A. Charakterystyka architektury x86/Pentium

A.1. Ewolucja systemu Intel x86/Pentium

We początkowym okresie rozwoju mikroprocesorów głównym ograniczeniem przepustowości była szybkość procesora. Procesor 8086 był synchronizowany zegarem o częstotliwości $f_{clk} = 8$ MHz ($T_{clk} = 125$ ns) a podstawowy cykl pamięci typu 0WS (*zero wait state*) trwał $4T_{clk} = 500$ ns. W wersji uproszczonej 8088 było $f_{clk} = 4{,}77$ MHz a cykl pamięci trwał $4T_{clk} = 838$ ns. Wszystkie ówczesne pamięci półprzewodnikowe miały czasy dostępu krótsze od 500 ns, więc najwolniejszym elementem był mikroprocesor.

Skrócenie cyklu procesora jest możliwe nie tylko przez zwiększenie częstotliwości taktowania, ale również przez zmianę organizacji przetwarzania. Ponieważ wykonanie każdego rozkazu wymaga pobrania kodu, więc realizacja tego etapu w tle innych akcji doprowadzi do faktycznego skrócenia cyklu rozkazowego. Praktyczna realizacja tego wniosku wymaga rozdzielenia jednostki wykonawczej (*execution unit*) od jednostki sprzęgu z magistralą (*bus interface unit*) oraz wyposażenia procesora w bufor kolejki rozkazów, realizujący antycypowane pobieranie kodów rozkazów z pamięci głównej i nieprzerwane zasilanie jednostki wykonawczej nowymi rozkazami. Skuteczność takiego rozwiązania wynika z *lokalności odwołań* w obszarze programu. Rozmiar bufora jest dostosowany do rozmiaru rozkazów (1 – 15 bajtów) i szerokości magistrali danych. Działanie bufora zakłóca wykonanie skoku, bo wymusza ono przeładowanie kolejki.

Rozdzielenie jednostki wykonawczej od jednostki sprzęgu z magistralą umożliwia też zapis wyniku w czasie wykonywania kolejnego rozkazu, lecz nie wystarcza do przyśpieszenia wolnych odczytów z pamięci. Dalszy rozwój tej koncepcji prowadzi do przetwarzania potokowego, które zaimplementowano w procesorach Pentium

Tablica A.1. Cechy funkcjonalne procesorów x86/Pentium

procesor	8086/88	80286/386	80486	Pentium	P- Pro	Pentium 3	Pentium 4
magistrala danych	16/8b	16/32b	32b	64b	64b	64b	64b
magistrala adresów	16b	24/32b	32b	32b	32b	32–36b	32–36b
długość rozkazu	1 – 6B	1 – 15B	1 – 15B	1 – 15B	1 – 15B	1 – 15B	1 – 15B
bufor kolejki	6/4B	16B	32B	2×64B	2×64B		12k μop
przetwarzanie	EU|BU	*wirtualne*	potok	2 potoki	*data flow*	μoperacje	μoperacje

Zwiększanie f_{clk} (80286 – 12/16 MHz, 80386 – 20…40 MHz, 80486DX – 33…66 MHz) wskutek rozwoju technologii i skracanie cyklu procesora spowodowało bardzo szybko

ujawnienie *bariery przepustowości pamięci* (*memory bottleneck*). Jedną z przyczyn jest wydłużenie czasu dostępu do pamięci wskutek wzrostu ich pojemności i nieuchronne opóźnienia w układach pośredniczących w realizacji dostępu. Wymusiło to zarówno opracowanie nowych technologii wytwarzania pamięci półprzewodnikowych jak też upowszechnienie buforów danych i metod blokowego dostępu do pamięci.

Skuteczność buforowania pamięci głównej przez szybkie układy przechowujące kopie fragmentów pamięci głównej procesora wynika z lokalności odwołań do danych, czyli obserwowalnej tendencji do skupiania danych w niewielkim spójnym obszarze pamięci (lokalność przestrzenna) oraz ponawianych podczas wykonania programu zwrotów do tych samych danych (lokalność czasowa).

Pierwszy sterownik zewnętrznego bufora *cache* opracowano dla procesorów 80386. Użycie zewnętrznego bufora w niewielkim stopniu zmniejsza straty czasu na dostęp procesora do pamięci, bo zwrot do bufora następuje w cyklu dostępu do pamięci.

