ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1996
Seria: INFORMATYKA z. 31 Nr kol. 1348

Michał KOLANO

WYBRANE ASPEKTY BUDOWY I DZIAŁANIA PROCESORÓW TYPU RISC

Streszczenie. W niniejszym artykule omówione zostały wybrane aspekty budowy i działania współczesnych procesorów typu RISC. Część pierwsza zawiera skrócony opis historii architektury RISC oraz przegląd najważniejszych różnic pomiędzy procesorami RISC i CISC. W części drugiej opisano wybrane elementy składowe współczesnego procesora o zredukowanej liście rozkazów, kładąc nacisk na rolę pamięci podręcznej w układzie SMP.

SELECTED ASPECTS OF ARCHITECTURE AND OPERATION OF RISC PROCESSORS

Summary. The paper presents some aspects of architecture and operation of RISC microprocessors. First part gives a short description of RISC history and also discusses main differences between RISC and CISC processors. Second parts describes main parts of the modern reduced instruction set microprocessor, putting the stress on the function of cache memory in SMP parallel computer.

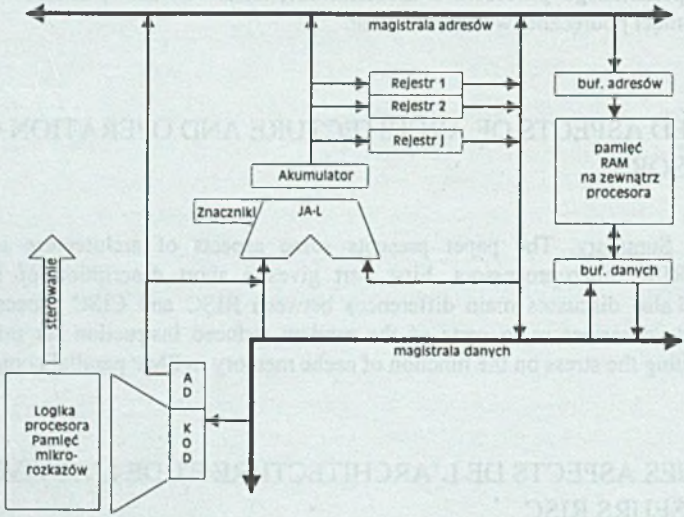
QUELQUES ASPECTS DE L'ARCHITECTURE ET DE L'ACTIVITÉ DES PROCESSEURS RISC

Résumé. L'exposé suivant présente quelques aspects de l'architecture et de l'activité des microprocesseurs RISC. La premiere partie rappelle l'histoire des processeurs à jeu d'intructions réduit el débat également des principales différences entre processeurs RISC et CISC. La seconde partie décrit succintement les principales caractéristiques des nouveaux processeurs RISC, insistant sur les fonctions de mémoire cache sur les machines paralléles SMP.

1. Wprowadzenie

Mikroprocesor jest układem o wielkim stopniu scalenia, służącym do przetwarzania informacji i sterowania innymi częściami komputera. Jest on najważniejszą częścią komputera i to głównie od niego zależy jego moc obliczeniowa. Obecnie stosowane są dwa główne typy procesorów: procesory o rozbudowanej i procesory o zredukowanej liście rozkazów (ang. Complex / Reduced Instruction Set Computer). Wszystko wskazuje na to, że w ciągu kilku najbliższych lat technika RISC wyprze technikę CISC w typowych zastosowaniach [10].

Architektura pierwszych procesorów była prosta. Potrafiły one wykonać niewiele rozkazów i były bardzo powolne, a to głównie za sprawą powolnej komunikacji procesora z pamięcią. Taki sposób wykonywania rozkazu charakteryzuje między innymi *Maszynę W* o architekturze jak na rys. 1 [1].



Rys. 1. Architektura maszyny W o rozwiniętej drodze adresów
Fig. 1. Architecture of W-machine with extended address buses

W pamięci, która najczęściej znajduje się poza układem procesora, przechowywany jest program. Program jest ciągiem rozkazów, które sterować mają pracą procesora i całego komputera. Procesor pobiera rozkaz z komórki pamięci o adresie przechowywanym w specjalnym rejestrze, zwanym licznikiem rozkazów. Adres komórki z rozkazem

wystawiany jest na magistralę adresową. Po odczycie kodu rozkazu poprzez magistralę danych (faza *pobrania*) następuje jego analiza w specjalnym rejestrze (faza *dekodowania*). Specjalny układ dekoduje kod rozkazu i wybiera odpowiadający mu ciąg przesłań międzyrejestrowych. W następnych taktach odbywa się wykonanie rozkazu poprzez wysyłanie przez układ sterujący odpowiednich sygnałów (faza *wykonania*). Sygnały te sterują rejestrami, jednostką arytmetyczno-logiczną oraz komunikacją procesora z pamięcią. Po zakończeniu tej fazy rozkazu następuje zapis wyników operacji do odpowiedniego rejestru bądź do pamięci (faza *zapisu*). Pierwsze procesory potrafiły wykonać niewiele rozkazów i pod tym względem przypominały dzisiejsze procesory RISC.

W latach 60. i 70. nastąpił wyraźny wzrost zapotrzebowania na moc obliczeniową komputerów. Ponieważ w owych czasach istniała duża dysproporcja pomiędzy szybkością działania procesora i szybkością działania pamięci komputera, stąd pomysł, aby kontakt procesora z pamięcią był jak najrzadszy. Najlepszym rozwiązaniem dla wyrównania szybkości tych dwóch elementów było pobranie kodu rozkazu z powolnej pamięci operacyjnej, a następnie realizowanie jego złożonej treści za pomocą wielu mikrorozkazów, ale już w obrębie szybkiego procesora. W ten sposób wiele rozkazów zostało zastąpionych jednym. Powstały takie rozkazy jak np. rozkaz *MOVTUC* komputera VAX:

> „*MOVTUC scrlen, scraddr, esc, tbladdr, dstlen, dstaddr* - ciąg źródłowy określony przez długość *scrlen* i adres początku *scraddr* jest „tłumaczony" i zastępuje ciąg docelowy określony przez długość *dstlen* i adres *dstaddr*. „Tłumaczenie" jest dokonywane przez użycie 256-bajtowej tablicy, której adres początkowy jest określony przez *tbladdr*. Wybrane w ten sposób znaki zastępują znaki ciągu docelowego. Tłumaczenie jest dokonywane do momentu, gdy przetłumaczony znak jest równy jednemu z ciągów, lub do wyczerpania znaków jednego z ciągów."

Rozkaz ten realizowany jest przez kilkadziesiąt (kilkaset) taktów zegarowych, a mógłby z powodzeniem zostać zastąpiony sekwencją prostych rozkazów. Jednak w dobie powolnej pamięci programu zastosowanie techniki przeniesienia dużych fragmentów programu do pamięci mikroprogramów dawało dobre rezultaty.

