situaţiile în care se pot aduce doar un număr limitat de instrucţiuni (de exemplu, dacă o instrucţiune transferă controlul unei alte instrucţiuni, aflate în interiorul unui bloc din cache, instrucţiunile ce o preced în bloc sunt inutile).

Citirea normală a instrucţiunilor (fetch) incrementează contorul de program cu numărul de instrucţiuni aduse, pentru a obţine adresa următorului bloc. La întâlnirea unei instrucţiuni de ramificaţie, acest mecanism (fetch) trebuie reorientat pentru a folosi noua adresă indicată de instrucţiunea de salt. Modul de tratare a instrucţiunilor de salt este vital pentru performanţa sistemului. Instrucţiunile de ramificaţie sunt tratate în 4 paşi:

1. recunoaşterea instrucţiunii de ramificaţie;
2. determinarea comportării instrucţiunii de ramificaţie;
3. calculul adresei de ramificaţie;
4. transferul controlului.

1. *Recunoaşterea instrucţiunii de ramificaţie* poate fi realizată mai rapid dacă în cache-ul de instrucţiuni se memorează şi informaţii de decodificare, adică nişte biţi suplimentari generaţi de logica de predecodificare şi memoraţi pe măsură ce instrucţiunile sunt aduse în cache.
2. *Determinarea comportării instrucţiunii de ramificaţie* este dependentă de instrucţiunile precedente incomplet executate. Pentru a evita aşteptarea terminării execuţiei acestora se folosesc metode de predicţie a comportării instrucţiunilor de salt. Unele predictoare folosesc informaţia statică, prezentă în codul binar: unele instrucţiuni de salt determină transferul controlului mai des decât altele, cum este cazul instrucţiunilor de ciclare (compilatoarele pot introduce informaţii despre probabilitatea de efectuare a salturilor, pe baza analizei programului în limbaj de nivel înalt). Majoritatea predictoarelor folosesc informaţia dinamică, disponibilă în momentul execuţiei programului; aceasta se referă la comportarea anterioară a instrucţiunii de salt (curentă, şi eventual a celor precedente) şi este păstrată în tabele de predicţie sau chiar în blocurile din cache care conţin instrucţiunea de salt. Aceste tabele de predicţie sunt organizate într-o manieră asemănătoare memoriilor cache, fiind accesibile cu adresa instrucţiunii de salt care trebuie executată anticipat. Istoria comportării saltului este înregistrată folosind contoare, ce sunt incrementate la execuţia saltului, respectiv decrementate la neîndeplinirea condiţiei de salt. Valoarea acestui contor reflectă comportarea dominantă a instrucţiunii de salt. Se utilizează, mai nou, scheme de predicţie corelate, pe baza a două informaţii: istoria saltului curent şi, respectiv, situarea acestuia în context, pe baza salturilor anterioare. Procesul de citire a instrucţiunilor (fetch) continuă pe baza predicţiei, până în momentul în care condiţia de salt poate fi evaluată pe baza datelor concrete. Acum informaţia de predicţie poate fi actualizată. Dacă predicţia nu a fost corectă, mecanismul de citire (fetch) trebuie reorientat de la o nouă adresă, acum cunoscută cu precizie. În plus, trebuie înlăturate şi efectele produse de execuţia speculativă a instrucţiunilor aduse pe baza predicţiei (în acest caz, eronate).
3. *Calculul adresei de salt* se realizează de obicei, relativ la contorul programului şi poate fi precizat printr-o valoare conţinută în codul instrucţiunii, eliminându-se necesitatea accesului la registre. Necesitatea accesului la registre, pentru calculul adresei de salt a condus la introducerea în tabela de predicţie a adresei folosită ultima oară de instrucţiunea de salt.
4. *Transferul controlului*. Întârzierile datorate complexităţii etapelor anterioare trebuie compensate, de obicei, prin folosirea bufferului de instrucţiuni deja amintit. Anterior, procesoarele RISC foloseau aşa-numitele salturi întârziate, prin care execuţia instrucţiunii următoare celei de salt era suprapusă peste faza de fetch a instrucţiunii care urma conform ramificaţiei. Structura pipeline a procesoarelor superscalare complică mult aplicarea acestei metode.