Istotne skrócenie czasu dostępu można uzyskać, jeśli przynajmniej część transferów nie będzie wymagać użycia magistrali. Jest to możliwe przez użycie wewnętrznej pamięci podręcznej. Rozwiązanie takie, w postaci bufora czterodrożnego o pojemności 8kB, zastosowano po raz pierwszy w procesorze 80486. Ponieważ procesor, tak jak jego poprzednik, współpracował też z buforem zewnętrznym, rozwiązanie to było bardzo efektywne. Dalsze skrócenie czasu dostępu do pamięci daje umieszczenie w strukturze procesora bufora dwupoziomowego, po raz pierwszy zastosowano w procesorach Pentium II (bufor L2 o pojemności 256/512 kB). Ostatnim etapem ewolucji bufora *cache* jest dwu- lub trzypoziomowy bufor wewnętrzny w procesorach Pentium 4.

W buforze pamięci pierwszego poziomu może wystąpić konflikt jednoczesnych żądań dostępu wysyłanych z układu wykonawczego (*execution unit*) i bloku wstępnego pobierania rozkazów (*prefetch unit*), który powoduje przestóje potoku (*pipeline stall*). Rozwiązaniem stosowanym we wszystkich procesorach Pentium jest rozdzielenie wewnętrznego bufora *cache* na niezależne bloki: pamięć programu i pamięć danych.

Radykalne skrócenie czasu dostępu do pamięci przez użycie pamięci podręcznych ujawnia kolejną barierę szybkości przetwarzania, którą jest sekwencyjne kierowanie rozkazów do potoku przetwarzania. Ponieważ niektóre rozkazy mogą być wykonane współbieżne, więc użycie potoków równoległych umożliwi szybsze przetwarzanie. Taki potok superskalarny (*superscalar pipeline*) nie wyklucza jednak przestojów podczas wykonania czasochłonnych operacji. Rozwiązaniem jest dopuszczenie wykonania rozkazów w kolejności innej niż kolejność ich napływu do potoku.

Związane z wykonywaniem rozkazów skoku i rozgałęzień przeładowania kolejki rozkazów i wstrzymywanie potoku można zminimalizować przez prognozowanie skoków (*branch prediction*). Użycie dwóch buforów kolejki rozkazów umożliwia wyeliminowanie zwłoki związanej z przeładowaniem kolejki w razie nietrafnej prognozy, szczególnie uciążliwe w architekturze Pentium z uwagi na zmienny rozmiar kodów rozkazów. Rozsądny podział cyklu rozkazowego na krótsze etapy prowadzi do zwiększenia wydajności potoku.

A.2. Opis architektury Pentium

Pierwsze procesory Pentium, oznaczane P54, były zasilane napięciem 5V. W wersji P54C zawierały wbudowany układ współpracy ze sterownikiem przerwań APIC, były przystosowane do wieloprocesorowości, zasilane napięciem 3,3V i współpracowały z magistralą z częstotliwością 2/3 f_{clk}. Pentium OverDrive to współprocesory dokładane do systemów zbudowanych w oparciu o procesory serii 80486DX2/DX4 lub Pentium. Procesory Celeron są zubożoną wersją procesora Pentium II/III, bez wbudowanej pamięci podręcznej L2. W Pentium MMX dołożono jednostkę stałoprzecinkową SIMD dla działań współbieżnych (praktycznie grafika 2D). Pentium III wykonuje rozkazy zmiennoprzecinkowe wspomagające współbieżne przetwarzanie sygnałów (SSE).

Tablica A.2 Charakterystyka procesorów o architekturze IA-32

procesor	80486	Pentium / MMX	Pentium Pro/ II/ III	Pentium 4
adres fizyczny	32b	32b	32b / 36b	32b / 36b
magistrala danych	32b	64b+8b (p)	64b+8b (p)	64b+8b (p)
jednostka stałoprzecinkowa	potokowa	superskalarna dwupotokowa, (F-D1-D2-E-W)	2 x superpotokowa, F1-F2-D1-D11-D2-D12 -E1-E2-E3-E4-E5-W *data flow*	3 x superpotokowa, *data flow*
pamięć podręczna L1	C+D – 8/16kB, czterodrożna, linia 16B	C – 8kB – 2w, D – 8kB – 4w, linia 32B	C – 8kB – 2w, D – 8kB – 4w, linia 32B	C – 8kB – 2w, D – 8kB – 4w, linia 32B
L2	–	–	256/512 kB	256/512 kB
kolejka rozkazów	pojedyncza 32B	podwojona 2×64B	podwojona 2×64B (PIII) mikrooperacje	12k µop
prognoza skoków	–	d	d+s	d+s
nowe instrukcje	BSWAP, XADD CMPXCHG8B	CPUID, CMOV, ...,RDTSC, {MMX}	{SSE}, {SSE2}	{SSE3}