Mimo że wraz z rozwojem techniki mikroprocesorowej pojawiły się nowe rozwiązania techniczne, takie jak np.: praca potokowa, rozwiązania superskalarne, pamięci podręczne, które implementowane były w istniejących procesorach CISC, nie zmieniło to zasady utrzymywania w nich rozbudowanych list rozkazów i wielu trybów adresacji.

Dopiero pojawienie się coraz szybszych pamięci półprzewodnikowych, zwłaszcza w latach osiemdziesiątych, radykalnie zmieniło relację czasową pamięć - procesor. Organizując listę rozkazów na tyle prostą, iż da się je wykonać w jednym cyklu procesora,

można traktować ich sekwencję jako mikrokod, umieszczony jednak nie w procesorze, lecz w pamięci operacyjnej komputera.

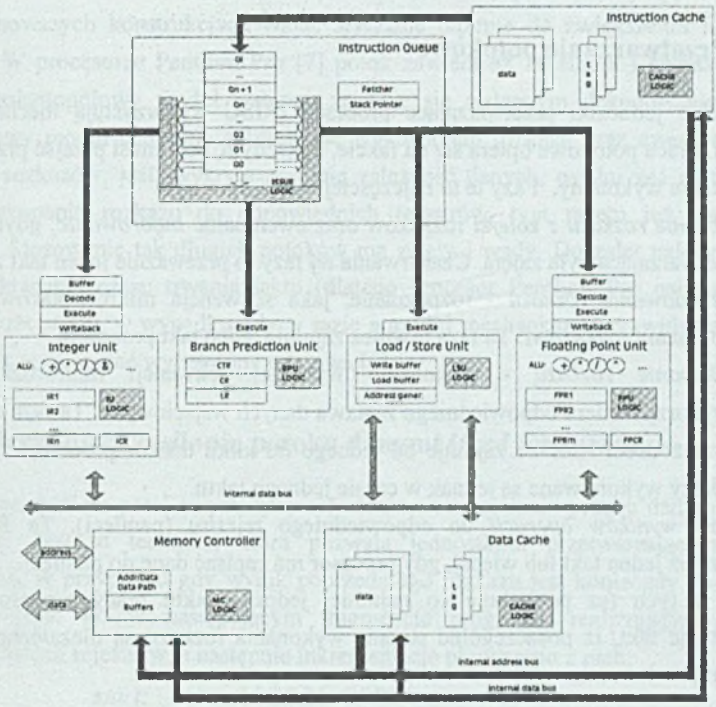
W połowie lat siedemdziesiątych przebadano wiele programów komputerowych tworzonych w języku asemblera i osiągnięto wyniki wskazujące, iż dominuje w nich grupa rozkazów najprostszych. Zamiast użycia złożonych rozkazów użytkownicy woleli wprowadzać własne konstrukcje, składające się z wielu rozkazów prostych. To spostrzeżenie zmobilizowało firmę IBM do pracy nad pierwszym procesorem typu RISC. Wynikiem jej prac był procesor typu RISC IBM 801, którego kolejna wersja została użyta w komputerze IBM PC/RT wprowadzonym na rynek w 1986 r. Termin RISC został użyty po raz pierwszy w roku 1980 przez Davida Pattersona, konstruktora procesorów na Uniwersytecie Berkeley. Konstruując kolejne wersje procesora RISC uzyskał on układ, który nawet bez stosowania subtelnych technik kompilacyjnych przewyższał w zakresie arytmetyki stałoprzecinkowej procesor 11/780 znanej rodziny komputerowej VAX.

W obecnych systemach komputerowych dominują dwa typy architektur procesorów: procesory o architekturze CISC oraz procesory o architekturze RISC. Linię podziału pomiędzy nimi stanowi liczba rozkazów zastosowanych w procesorze, w szczególności zaś liczba rozkazów komunikacji z pamięcią oraz liczba trybów adresacji. Część konstrukcji charakteryzuje się architekturą mieszaną, tj. pozostawiono w nich rozbudowane listy rozkazów, które w czasie wykonania tłumaczone są na ciąg prostszych rozkazów RISC. Taką budowę mają na przykład procesory PentiumPro i NexGen [6].

2. Budowa i działanie procesorów typu RISC

W budowie wielu dostępnych na rynku procesorów typu RISC zauważyć można pewne podobieństwo. W większości z nich znaleźć można kilka pracujących równolegle jednostek przetwarzających, zaawansowane kolejki rozkazów, pamięci podręczne rozkazów i danych, układy zarządzające magistralami, układy dostępu do pamięci oraz złożoną logikę sterującą. Na rys. 2 przedstawiono wewnętrzną budowę procesora typu RISC. Architektura ta nie reprezentuje żadnego konkretnego procesora, jest raczej pewnym uogólnieniem. Najważniejsze z elementów procesora to: jednostki przetwarzające pracujące w trybie potokowym (ang. *Integer Unit, Floating Point Unit)*, jednostka przewidywania wyniku skoku (ang. *Branch Prediction Unit*), jednostka zarządzająca odczytem i zapisem danych z i do pamięci (ang. *Load / Store Unit*), kolejka rozkazów (ang. *Instruction Queue*), pamięci podręczne (ang. *Instruction Cache, Data Cache*) oraz jednostki sterujące magistralami procesora i pamięcią.

Przetwarzanie rozkazów odbywa się *superskalarnie* w obrębie procesora, co oznacza równoczesną aktywność kilku jednostek, oraz *potokowo* w obrębie każdej z jednostek. Tak więc równolegle przetwarza się kilka kolejnych rozkazów.



Rys. 2. Architektura procesora typu RISC
Fig. 2. RISC architecture

2.1. Przetwarzanie superskalarne

Procesor RISC składa się z kilku niezależnych (do pewnego stopnia) jednostek przetwarzających. Każda z nich składa się z zestawu niezbędnych rejestrów, jednostki arytmetyczno-logicznej oraz wbudowanego układu sterującego. Jednostki te to w zasadzie uproszczone procesory, które potrafią realizować tylko określony typ rozkazów. Wszystkie układy mogą pracować równolegle, przynajmniej tak długo, jak długo nie istnieje powiązanie danych, na których operują. Nie ma żadnych przeszkód, aby w procesorze istniała więcej niż jedna jednostka danego typu.

Aby stworzyć odpowiednie warunki do równoległego przetwarzania rozkazów w wielu jednostkach przetwarzających, konieczne było wprowadzenie bufora, z którego jednostki

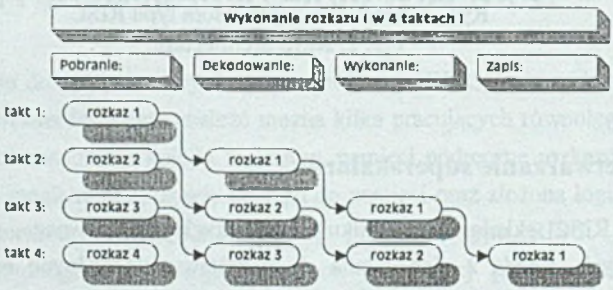
mogłyby pobierać odpowiednie rozkazy. Rolę tę w procesorze RISC pełni *kolejka rozkazów* wraz z odpowiednim układem logicznym rozstrzygającym kwestie związane ze zmianą pierwotnej kolejności rozkazów (rozdz.2.6).