Decodificarea instrucţiunilor şi analiza dependenţelor de date

În această fază, instrucţiunile sunt scoase din bufferul de instrucţiuni şi examinate pentru stabilirea dependenţelor de control şi de date necesare etapelor următoare din banda de asamblare. Acum se detectează dependenţele reale (hazardurile RAW) şi se elimină celelalte tipuri de hazarduri la registre (WAW, WAR). Tot în această fază, instrucţiunile sunt distribuite în bufferele asociate unităţilor de execuţie, de unde vor fi preluate şi executate într-o fază ulterioară.

În urma decodificării, fiecărei instrucţiuni îi este asociată o structură de date (staţie de rezervare) sub forma unei liste conţinând operaţia care trebuie executată, identificatori pentru elementele care constituie operanzii, respectiv pentru elementele care vor conţine rezultatele (tag-

uri), unitate ocupată sau nu etc. În imaginea statică a programului, aceste elemente sunt precizate de locații de memorie şi de registre logice, indicate de arhitectura procesorului. Pentru a preveni hazardurile WAR şi WAW şi a îmbunătăți gradul de paralelism al execuției, registrelor logice le sunt asociate elemente de memorare fizice. Resursele logice ale procesorului pot fi văzute, deci ca nume simbolice, folosite pentru a descrie operația dorită. În cursul execuției paralele, unei resurse logice îi pot fi asociate mai multe valori, memorate în resurse fizice distincte, corespunzătoare diferitelor momente de execuție ale procesului secvențial.

Când o instrucțiune creează o nouă valoare pentru un registru logic, acestuia îi este asociată o locație fizică şi este redenumit în structura de date asociată, în faza de decodificare, cu identificatorul locației ce va conține valoarea nou creată. În orice instrucțiune ulterioară în care registrul logic apare ca operand sursă, el va fi redenumit cu identificatorul respectivei locații.

O primă metodă de redenumire foloseşte un set de registre fizice, mai mare decât cel logic. Asocierea dintre un registru logic şi valoarea conținută într-un registru fizic se face pe baza unei tabele de mapare. Redenumirea are loc la decodificarea instrucțiunilor, în ordinea secvențială a programului. Registrele fizice neocupate sunt păstrate într-o listă de resurse libere, de unde sunt luate, în momentul în care sunt atribuite unui registru logic prin intermediul tabelei de mapare. Dacă lista de resurse este goală, decodificarea instrucțiunilor următoare este temporar oprită. Asocierea dintre un registru logic şi unul fizic are loc atunci când registrul logic este redenumit din nou şi ultima instrucțiune, în care apare ca operand sursă, şi-a încheiat execuția. În acest moment, registrul fizic este trecut în lista de resurse libere.

O altă metodă de redenumire are numărul de registre logice egal cu acela al de registrelor fizice. Se utilizează un buffer de reordonare (“reorder buffer“), care conține câte o intrare pentru fiecare instrucțiune în curs de execuție. Bufferul de reordonare este implementat ca o memorie (de tip FIFO), instrucțiunile fiind introduse aici în ordinea secvențială din program. În momentul în care instrucțiunile îşi termină execuția, în intrarea corespunzătoare din buffer se înscriu rezultatele produse. Aceste rezultate vor fi înscrise în setul de registre doar atunci când intrarea respectivă ajunge ultima în buffer. Setul de registre va conține, deci, întotdeauna valori corespunzătoare execuției secvențiale a instrucțiunilor, un lucru deosebit de util din perspectiva tratării precise a excepțiilor. Valoarea logică a unui registru poate fi conținută fie în setul fizic de registre, fie în bufferul de reordonare. Când o instrucțiune este decodificată, rezultatul ei este asociat cu o intrare din bufferul de reordonare şi tabela de mapare este actualizată. Tabela de mapare este accesibilă folosind registrele sursă ale instrucțiunilor şi indică dacă valorile corespunzătoare acestora se află în setul de registre sau în bufferul de reordonare. În concluzie, indiferent de metoda utilizată, redenumirea registrelor elimină dependențele artificiale, datoare hazardurilor WAW şi WAR.