W procesorach Pentium 4 zastosowano nowe rozwiązania sprzętowe – architekturę NetBurst dla transferów wewnętrznych, umożliwiającą szybkie przesyłania danych pomiędzy buforami, wielopoziomowy bufor *cache*, stos sprzętowy oraz wewnętrzną kolejkę mikrooperacji. Procesory te zawierają powiązaną ze środowiskiem (pamięcią i peryferiami) powłokę CISC, dostarczającą dane i przetwarzającą strumień rozkazów na mikrooperacje oraz jądro RISC, w którym mikrooperacje są przetwarzane w trybie *data flow*. Listę rozkazów SSE rozszerzono na argumenty podwójnej precyzji (SSE2).

W roku 2004 pojawiło się 64-bitowe rozszerzenie architektury IA-32. Procesory o architekturze IA-32e dysponują 8 dodatkowymi 64-bitowymi rejestrami jednostki stałoprzecinkowej i mają mozliwość adresowania przestrzeni 2^{64}B.

Szczegółowe informacje o wersji procesora i jego możliwościach wykonawczych można uzyskać za pomocą instrukcji CPUID.

A.2.1. Struktura funkcjonalna procesorów Pentium

Zasadniczymi elementami struktury funkcjonalnej procesorów Pentium są:
- układ sprzęgu z magistralą (*bus unit*)
- pamięć podręczna kodu (*code cache*) wraz z układem stronicowania
- układ uprzedzającego pobierania kodu (*prefetcher*) i prognozy skoków
- pamięć podręczna danych (*data cache*) wraz z układem stronicowania
- układ rozsyłania rozkazów
- jednostka wykonawcza.

Poszczególne bloki funkcjonalne w różnym stopniu podlegały ewolucji. Największe zmiany nastąpiły w strukturze jednostki wykonawczej, która najpierw zawierała dwa współbieżne potoki stałoprzecinkowe (*pipelined integer units*) i modułową jednostkę zmiennoprzecinkową (*floating-point unit*) a następnie uległa przekształceniu w stację rezerwacyjną zawierającą wiele różnych jednostek funkcjonalnych.

Układ sprzęgu z magistralą zewnętrzną zawiera:
- sterowniki magistrali adresowej i odbiorniki stanu (*address drivers and receivers*);
- dwukierunkowe bufory magistrali danych (*data bus transceivers*)
- logikę sterowania (*bus control logic*) wytwarzającą kody identyfikujące cykl
- sterowanie dostępem do magistrali (*bus master control*)
- sterowanie zewnętrzną pamięcią podręczną (*cache control*)
- sterowanie wewnętrzną pamięcią podręczną (*internal cache control*)
- bufory zapisu (*write buffers*)
- generatory i kontrolery parzystości (*parity generation and control*).

Logika sterowania magistrali określa rodzaj transferu (standardowy lub blokowy) i wytwarza sygnały sterujące. Cykle standardowe występują podczas komunikacji z wejściem i wyjściem lub niekopiowalnymi (*non-cacheable*) lokacjami pamięci głównej oraz podczas zapisu do linii nieskopiowanych w pamięci podręcznej. Podczas tych transferów rozmiar danych (8/16/32b) odpowiada rozmiarowi argumentu instrukcji. Cykle blokowe (*burst*) są wykonywane przy wypełnianiu linii pamięci podręcznej oraz podczas zapisu zwrotnego (*write-back*) danych.

Adres słowa 64-bitowego jest wyprowadzany na liniach A31:A3, linie BE7:BE0 wskazują aktywne bajty (co pozwala odtworzyć sygnały A2:A0 potrzebne w cyklach standardowych). Pamięć główna ma często strukturę dwóch banków przechowujących słowa parzyste (BE3:BE0) i nieparzyste (BE7:BE4). Linie A31:A5 są dwukierunkowe, aby możliwe było odczytywanie stanu magistrali w cyklach podglądania (*cache snoop*), niezbędne dla zapewnienia spójności wewnętrznej pamięci podręcznej.

Podczas zapisu są wytwarzane bity parzystości dla każdego bajtu danych i adresu. Parzystość linii danych jest testowana przy odczycie a adresowych przy podglądaniu.