2.2. Przetwarzanie potokowe

Wszystkie jednostki przetwarzające procesora RISC wykorzystują mechanizm pracy potokowej. Praca potokowa opiera się na fakcie, iż każdy rozkaz musi przejść przez kilka faz, zanim zostanie wykonany. Fazy te to najczęściej:

- *Pobranie rozkazu z kolejki* rozkazów oraz ewentualne *buforowanie*, gdyby jednostka przetwarzająca była zajęta. Czas trwania tej fazy to przeważnie jeden takt zegarowy.
- *Zdekodowanie rozkazu* - rozpoznanie, jaka sekwencja mikrorozkazów odpowiada pobranemu rozkazowi. Ta faza również zajmuje jeden takt procesora.
- *Wykonanie rozkazu* - wykonanie wybranej sekwencji mikrorozkazów wraz z wykorzystaniem odpowiedniego zestawu danych wejściowych. Ta faza w zależności od złożoności rozkazu zajmuje od jednego do kilku taktów procesora. Podstawowe rozkazy wykonywane są jednak w czasie jednego taktu.
- *Zapis wyników operacji* do odpowiedniego rejestru (pamięci). Ta faza zajmuje również jeden takt lub więcej, gdy procesor ma zapisać dane do pamięci.

Każda z tych faz potrzebuje co najmniej jednego taktu pracy procesora (rys. 3). Wykorzystując fakt, iż poszczególne stopnie wykonania rozkazu są niezależne, stworzono mechanizm potokowania.



Rys. 3. Mechanizm pracy potokowej
Fig. 3. Pipeline mechanism

Gdy wyniki wykonania pierwszego rozkazu są zapisywane, drugi może być wykonywany, trzeci dekodowany, a czwarty pobierany. W następnej chwili następuje przesunięcie przetwarzanych rozkazów o kolejny stopień. Tak więc w tej samej chwili czasu jednostka przetwarza cztery kolejne rozkazy z ciągu rozkazowego. Faktycznie,

choć wykonanie pojedynczego rozkazu zajęłoby 4 takty, to grupa rozkazów będzie wykonywana w taki sposób, iż co jeden takt wykonany zostanie jeden rozkaz. Stosowanie *k*-stopniowych jednostek potokowych wnosi duże przyspieszenie w działaniu programu. Dla liczby rozkazów zmierzających do nieskończoności przyspieszenie to dąży do *k*.

W najnowszych konstrukcjach widać wyraźnie dążenie do zwiększenia liczby stopni w potoku. W procesorze Pentium Pro [7] potok zawiera aż 14 stopni i składa się z kilku sekcji. Ośmiostopniowy moduł wstępny zajmuje się wstępnym dekodowaniem rozkazu, trzystopniowy moduł centralny zajmuje się wykonaniem rozkazu oraz ewentualną zmianą kolejności rozkazów, jeśli wykryta zostanie zależność danych, ostatni zaś moduł zapisuje wyniki wykonania rozkazu do odpowiednich rejestrów, tym razem już we właściwej kolejności. Stosowanie tak długich potoków ma zalety i wady. Do zalet należy możliwość dalszego skracania czasu trwania taktu (dlatego procesor Pentium Pro osiąga 150MHz), a do wad duże straty, w wypadku gdy w razie pomyłki mechanizmu przewidywania wyniku skoku potok musi zostać wypełniony od nowa [6].

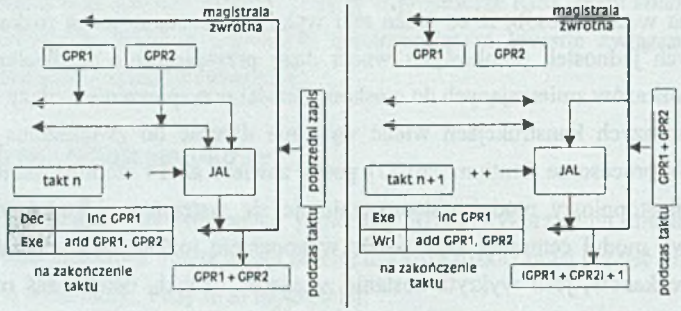
2.3. Mechanizm zasilania potoku danymi (*feed forwarding*)

Z przetwarzaniem potokowym łączy się mechanizm zasilania potoku danymi (ang. *feed forwarding*). Jest to technika, która pozwala jednostkom przetwarzającym na pracę bez opóźnień, w przypadku gdy wynik poprzedniego rozkazu jest konieczny do wykonania następnego (rys. 4). W następującym fragmencie programu, realizującym sumowanie zawartości dwóch rejestrów, a następnie inkrementację pierwszego z nich:

start: *iadd GPR1, GPR2*
iinc GPR1

aby poprawnie wykonać rozkaz *iinc GPR1*, już w trakcie fazy wykonywania tego rozkazu należałoby znać wartość rejestru *GPR1*. Oznacza to konieczność wstrzymania wykonania rozkazu *iinc GPR1* na jeden takt, tak aby wynik wykonania *iadd GPR1, GPR2*, czyli suma *GPR1+GPR2*, był już zapisany do właściwego rejestru, tj. *GPR1*. Problem unikania takich opóźnień został rozwiązany poprzez użycie magistrali zwrotnej i rejestrów zwrotnych (ang.*writeback bus, writeback register*).

Kiedy procesor RISC kończyć będzie wykonanie rozkazu *iadd GPR1, GPR2*, prześle sumę na specjalną magistralę zwrotną. W następnym takcie wartość ta będzie równocześnie wpisywana do rejestru *GPR1*, jak również wykorzystywana przy inkrementowaniu. Mechanizm *feed forwarding* wbudowany jest w jednostki przetwarzające dane stało- i zmiennoprzecinkowe.



Rys. 4. Mechanizm *feed forwarding*
Fig. 4. Feed forwarding mechanism

2.4. Jednostki przetwarzające stało- i zmiennoprzecinkowa

Jednostka stałoprzecinkowa IU (ang. *Integer Unit*) jest wyspecjalizowanym, autonomicznym układem zajmującym się arytmetyką liczb całkowitych. Oznacza to (w zależności od procesora) przetwarzanie danych typu bajt, słowo, podwójne słowo, poczwórne słowo. Jednostka stałoprzecinkowa zawiera najczęściej kilkanaście rejestrów o takiej samej konstrukcji i długości (np. 32 rejestry 64-bitowe). Rejestry te nie różnią się od siebie i w żaden sposób nie są wyróżniane w czasie realizacji rozkazów. Oprócz tych rejestrów w jednostce znajduje się rejestr kontrolny zawierający znaczniki wykonania ostatniej operacji (np. *zero, przepełnienie, zgłoszona próba dzielenia przez zero* itp.) oraz rejestry systemowe służące do sterowania pracą procesora.