Alocarea unităților de execuție paralelă a instrucțiunilor

În fazele anterioare, ale benzii de asamblare, se formează structuri de date asociate instrucțiunilor conținând informații referitoare la operația care urmează să se execute şi la modul de localizare a operanzilor fizici. Următoarea etapă este aceea de a determina, pe baza acestor informații şi ținând seama de disponibilitatea resurselor, instrucțiunile care pot fi executate. Pentru un model ideal de procesor superscalar, având resurse infinite, o instrucțiune este gata de execuție în momentul în care operanzii sursă sunt disponibili. Restricțiile care se impun acestui model provin din limitarea numărului de unități de execuție de un anumit tip, interconectarea acestora cu setul de registre sau bufferul de reordonare, precum şi metoda de organizare a bufferelor care conțin structurile de date, asociate instrucțiunilor (fereastra de execuție), ce pot fi de tip: coadă simplă, coadă multiplă sau stații de rezervare.

În primul caz, coada simplă, execuția instrucțiunilor are loc în ordinea secvențială din program (in-order), iar redenumirea registrelor nu este necesară. Disponibilitatea operanzilor poate fi controlată prin biți de rezervare asociați fiecărui registru, un registru fiind rezervat pe întreaga durată a execuției unei instrucțiuni care îl modifică. Din punct de vedere al disponibilității operanzilor, o instrucțiune își poate începe execuția în momentul în care operanzii săi nu sunt rezervați de instrucțiunile anterioare.

Pentru cea de-a doua metodă instrucțiunile se execută „in-order", din fiecare coadă, dar nu neapărat în ordinea în care apar în programul secvențial, adică „out-of-order" între cozi. Fiecare coadă conține instrucțiuni de un anumit tip, de exemplu fie instrucțiuni aritmetice întregi, fie în virgulă mobilă, fie instrucțiuni de transfer (load/store). Redenumirea registrelor poate fi folosită într-o formă restrânsă, luând în considerare doar pe aceia care pot fi operanzi comuni instrucțiunilor aflate în mai multe cozi (de exemplu registrele care pot fi operanzi pentru instrucțiunile de transfer – de tip load/store).

O a treia metodă, stațiile de rezervare, propuse în algoritmul lui Tomasulo, permite execuția complet „out-of-order" a instrucțiunilor, evident ținând seama de hazardurile RAW. Stațiile de rezervare sunt structuri de date asociate fiecărei instrucțiuni, având următoarele intrări: operația ce trebuie executată, identificatori pentru fiecare operand sursă şi destinație, valorile propriu-zise ale operanzilor sursă, biții de validare, care indică că valorile operanzilor sursă, conținute în stația de rezervare, sunt corecte. Când o instrucțiune își termină execuția şi produce un rezultat, identificatorul destinație este comparat cu identificatorii sursă ai stațiilor de rezervare asociate instrucțiunilor care aşteaptă date, iar în caz de potrivire, rezultatul este înscris în câmpul valoare şi validat. În momentul în care toți operanzii sunt disponibili, instrucțiunea din stația de rezervare se poate executa. Algoritmul reduce mult din presiunea, la citire, asupra seturilor de registre, prin transferul (bypassing) rezultatelor direct între stațiile de rezervare.

Operațiile cu memoria

În cazul procesoarelor superscalare, operațiile cu memoria necesită o tratare specială. Cele mai multe seturi de instrucțiuni RISC permit accesul la memorie doar prin instrucțiuni specifice *load* şi *store*. Pentru a reduce latența operațiilor cu memoria sunt folosite ierarhii de memorie, în ideea că accesele cele mai frecvente la memorie vor fi rezolvate de memoriile cache, aflate la nivelurile inferioare ale acestor ierarhii. Practic, toate procesoarele actuale conțin cel puțin un cache de date, tendința generală fiind de utilizare a mai multor niveluri de cache.

Spre deosebire de instrucțiunile aritmetice/logice, ai căror operanzi pot fi identificați încă din faza de decodificare, locațiile de memorie la care obțin acces instrucțiunile de transfer (load/store) nu pot fi determinate decât în faza de execuție. Determinarea locației de memorie necesară unei astfel de instrucțiuni necesită un calcul de adresă, adică o adunare de întregi. După calculul adresei logice este necesară o translatare a acesteia, pentru obținerea adresei fizice (datorită folosirii mecanismului de paginare). După obținerea adresei valide, se poate executa operația respectivă cu memoria.