Wypełnienie linii bufora *cache* wymaga jednego przesłania blokowego (*burst*), które obejmuje 4 przesłania 8-bajtowe (64-bitowe) i nie wymaga użycia linii adresowych A4:A0, lecz tylko 27 linii A31:A5. lub 31 linii A35:A5 (przy rozszerzonej magistrali).

A.2.2. Struktura potoku Pentium i Pentium MMX (P5)

Linia kodu pobrana z podręcznej pamięci kodu jest kopiowana do aktywnej kolejki rozkazów (*prefetch queue*). Pobranie linii (32 bajty) do aktywnego spośród 2 buforów następuje zawsze, gdy jest w nim dość wolnego miejsca. Jeśli na czele kolejki zostanie wykryta instrukcja rozgałęzienia, to zostaje uaktywniony układ prognozy dynamicznej i następuje załadowanie kolejki nieaktywnej. Bufor prognozy BTB jest czterodrożną pamięcią podręczną o 256 wejściach (liniach). Każde wejście (*directory entry*) zawiera:

- bit ważności wejścia,
- adres docelowy rozgałęzienia – aby uniknąć każdorazowego obliczania adresu
- adres źródłowy rozkazu rozgałęzienia – w celu identyfikacji rozgałęzienia
- co najmniej dwa bity historii rozgałęzienia, stanowiące podstawę prognozy.

Nie jest możliwa prognoza skoku pośredniego JMP/CALL [BX] i rozkazów RET, bo ich adres docelowy nie jest znany podczas pobrania kodu. Brak też prognozy instrukcji LOOP, bo jej wykonanie zależy od stanu rejestru (E)CX, który może być zmieniona podczas wykonania rozkazu poprzedniego. W efekcie w każdym powtórzeniu pętli zakończonej instrukcją LOOP następuje wstrzymanie potoku na 1 pełny cykl.

Dwie instrukcje z początku kolejki są przesłane jednocześnie do dekoderów D1 obu potoków w celu sprawdzenia możliwości ich współbieżnego wykonania. Instrukcje współbieżne są zsynchronizowane. Jeśli instrukcje muszą być wykonane sekwencyjnie, wówczas obie są kierowane kolejno do potoku U. Współbieżnie z innymi mogą być wykonane tylko tzw. instrukcje proste (arytmetyczne, logiczne, zliczanie, składanie i zdejmowanie ze stosu, kopiowanie, skoki i rozgałęzienia) pod warunkiem, że nie występują między nimi zależności funkcjonalne. Niektóre instrukcje mogą być kojarzone z innymi tylko wtedy, gdy są kierowane do potoku U, inne mogą być kojarzone tylko gdy stanowią drugą instrukcję w parze, wykonywaną w potoku V.

Tablica A.3. Kojarzenie instrukcji w Pentium

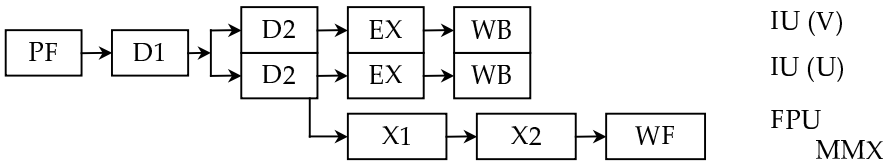
Instrukcja	Opis	Operand	Potok
ADC/SBB	*dodaj z przeniesieniem / odejmij z pożyczką*	rejestr lub pamięć	tylko U
ADD/SUB/CMP	*dodaj / odejmij / porównaj*	rejestr lub pamięć	U lub V
DEC / INC	*zmniejsz / zwiększ o 1*	rejestr lub pamięć	U lub V
AND/OR/XOR	*logiczne and /or / xor*	rejestr lub pamięć	U lub V
Jcc	*skok warunkowy*		tylko V
CALL/JMP NEAR	*skok (do procedury) wewnątrzsegmentowy*	bezpośredni (lub krótki)	tylko V
LEA	*zapisz adres efektywny*	rejestr, pamięć	U lub V
MOV	*kopiuj*	pamięć lub rejestr ogólny	U lub V
NOP	*nic nie wykonuj*		U lub V
POP/PUSH	*zapisz na stos / odczytaj ze stosu*	rejestr/(lub stała)	U lub V
Rxx / Sxx	*rotacje i przesunięcia*	rejestr lub pamięć o *n* pozycji	tylko U
TEST	*porównuj kody*	rejestr oraz rejestr lub pamięć	U lub V

W fazie D2 adres logiczny (segment, przemieszczenie) jest przekształcony na adres liniowy i są sprawdzone prawa dostępu. Generator adresu pełni te same funkcje jak w 80486, ale w każdym trybie adresowania oblicza adres efektywny w jednym cyklu.