W przypadku gdy w układzie brak wyspecjalizowanej jednostki realizującej rozkazy dostępu do pamięci, jednostka IU przejmuje także tę rolę. Zdarza się również, iż jednostka IU współpracuje z jednostką zmiennoprzecinkową podczas realizacji skomplikowanych rozkazów zmiennoprzecinkowych.

Jednostka IU, jak każda z jednostek, działa w trybie potokowym. Oznacza to, iż jest zdolna do wykonywania rozkazów praktycznie w jednym takcie. Przetwarzanie rozkazu obejmuje wszystkie podstawowe fazy. Liczba stopni potoku może zmieniać się w zależności od konkretnego procesora.

Jednostka zmiennoprzecinkowa FPU (ang. *Float Point Unit*) zawiera zestaw wyspecjalizowanych rejestrów służących do przechowywania danych zmiennopozycyjnych (np. 32 rejestry 80-bitowe). Opiera się ona na przetwarzaniu znormalizowanych (zgodnie z IEEE 754) typów danych (całkowite, całkowite długie, rzeczywiste krótkie, długie). Podstawowe grupy rozkazów to: rozkazy przesyłania danych, grupa rozkazów

arytmetycznych, grupa rozkazów porównania, grupa rozkazów obliczania funkcji przestępnych (log, tg, arctg), grupa rozkazów sterujących.

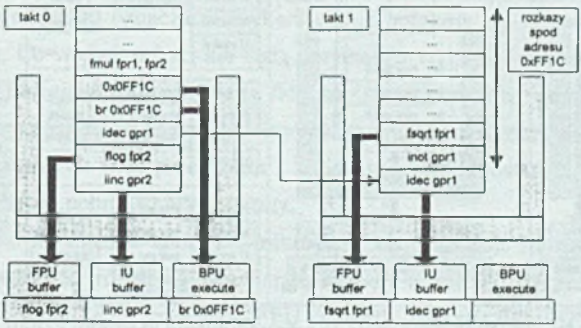
Jednostka *FPU* działa podobnie jak *IU* w trybie potokowym, z tym że wiele rozkazów zmiennoprzecinkowych wykonywanych jest (faza *wykonania*) znacznie dłużej niż w jednym takcie procesora. Stąd w niektórych systemach stosuje się dodatkowe stopnie buforujące, pozwalające na pobranie i przechowywanie rozkazu z kolejki, w czasie gdy jednostka zajęta jest obliczeniami [2].

Ponieważ algorytmy zmiennoprzecinkowe są znacznie bardziej skomplikowane, niektóre konstrukcje (np. PowerPC [2]) przewidują podzielenie fazy wykonywania na dwie części. Pozwala to na bardziej optymalne wykorzystanie jednostki.

2.5. Przewidywanie wyniku skoku - jednostka BPU

Potencjalne trudności w pracy procesora mogą być spowodowane dużą liczbą rozgałęzień programu po natrafieniu na rozkaz warunkowy. Metodą pozwalającą na rozwiązanie problemów tego rodzaju jest przewidywanie sekwencji przyszłych rozkazów. Można to porównać do określania prawdopodobnego biegu rzeki w określonych warunkach terenowych. Według różnych opinii dzięki tej technice można zwiększyć wydajność procesora RISC o około 30%.

Brak takiego układu powoduje, iż w przypadku natrafienia przez procesor na rozkaz skoku warunkowego nastąpiłaby konieczność opróżnienia kolejki rozkazów oraz wszystkich potoków, które pobrały już rozkazy następujące po rozkazie skoku. Procesor zmuszony byłby do wczytania nowej zawartości kolejki adresów, począwszy od adresu, do jakiego nastąpił skok warunkowy. Oznacza to wprowadzenie kilkunastu taktów niepotrzebnego opóźnienia.

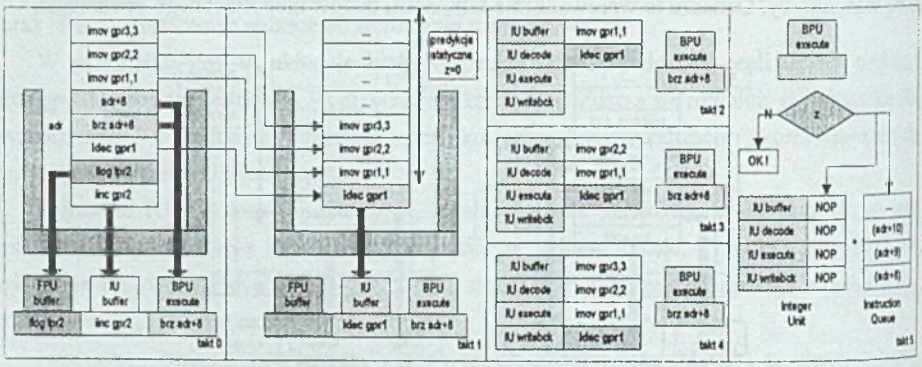


Rys. 5. Przewidywanie wyniku skoku bezwarunkowego do adresu *0x0FF1C*
Fig. 5. Prediction of the unconditional branch to *0x0FF1C*

W programach występują dwa rodzaje skoków: bezwarunkowe i warunkowe. *Skok bezwarunkowy* powoduje przejście do wykonania rozkazu znajdującego się pod wskazanym adresem. Jednostka BPU (ang. *branch prediction unit*) wychwytuje rozkazy skoku bezwarunkowego i odpowiednio modyfikuje kolejkę rozkazów, wstawiając w miejsce wykonania skoku ciąg rozkazów z pamięci, począwszy od adresu docelowego (rys. 5). Tak więc żaden takt opóźnienia nie jest wprowadzany, a kolejka zachowuje się tak, jakby rozkazu skoku nigdy nie było.

Znacznie trudniejszym przypadkiem jest rozkaz *skoku warunkowego*, który zawsze pociąga za sobą wiele możliwości dalszego przebiegu sterowania w programie. Może on, w zależności od spełnienia określonego warunku (np. w zależności od wyniku operacji porównania), zmienić lub pozostawić niezmienioną sekwencję wykonywanych rozkazów. W kodzie generowanym przez kompilatory języków wysokiego poziomu rozkazy skoków warunkowych stanowią 15-20% kodu. W przypadku języków obiektowych współczynnik ten może przekroczyć nawet 35%. Każda pętla, porównanie czy instrukcja wyboru oznaczają (często wielokrotne) wykonanie rozkazu skoku warunkowego. Obliczono, iż rzeczywisty udział skoków warunkowych (łącznie z rozkazami następującymi po nich i wchodzącymi do kolejki) jest średnio równy 50% kodu programu. Dobrze zaprojektowany system przewidywania wyniku skoku powinien prawidłowo przewidzieć efekt wykonania 80% skoków. Procesor P6 osiąga poziom skuteczności rzędu 90%.