Ca şi în cazul operațiilor cu registre, este de dorit ca un număr cât mai mare de operații cu memoria să se execute cât mai rapid. Acest lucru poate implica: reducerea latenței acceselor la memorie, executarea simultană a mai multor operații cu memoria, suprapunerea execuției operațiilor cu memoria cu execuția unor instrucțiuni de alte tipuri, executarea out-of-order a operațiilor cu memoria. Totuşi, datorită numărului de locații de memorie, mult mai mare decât cel al registrelor, precum şi datorită faptului că memoria poate fi adresată şi indirect, prin registre, soluțiile anterioare nu sunt aplicabile. În loc de a folosi tabele de redenumire cu intrări pentru toate locațiile de memorie, ca în cazul registrelor se păstrează informații numai pentru subsetul de locații asupra cărora se

execută operații, la un moment dat. Pentru a permite accese multiple la memorie, ierarhia de memorii trebuie să fie de tip multiport, permițând mai multe accese într-un ciclu. Uzual, este suficient ca numai memoria cache primară să fie de tip multiport, deoarece este probabil ca majoritatea acceselor să fie rezolvate aici. Mai mult, transferurile în cache din memoria de la nivelurile superioare, au loc sub formă de blocuri ce includ mai multe cuvinte de date consecutive. Pentru a permite suprapunerea operațiilor cu memoria cu alte instrucțiuni (fie cu memoria, fie de alte tipuri), ierarhia de memorie trebuie să fie fără blocare; cu alte cuvinte, dacă data solicitată nu se află în cache, următoarele instrucțiuni trebuie să se poată executa, fără a aştepta terminarea instrucțiunii blocate, şi la fel şi în cazul datelor.

Suprapunerea execuției operațiilor cu memoria şi executarea lor out-of-order presupune rezolvarea hazardurilor şi conservarea semanticii programului secvențial. Dificultatea principală constă în determinarea în timp util a adreselor fizice de acces pentru instrucțiunile de transfer, pentru a le procesa out-of-order. Se utilizează buffere de adrese, de tip FIFO, în ordinea în care apar în program, în care sunt înscrise adresele la care obțin accesul instrucțiunile store în curs de execuție. Adresele sunt menținute aici până când datele necesare execuției instrucțiunii sunt disponibile, iar instrucțiunea se poate încheia, prin scrierea efectivă în memorie, în momentul în care ajunge ultima în buffer-ul FIFO. Adresele la care obțin accesul instrucțiunile load sunt comparate cu adresele din buffer, iar în caz de potrivire trebuie să aştepte terminarea instrucțiunii store corespunzătoare.

Înscrierea rezultatelor

În această ultimă fază a execuției unei instrucțiuni, efectele ei modifică starea logică a procesului din care face parte. Scopul acestei faze este de a implementa un mecanism de modificare secvențială a acestei stări logice, deşi instrucțiunile programului au fost executate speculativ, out-of-order.

Într-o primă tehnică, starea maşinii este dublată de un buffer „history“, actualizat conform ordinii secvențiale a programului. Instrucțiunile actualizează starea maşinii pe măsură ce sunt executate; când este necesară o stare precisă (de exemplu, în cazul unei excepții), starea este regăsită din acest buffer „istorie“.

Altă metodă foloseşte separarea stării logice (arhitecturale) a maşinii de starea sa fizică. Starea fizică este modificată imediat după terminarea execuției instrucțiunilor, în timp ce starea logică se actualizează în ordinea secvențială a programului, făcând astfel transparentă execuția speculativă a instrucțiunilor. Starea fizică a maşinii este reprezentată de bufferul de reordonare, de unde rezultatele produse de instrucțiuni sunt mutate, în această fază, în setul de registre logice sau în memorie (pentru instrucțiunile store). Pe lângă identificatorii logici şi valorile operanzilor fiecărei instrucțiuni, în bufferul de reordonare se înscrie şi valoarea registrului contor de instrucțiuni (PC), necesar restaurării stării precise a maşinii, în cazul apariției unei excepții.

Deşi tehnica bufferului „history“ a fost implementată în unele procesoare superscalare, tehnica preferată rămâne cea cu buffer de reordonare, deoarece, pe lângă implementarea unui mecanism de excepții precis, îndeplineşte şi funcția de redenumire dinamică a registrelor.