Układ stronicowania (*page unit*), podobny jak w 80486, jest załączany ustawieniem bitu PG w rejestrze CR0 i zawiera osobne bufory dla obu wewnętrznych pamięci podręcznych. Każdy bufor antycypacji translacji TLB umożliwia jednoczesną obsługę dwóch adresów liniowych, czyli jednoczesną translację adresu strony dla 2 potoków. Układ umożliwia obsługę stron o rozmiarze 4kB lub 4GB.

Fazę wykonania EX realizują jednostki arytmetyczno-logiczne. W jednostce ALU-U mogą być wykonywane dowolne instrukcje, jednostka ALU-V nie wykonuje rotacji i przesunięć. Wstrzymanie ALU-U blokuje możliwość zapisu wyniku wytworzonego w jednostce V, lecz nie odwrotnie. Wykonanie rozkazów zmiennoprzecinkowych (oprócz FXCH), których etap wykonania jest 3- fazowy (lub dłuższy przy obliczaniu wartości funkcji przestępnych) powoduje wstrzymanie potoku dla innych instrukcji.

Stałoprzecinkowa jednostka arytmetyczno-logiczna ALU jest podwojona, jednak współpracuje z pojedynczym dwuportowym plikiem rejestrowym. ALU potoku U może zakończyć operację przed ALU potoku V. Jednostka ALU potoku V nie zawiera przesuwnika cyklicznego (*barrel shifter*) i układu korekcji dziesiętnej. Obliczenie adresu docelowego skoku względnego może być natomiast wykonane tylko w potoku V.



Rys. 1. Struktura potoku Pentium / Pentium MMX

Jednostka zmiennoprzecinkowa ma 3 bloki: dodawania, dzielenia i mnożenia, co pozwala przyśpieszyć wykonanie złożonych instrukcji zmiennoprzecinkowych. Proste instrukcje zmiennoprzecinkowe, takie jak:

- zmień znak [FABS (*absolute value*), FCHS (*change sign*)]
- dodaj [FADD, FADDP (*add*)]
- odejmij [FSUB, FSUBP (*subtract*), FSUBR, FSUBRP (*reverse subtract*)]
- porównaj [FCOM, FCOMP(P) (*compare real*)]
- porównaj nieuporządkowane [FUCOM, FUCOMP(P) (*unordered compare real*)]
- podziel [FDIV, FDIVP (*divide*), FDIVR, FDIVRP (*reverse divide*)]
- pomnóż [FMUL, FMULP (*multiply*)]
- prześlij na lub ze stosu zmiennoprzecinkowego [FLD (*load real*)]
- testuj [FTST (*test*)],

nie mogą być wykonane współbieżnie, lecz jednocześnie z ich wykonaniem w potoku U, w potoku V może być wykonana instrukcja wymiany FXCH (*floating-point exchange*).

Jednostka MMX używa tych samych rejestrów co jednostka FPU, lecz w swobodnej konfiguracji (oznaczanych wtedy MM# zamiast ST(#). Skutkiem tego po sekwencji rozkazów MMX należy unieważnić stos zmiennoprzecinkowy (rozkazem EMMS).

Ostatnią fazą przetwarzania instrukcji jest zapis wyniku (*write-back*) do rejestru, pamięci podręcznej danych lub bufora zapisu i uaktualnienie znaczników (EFLAGS).

A.2.3. Struktura potoku w architekturze P6 (Pentium Pro, II, III)

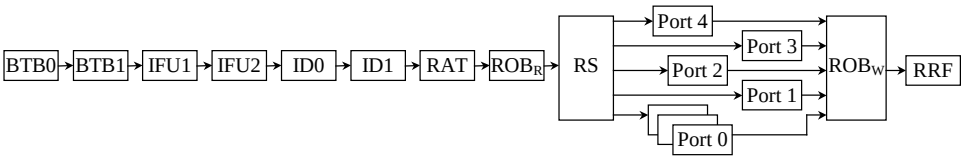
W architekturze P6 (Pentium Pro, II, III) zasadniczym zmianom uległa jednostka wykonawcza procesora. W miejsce dwóch potoków stałoprzecinkowych i jednostki zmiennoprzecinkowej wprowadzono stację rezerwacyjną o 20 wejściach. Dekoder może dekodować co najmniej 3 rozkazy jednocześnie. Strumień przetwarzania w P6 przekształcono w 12-etapowy superskalarny superpotok dynamiczny, z możliwością niekolejnego wykonania mikrooperacji.