Proces przewidywania wygląda następująco: układ BPU sprawdza kolejne rozkazy znajdujące się w kolejce. Jeśli wykryje, iż jeden z rozkazów to rozkaz skoku warunkowego, wówczas stara się przewidzieć, w jakim kierunku skok ten nastąpi. BPU może posługiwać się dwoma metodami przewidywania skoków.



Rys. 6. Statyczna metoda przewidywania kierunku skoku
Fig. 6. Static branch prediction

Metoda statyczna przewidywania skoku polega na wykorzystaniu faktu, iż podczas wykonania programu efektem 80% skoków warunkowych jest cofnięcie wykonania do rozkazu spod wcześniejszego adresu. Jednostka BPU zakłada, że wszystkie skoki wstecz zostaną wykonane, natomiast nie zostanie wykonany żaden skok do przodu. Dzięki technice statycznego przewidywania procesor może zacząć pobierać następny rozkaz występujący w przewidzianym kierunku przetwarzania (metoda zastosowana np. w PowerPC).

Na rys. 6 przedstawiono statyczną metodę przewidywania wyniku skoku. Układ BPU wykrywa rozkaz skoku warunkowego *brz adr+8*, gdzie *adr+8* jest adresem komórki odległej o 8 bajtów od adresu skoku (w przykładzie komórkę tę oznaczano jako *etykieta*)

```
start:    iinc GPR2
          flog FPR2
          idec GPR1            ;w pamięci pod adresem
          brz etykieta         ;adr, adr+1
          imov GPR1,1          ;adr+2,adr+3
          imov GPR1,2          ;adr+4,adr+5
          imov GPR1,3          ;adr+6,adr+7
etykieta: ...                  ;adr+8
          ...
```

Skok *brz etykieta* powinien zależeć od efektu wykonania rozkazu *idec*, tj. od tego, czy po dekrementacji rejestr *GPR1* zawiera zero. Ponieważ wynik tej operacji (w postaci dekrementacji rejestru oraz ustawienia lub nie znacznika *zero*) będzie znany dopiero po kilku taktach (zakończenie fazy zapisu rozkazu *idec*), na ten czas potoki jednostek przetwarzających powinny wstrzymać pobieranie kolejnych rozkazów.

Takie działanie układu oznacza niewykorzystanie potoków we wszystkich jednostkach przetwarzających. Co więcej, potoki nie będą optymalnie wykorzystane także w następnych taktach. Jest tak dlatego, że po wykonaniu fazy zapisu rozkazu *idec*, gdy można już podjąć decyzję o właściwym kierunku skoku, kolejny rozkaz musi przejść wszystkie stopnie potoku, by zostać wykonany. Tak więc przez kilka - kilkanaście (w jednostkach superpotokowych) taktów potok nie jest w pełni wykorzystywany.

Aby wyeliminować opisane zjawisko, mikroprocesor zachowuje się następująco (rys. 6): w takcie 0 trzy jednostki równolegle pobierają rozkazy z kolejki rozkazów. Jednostka IU pobiera rozkaz *iinc GPR2*, jednostka FPU pobiera rozkaz *flog FPR2*, a jednostka przewidywania adresu skoku rozkaz *brz adr+8*. Adres docelowy skoku jest większy niż adres własny rozkazu skoku, stąd w następnym takcie procesor zakłada, iż skok nie zostanie wykonany i usuwa go z kolejki. Jednocześnie układ zapamiętuje przewidziany przez siebie kierunek skoku oraz alternatywny adres docelowy (adres docelowy

nie wykonanego skoku). Kolejka porządkuje rozkazy (takt 1), a w następnych taktach rozpoczyna wykonywanie rozkazów *imov GPRx, x* w jednostce IU (takty 2,3,4). W momencie gdy rozkaz *idec* zostanie wykonany (takt 5), BPU sprawdza trafność predykcji. Jeśli predykcja powiodła się (zapamiętany kierunek skoku jest właściwy), przetwarzanie odbywa się dalej bez opóźnień. Jeśli nie, BPU sygnalizuje jednostce stałoprzecinkowej konieczność opróżnienia potoku oraz informuje kolejkę rozkazu o konieczności wczytywania zestawu rozkazów spod zapamiętanego adresu docelowego.

Metoda dynamiczna przewidywania wyniku skoku (np. P6 [7], Alpha [5]) polega na użyciu specjalnego bufora zawierającego „historię" skoków. Każdy rekord składa się z trzech elementów: adresu rozkazu skoku, adresu docelowego rozkazu skoku oraz bitów rejestrujących przebieg procesu. Procesor przewiduje kierunek skoku i jeśli mu się powiodło, ustawia bity w odpowiednim rekordzie bufora. W ten sposób procesor analizuje „historię" pętli i często udaje mu się właściwie przewidzieć kierunek skoku (ang. *branch folding*).

Jeśli wynik przewidywania okazał się nieprawdziwy, spowalniane jest tempo przetwarzania, ponieważ następuje konieczność wprowadzenia kilku dodatkowych taktów na sprowadzenie właściwych rozkazów z pamięci. Jeśli jednak przewidywanie było słuszne, to rozkazy przekazywane są do jednostek wykonawczych bez żadnego opóźnienia.

Część procesorów jako tablicę historii przetwarzania wykorzystuje pamięć podręczną rozkazów (Alpha, P6, PA-RISC). Procesor P6 potrafi śledzić grupę skoków warunkowych wykorzystując do tego wielostopniowe algorytmy adaptacyjne. Zaawansowane procesory potrafią przewidywać kierunki skoków kilkustopniowo - czyli ułożyć przewidywany ciąg rozkazów na więcej niż jeden skok naprzód. Szczegóły tych rozwiązań nie są niestety opublikowane.

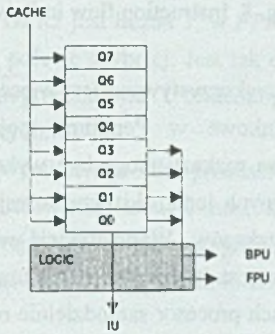
Układ przewidywania kierunku skoku często zawiera też mechanizm, który można opisać jako pamięć podręczną adresów powrotów z procedur. Procesor wykonując skok do procedury musi odłożyć na stos adres powrotny. Wewnątrz układu BPU może znajdować się zestaw rejestrów, który jest jak gdyby odpowiednikiem stosu w pamięci. W ten sposób rozkaz powrotu z podprogramu nie musi oznaczać już konieczności komunikacji procesora z pamięcią.