Cykl potoku rozpoczyna się od prognozy wykonania (rys. xx), przy tym:

- dynamiczna prognoza skoku wykorzystuje wzorce wykonania (4- lub 5-bitowe), a w razie jej niepowodzenia wykonywana jest prognoza statyczna
- nadal kłopotliwe dla potoku są instrukcje skoku pośredniego oraz instrukcja RET (adres docelowy jest znany dopiero po zakończeniu dekodowania)

W procesorach Pentium Pro i Pentium II użycie instrukcji LOOP blokuje możliwość wykrycia konfliktu danych i wstrzymuje przetwarzanie potokowe, dlatego zaleca się jej zastąpienie przewidywalnymi rozkazami skoków warunkowych J*cc*. W procesorach Pentium III instrukcja LOOP jest dekodowana na mikroinstrukcje proste.

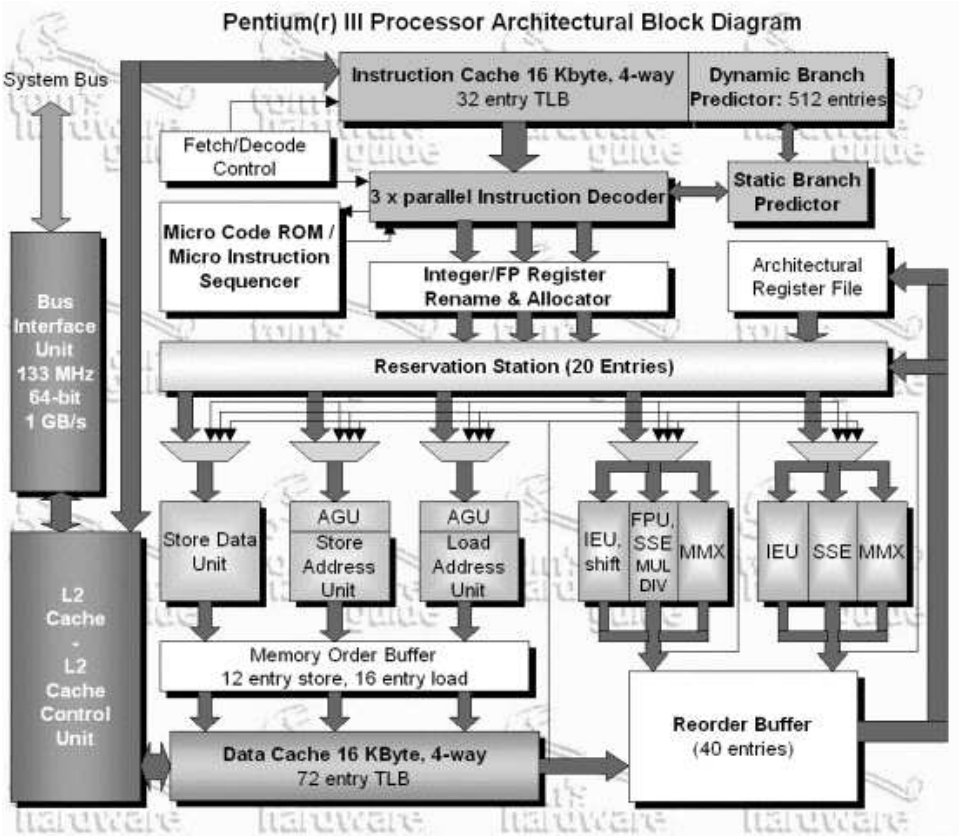
Potok rozkazów z bloku wstępnego pobierania jest dekomponowany na rozkazy proste (load, store, inne działania tylko na rejestrach), niekiedy z użyciem pamięci stałej mikrorozkazów. W trakcie wstępnej analizy może też nastąpić przemianowanie rejestrów. Dekoder (z pewnymi ograniczeniami) może jednocześnie wysłać od 3 do 6 mikrorozkazów do stacji rezerwacyjnej (*reservation station*), w której wykonanie jest synchronizowane napływem danych (tryb „data flow").