2.6. Kolejka rozkazów

Jednym z ważniejszych elementów procesora RISC jest *kolejka rozkazów* oraz stowarzyszony z nią *układ pobierania rozkazów* wraz z otoczeniem. Kolejka rozkazów to zestaw kilku bardzo szybkich rejestrów, przechowujących kolejne rozkazy, jakie wykonać ma procesor. Z kolejką rozkazów związany jest skomplikowany układ logiczny, którego zadaniem jest pobieranie rozkazów z pamięci podręcznej oraz szybkie predekodowanie rozkazów - rozpoznawanie, dla jakich jednostek wykonawczych są one przeznaczone. Inną

funkcją układu kolejki jest rozpoznawanie, które elementy kolejki są faktycznie rozkazami a które argumentami im towarzyszącymi. Kolejka rozkazów może przesuwać się o dowolną liczbę pozycji w ciągu jednego taktu.

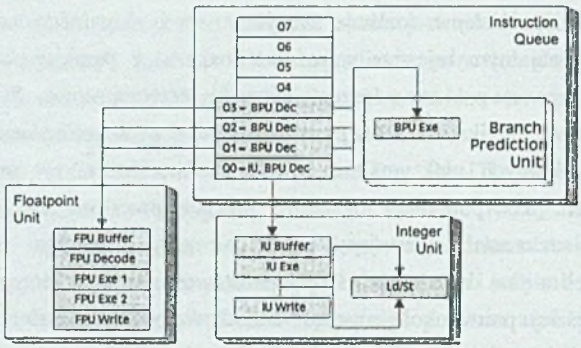
Kolejka zawiera kolejne rozkazy, wczytane spod aktualnego adresu (pamiętanego najczęściej w specjalnym rejestrze wskaźnika rozkazu) z pamięci podręcznej rozkazów. Z kolejki rozkazy są pobierane przez jednostki przetwarzające. Na rys. 7 znajduje się ośmiorejestrowa kolejka rozkazów procesora PowerPC. W czasie każdego cyklu z kolejki może zostać pobranych maksymalnie tyle rozkazów, ile jest w procesorze jednostek przetwarzających. Po pobraniu rozkazów kolejka przesuwa się, a wolne miejsca są uzupełniane rozkazami „wlewającymi się" od góry z pamięci podręcznej. Kolejka najczęściej dzieli się na dwie części. Część podstawowa to ta, z której jednostki pobierają rozkazy (najczęściej połowa kolejki), część dodatkowa pełni rolę bufora. Procesor wypełnia tę część kolejki dopiero wtedy, gdy więcej niż połowa kolejki jest pusta.



Rys. 7. Kolejka rozkazów procesora PowerPC
Fig. 7. Instruction queue in PowerPC

W przypadku procesora PowerPC rozkazy wchodzą do kolejki przez rejestr Q_7 i są „filtrowane" w dół aż do pozycji Q_0. Magistrala łącząca pamięć podręczną z kolejką rozkazów jest szeroka na tyle, że potrafi wypełnić całą kolejkę w czasie jednego taktu (najczęściej istnieje zgodność wielkości sektora pamięci podręcznej i wielkości kolejki). Każda jednostka przetwarzająca potrafi „wyciągnąć" rozkaz tylko z określonego miejsca. Jednostki BPU i FPU potrafią wyciągnąć rozkazy z rejestrów Q_0, Q_1, Q_2, Q_3 kolejki, natomiast IU wyłącznie z Q_0. Kolejka rozkazów procesora jest zintegrowana z blokiem predykcji BPU - rejestry Q_0, Q_1, Q_2, Q_3 są jednocześnie fragmentem kolejki oraz stopniem dekodującym jednostki BPU (rys. 8). Dzięki takiej konstrukcji BPU może obsługiwać rozkazy skoków w czasie praktycznie jednego cyklu. Struktura ta pozwala także na integrowanie układów logiki przetwarzania skoków i kolejki rozkazów. Czasem zdarza

się, iż stopnie zerowy i pierwszy stają się stopniami wstępnego dekodowania rozkazów także dla innych jednostek przetwarzających.



Rys. 8. Przepływ rozkazów w procesorze PowerPC
Fig. 8. Instruction flow in PowerPC

W procesorach, w których wykorzystywana jest więcej niż jedna jednostka danego typu (np. dwa potoki stałoprzecinkowe w Pentium), pojawiają się problemy związane z podejmowaniem decyzji, które rozkazy mogą być wykonywane równolegle, a które nie. Dlatego wśród układów logicznych jednostki przetwarzającej znajduje się specjalny układ śledzący wzajemną zależność rozkazów. Warto zwrócić uwagę na fakt, iż procesor z kilkoma jednostkami tego samego typu jest jak gdyby miniaturą komputera o architekturze typu MIMD [8]. W wielu przypadkach procesor samodzielnie rozstrzyga, które z rozkazów mogą być przetwarzane równolegle, a które należy wykonać sekwencyjnie.

Przy hipotetycznej architekturze procesora RISC z dwiema jednostkami stałoprzecinkowymi, w następującym przykładzie:

start: *idec GPR2*
iadd GPR1, GPR2
imove GPR3, 4

okaże się, że nie warto kierować dwóch pierwszych rozkazów do dwóch pracujących równolegle jednostek przetwarzających IU. Druga jednostka i tak musiałaby czekać na pierwszą, ponieważ rozkaz *iadd* wykorzystuje wynik rozkazu *idec*. Procesor musi znać bowiem wartość rejestru *GPR2* w trakcie fazy *wykonywania* rozkazu *iadd*.. Lepiej więc wprowadzić te dwa rozkazy w kolejnych taktach do jednej jednostki i wykorzystać opisany mechanizm stałego zasilania potoku (*feed forwarding*). Równolegle z wykonywaniem rozkazu *idec* w drugiej jednostce stałoprzecinkowej można natomiast wykonać rozkaz *imove*, ponieważ nie jest on bezpośrednio zależny od poprzednich rozkazów. Taki mechanizm

niekolejnego wykonywania rozkazów (ang. *out of order dispatch*) zaimplementowany został w wielu nowoczesnych procesorach RISC.

Rozważania dotyczące szybkości wykonywania programu dotyczą także przypadku współpracy jednostek przetwarzających różnych typów (np. IU i FPU). W tym jednak przypadku szybkość działania procesora RISC jest silnie zależna od jakości kompilacji. Gdy porównać dwa fragmenty programów:

prog1:	*imove GPR1, 1*	*prog2:*	*imove GPR1, 1*
	imove GPR2, 2		*fmove FPR1, 1.113*
	iadd GPR2, GPR1		*imove GPR2, 2*
	fmove FPR1, 1.113		*fmove FPR2,2.323*
	fmove FPR2, 2.323		*iadd GPR2, GPR1*
	fmul FPR1, FPR2		*fmul FPR1, FPR2*

to pomimo że liczba rozkazów oraz rezultaty wykonania obu są identyczne (po zakończeniu działania programów w rejestrze *GPR2* jest liczba 3, w *FPR1* liczba 2.585499), to program *prog2* wykonywany jest blisko o połowę szybciej. Jest tak dlatego, że w każdym kolejnym takcie równolegle jednostki stałoprzecinkowa i zmiennoprzecinkowa pobierają kolejne rozkazy do potoków przetwarzających.