Rys. 2. Struktura potoku w architekturze P6 (BTB# – prognoza skoku, IFU# – pobranie rozkazu z kolejki, ID# – dekodowanie, RAT – wytworzenie adresu fizycznego, ROB – bufor przestawiania, RS – bufor stacji rezerwacyjnej, Port# – jednostki wykonawcze, ROBw – bufor przywracania kolejności zapisu, RRF – zapis)

Nowa jednostka SSE używa osobnego zestawu rejestrów zmiennoprzecinkowych i realizuje tylko podstawowe działania arytmetyczne: dodawanie, odejmowanie,

mnożenie, dzielenie, obliczanie pierwiastka kwadratowego, porównanie, obliczanie minimum i maksimum oraz rozkazy umożliwiające komunikację z rejestrami MMX. Wykonanie instrukcji jest równoległe (4 operacje jednocześnie na 4 polach rejestru XMM#) lub skalarne (na najniższym polu z jednoczesnym kopiowaniem 3 wyższych pól pierwszego argumentu). Wyniki są zaokrąglane według wybranej reguły, wyjątki mogą być maskowane.



Rys. 3 Struktura blokowa procesora Pentium III

A.2.4. Przetwarzanie potokowe w Pentium IV

W architekturze P7, podobnie jak w P6, potok rozkazów z bloku wstępnego pobierania jest dekomponowany na rozkazy proste (niekiedy z użyciem pamięci stałej mikrorozkazów). Powstały w wyniku przekodowania strumień mikrooperacji jest kierowany do bufora o pojemności 12k takich operacji. Umożliwia to wykonywanie nawet długich pętli bez konieczności każdorazowego przekodowania rozkazów.

Wewnętrzne transfery odbywają się za pomocą szybkiej i szerokiej wewnętrznej magistrali, przystosowanej do transferów blokowych (*Net Burst Microarchitecture*).

Superpotok superskalarny procesora Pentium 4 ma 20 stanowisk :

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TCNxtIP		TC Fetch		Dr	All	Rename		Que	Sch			Disp		RF		EX	Flg	BrC	Dr

Rys. 4 Cykle potoku Pentium 4

1,2 – *Trace Cache Next IP* – wskaźnik IP z bufora TC
3,4 – *Trace Cache Fetch* – Pobranie instrukcji do niekolejnego wykonania
5 – *Drive* – propagacja sygnału, może nie zdążyć w czasie pojedynczego cyklu
6,7,8 – *Allocation, Rename* – alokacja danych i przemianowanie rejestrów
9 – *Queue* – kolejka, oczekiwanie na wysłanie do portu ekspedycyjnego
10–12 – *Scheduling* – szeregowanie, pozwala przetwarzania w trybie *data driven*

- operacje na pamięci
- szybkie ALU proste działania arytmetyczne i logiczne
- powolne ALU/ podstawowa FPU
- proste działania FP oraz komunikacja FPU z pamięcią

13–14 – *Dispatch* – ekspedycja do portów j.w., maks. 6 mikrooperacji (μop) w cyklu, przepustowość wyższa od przepustowości poprzednich etapów (*front end*) (3 μop na cykl) i przepustowości wykonania (*back end*) (3 μop na cykl)