Proces optymalizacji kodu wynikowego dla procesorów superskalarnych następuje w ostatniej fazie kompilacji (generowanie kodu) i nosi nazwę *szeregowania rozkazów*. Algorytmy szeregujące wykorzystują reprezentację programu w postaci acyklicznego grafu skierowanego DAG (ang. *directed, acyclic graph*). Problem szeregowania jest problemem NP zupełnym.

2.7. Odczyt i zapis do pamięci - jednostka LSU

Jednostka LSU (ang. *Load/Store Unit*) jest wyspecjalizowanym, autonomicznym układem zarządzający dostępem do pamięci z poziomu programu użytkownika. Posiada on wewnętrzne rejestry i bufory pozwalające na arbitraż i optymalizację dostępów do pamięci realizowanych przez jednostki przetwarzające. Układ LSU jest z nimi połączony poprzez wewnętrzną magistralę danych. Układ połączony jest również z pamięcią podręczną danych.

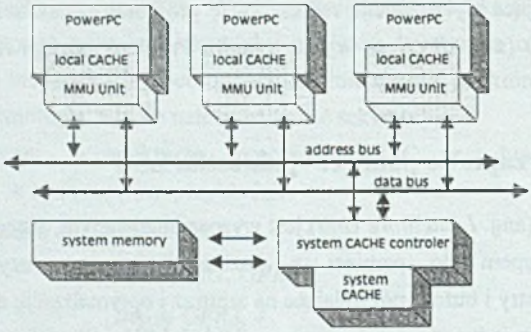
Klasyczny procesor RISC realizuje jedynie dwa rozkazy komunikacji z pamięcią: rozkazy *load* i *store* (najczęściej osobne dla danych stało- i zmiennoprzecinkowych). Rozkazy te mogą mieć modyfikatory, które służą do obsługi danych różnego typu (bajt, słowo). W procesorze RISC obowiązuje zasada zmniejszania liczby trybów adresacji.

2.8. Pamięć podręczna procesora

W procesorach RISC stosuje się rozbudowane konfiguracje pamięci podręcznej (ang. *cache memory*). W najprostszych modelach procesorów pamięć podręczna jest wspólna dla rozkazów i dla danych; w bardziej zaawansowanych konstrukcjach stosuje się pamięci rozdzielone. Pamięć podręczna może odwzorowywać pamięć fizyczną zarówno w trybach *jednoczesnego zapisu* (ang. *write-through mode*), jak i *opóźnionego zapisu* (ang. *write-back mode*) [2].

W wielu procesorach rewolucyjna jest jednak nie sama konstrukcja pamięci podręcznej, lecz układ jej synchronizacji w systemach wieloprocesorowych. W wielu procesorach RISC wbudowany jest specjalny mechanizm utrzymywania spójności pamięci podręcznej procesora z pamięcią główną komputera i pamięciami podręcznymi pozostałych procesorów. Stosowanie takich mechanizmów upraszcza konstrukcję komputerów równoległych o architekturze MIMD.

Na rys. 9 przedstawiono układ kilku pracujących równolegle procesorów PowerPC. Jeszcze kilka lat temu konstrukcje wieloprocesorowe były bardzo drogie, głównie z powodu konieczności stosowania wyspecjalizowanych układów arbitrażowych, rozstrzygających konflikty przy dostępie do wspólnej pamięci.



Rys. 9. Układy PowerPC w komputerze SMP
Fig. 9. PowerPC microprocessors in SMP computer

Aby wszystkie układy PowerPC mogły korzystać ze wspólnej, dzielonej pamięci systemu, konieczne jest przełączenie procesorów w tryb automatycznego utrzymywania spójności pamięci podręcznych. W trybie tym procesor „śledzi" magistralę adresową, sprawdzając, czy dana zapisywana lub odczytywana z pamięci przez inny procesor (lub np. układ DMA) nie jest przechowywana w jego lokalnej pamięci podręcznej.

Jeśli którykolwiek z procesorów zmodyfikuje wspólną pamięć systemu, nastąpi unieważnienie określonego sektora pamięci podręcznej we wszystkich układach, odwzorowujących zmieniony fragment we własną, lokalną pamięć podręczną.

W celu utrzymania spójności pamięci podręcznych wprowadzono *protokół MESI*. Każdy z sektorów pamięci podręcznej procesora PowerPC może znajdować się w jednym z czterech stanów MESI:

stan *MESI*	opis stanu
zmodyfikowany (ang. *modified*)	Sektor pamięci podręcznej jest poprawny, ale tylko w danym procesorze. Dane z tego sektora nie zostały jeszcze przepisane do pamięci systemowej.
wyłączny (ang. *exclusive*)	Sektor jest wyłącznie w pamięci podręcznej danego procesora. Sektor jest spójny z pamięcią główną systemu.
współdzielony (ang. *shared*)	Sektor jest spójny z pamięcią główną systemu. Co najmniej dwa procesory przechowują dany sektor we własnych pamięciach podręcznych.
błędny (ang. *invalid*)	Sektor przechowywany w pamięci podręcznej procesora jest niespójny z pamięcią główną systemu. Należy odczytać właściwe dane bezpośrednio z pamięci głównej systemu.

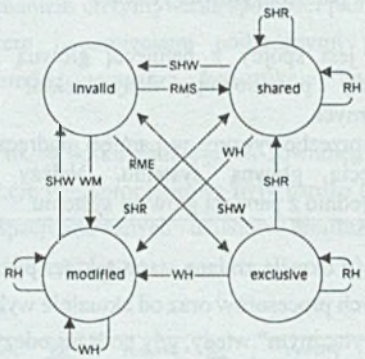
Diagram MESI (rys. 10) określa zmianę stanu sektora pamięci podręcznej w zależności od pamięci podręcznej innych procesorów oraz od aktualnie wykonywanej operacji.

Sektor jest w stanie „wyłącznym" wtedy, gdy podczas odczytu danych z pamięci głównej do pamięci podręcznej żaden inny procesor nie poinformuje, że przechowuje odczytywaną daną we własnej pamięci podręcznej. Wykonanie rozkazu *load* odczytującego przechowywaną daną spowoduje „trafienie" (RH ang. *cache read hit*), a sektor pamięci podręcznej pozostanie w stanie „wyłącznym". Rozkaz *store* zmieniający przechowywaną wartość (WH ang. *cache write hit*) zmieni stan sektora na „zmodyfikowany". Sektor automatycznie zmieni stan na „współdzielony", gdy inny procesor prześle informację, że wczytał ten sam sektor do swojej pamięci podręcznej (SHR ang. *snoop hit on read*). Jeśli inne urządzenie lub inny procesor odwoła się bezpośrednio do pamięci systemowej i zmodyfikuje daną, przechowywaną również w pamięci podręcznej procesora, stan sektora zmieni się na „błędny" (SHW ang. *snoop hit on write*).

Sektor znajduje się w stanie „współdzielony", gdy przynajmniej dwa procesory odwzorowują ten sam obszar pamięci systemu we własne pamięci podręczne. Sektor pozostanie w stanie „współdzielony", jeśli nastąpi trafienie przy próbie odczytu (RH) lub kolejny procesor poinformuje o załadowaniu tego samego sektora do własnej pamięci podręcznej (SHR). Jeśli któryś z procesorów zmieni pamięć główną (SHW), stan sektora zmieni się na „błędny". Jeżeli procesor wykona rozkaz *store* zmieniający daną przechowywaną we własnej pamięci podręcznej (WH), dany sektor zmieni stan

na „zmodyfikowany". Procesor poinformuje przy tym pozostałe układy o konieczności unieważnienia sektorów odwzorowujących zmieniony fragment w lokalne pamięci podręczne.

Sektor jest „zmodyfikowany", jeśli procesor zmienił zawartość pamięci podręcznej, lecz zmiana ta nie została jeszcze zapisana do pamięci głównej systemu (tryb opóźnionego zapisu). Trafienie przy próbie odczytu (RH) lub zapisu (WH) nie zmienia stanu sektora. Jeśli kolejny procesor zgłosi żądanie odczytu danych do własnej pamięci podręcznej (SHR), nastąpi natychmiastowy zapis danych do pamięci głównej systemu i zmiana stanu sektora na „współdzielony". W przypadku nadejścia informacji o zapisie danych od innego procesora (SHW) sektor przejdzie do stanu "błędny".



Rys. 10. Diagram stanów MESI
Fig. 10. MESI diagram

Ze stanu „błędny" sektor może przejść do stanu „wyłączny", jeśli przy próbie odczytu żaden procesor nie poinformuje o przechowywaniu odczytywanych danych we własnej pamięci podręcznej (RME ang. *read miss exclusive*). Sektor zmieni stan na „współdzielony", jeśli podczas odczytu takie zgłoszenie nastąpiło (RMS ang. *read miss shared*). Sektor może przejść do stanu „zmodyfikowany", jeśli nastąpiło chybienie przy próbie zapisu do pamięci (WM ang. *write miss*)

Opisana komunikacja pomiędzy procesorami PowerPC następuje w sposób całkowicie automatyczny, bez udziału programisty. Metoda oparta na synchronizacji pamięci podręcznych, wsparta odpowiednim, wielowątkowym systemem operacyjnym, umożliwia praktycznie nieograniczone skalowanie komputerów o architekturze SMP.

3. Podsumowanie

W ciągu kilku najbliższych lat technologia RISC zdominuje prawdopodobnie typowe konstrukcje procesorów. W chwili obecnej trend ten jest już wyraźnie odczuwalny. Ciekawym zjawiskiem jest próba implementacji technologii RISC w architekturze popularnych procesorów serii Intel 80x86. W procesorze P6, który jest programowo zgodny z linią Intel 80x86, zastosowano mechanizm tłumaczenia złożonych rozkazów CISC na zestawy prostszych rozkazów RISC. Skuteczność takiego rozwiązania jest trudna w ocenie, ponieważ w chwili obecnej brak jest obiektywnych testów tego procesora.

Do najciekawszych rozwiązań wykorzystywanych w procesorze P6 (Pentium Pro) można zaliczyć dynamiczną zmianę nazw rejestrów w czasie działania programu. Jest to próba obejścia ograniczenia polegającego na dostępie do jedynie ośmiu 32-bitowych rejestrów w strukturze 80x86, poprzez odwzorowanie tych rejestrów w dostępnych faktycznie 40 rejestrów 32-bitowych. Wprowadzenie takiego rozwiązania implikuje jednak konieczność stosowania specjalnych buforów przestawień i śledzenia, które z rejestrów przechowują faktyczne wartości odwzorowywanych rejestrów.

Mimo iż stosowanie popularnych procesorów RISC w komputerach SIMD i MIMD wydaje się nieskomplikowane technicznie (większość niezbędnych mechanizmów zawiera najczęściej sam procesor), rodzi się wiele nowych pytań, szczególnie zaś w kwestii optymalizacji kodu dla jednostek superskalarnych.

W chwili obecnej do najszybciej rozwijających się należą niewątpliwie mechanizmy przewidywania kierunku skoku. W przyszłości można się spodziewać optymalizacji przetwarzania skoków w celu dostosowania procesora do obsługi odwołań polimorficznych w językach obiektowych.

LITERATURA

[1] Węgrzyn S.: Podstawy informatyki, PWN, Warszawa 1982.

[2] PowerPC 601 RISC Microprocessor User's Manual, Motorola 1993.

[3] Małysiak H., Pochopień B., Wróbel E.: Mikrokomputer klasy IBM PC. WNT, Warszawa 1992.

[4] Goczyński R., Tuszyński M.: Mikroprocesory 80286, 80386 i i486. HELP, Warszawa 1991.

[5] Alpha 21066/21066A Microprocessors Data Sheet, DEC, 1994.

[6] Cyrix 6x86 Architecture Overview, Cyrix Product Information, Cyrix 1995.

[7] Stam N.: P6 od wewnątrz, PC Magazine po polsku, 1995, 11/33, s.63-70.

[8] Programowanie współbieżne: wybrane zagadnienia, praca zbiorowa pod redakcją Z.Czecha, Skrypt Pol. Śl. nr 1638, Gliwice 1991.

[9] Potter T., Vadden M., Young J.: Resolution of Data and Control-flow Dependencies in the Power PC 601, IEEE Micro 1994/14, s. 18-29.

[10] PC Rynek, PC Kurier 1995/18.

Recenzent: Dr inż. Zdzisław Szczerbiński

Wpłynęło do Redakcji 4 marca 1996 r.

Abstract

This paper discusses some technical solutions built in modern RISC processor. Fig. 1 presents architecture of an RISC microprocessor. Microprocessor is superscalar when several working parallel units get instruction from the instruction queue (Fig.7) and processes them in the same time. Unit can process instruction in pipeline (Fig. 3) what means that each instruction is divided into phases (steps) which are processed by independent hardware parts. Modern microprocessor uses a special feed forwarding mechanism (Fig.4) to reduce time needed to dispatch two dependent instructions (dependent in sense of operating on the same data). One of the biggest problem is the problem of resolving branches. In the pipelined and superpipelined units conditional branches can cause a long delay in processing instructions. Modern RISC processor uses a branch prediction technique to optimize jumps. Fig. 6 shows the static prediction of conditional jump. Processor forces only those jumps which change control (current instruction pointer) to the address previous then the actual. Fig. 9 shows possibility of connecting PowerPC processors in SMP computer.