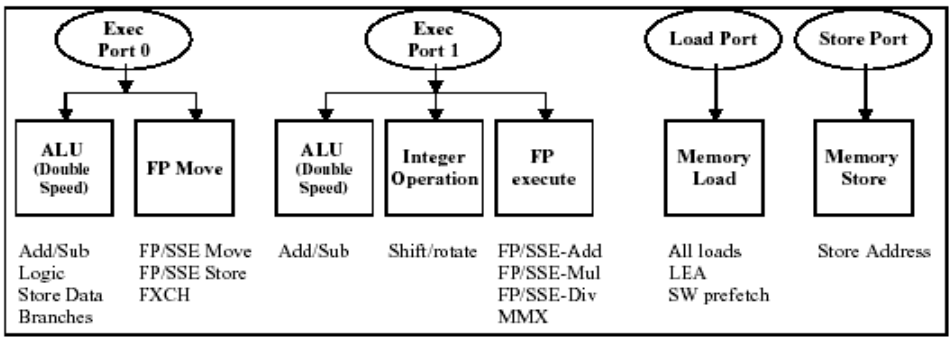
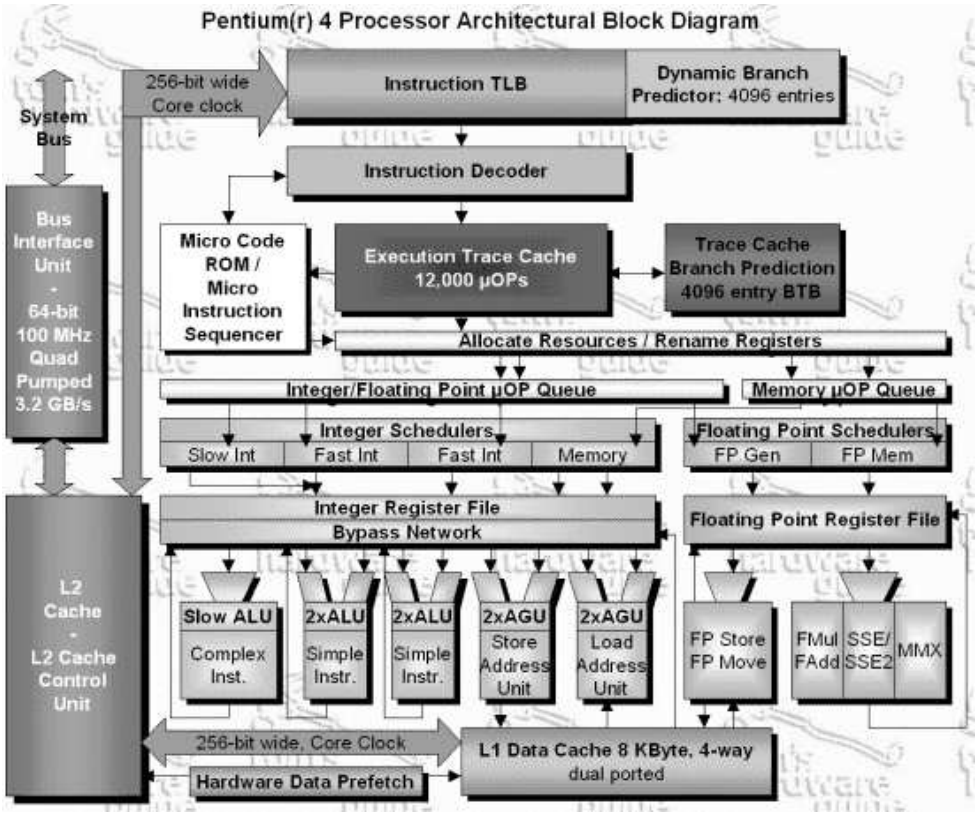


Figure 6: Dispatch ports in the Pentium® 4 processor

Rys. A.

15–16 – *Register File access* – dostęp do plików rejestrów
17 – *Execution* – wykonanie instrukcji
18 – *Flags* – aktualizacja flag (jeśli potrzebna)
19 – *Branch Check* – sprawdzenie trafności prognozy rozgałęzienia
20 – *Drive* – propagacja sygnału, może nie zdążyć w czasie pojedynczego cyklu



Rys. 5 Struktura blokowa procesora Pentium 4

A.3. Adresowanie wirtualne w architekturze IA-32

Podstawową metodą wspomagania pracy wielozadaniowej w architekturze IA-32 (80386/486/P5/P6/P7) jest segmentacja. Rozmiar i struktura segmentów systemowych jest jednolita dla wszystkich procesorów tej grupy, jedynie format segmentu stanu zadania TSS (*Task State Segment*), musi być dostosowany do zmian architektury. Wtórnym mechanizmem jest stronicowanie.

W każdym trybie pracy (rzeczywistym, wirtualnym, systemowym, symulowanym V86) adresowanie dowolnego obiektu w pamięci wymaga użycia dwóch wskaźników:

- selektora segmentu, który podczas adresowania zostaje przekształcony na adres bazowy segmentu, lub (dla tablic GDT i IDT) adresu bazowego segmentu
- adresu względnego (przemieszczenia) wewnątrz segmentu.

Przekształcenie selektora na adres bazowy segmentu jest bezpośrednie w trybie 8086/88 (*real mode*) – 20-bitowy adres bazowy tworzy 16 bitów selektora segmentu

uzupełnionych czterema bitami 0. W trybie wirtualnym adres bazowy segmentu jest wynikiem przekodowania selektora segmentu przez tablicę opisów (LDT lub GDT).

Zgodnie z zasadą dwuwskaźnikowego adresowania logicznego w procesorach rodziny Intel 80x86/Pentium, systemowe struktury danych muszą też być umieszczane w segmentach i adresowane za pomocą 2 wskaźników. Wykonanie każdego zadania w trybie chronionym / wirtualnym (*protected / virtual mode*) wymaga odwołań do:

- segmentu stanu zadania TSS – numer w rejestrze TR
- segmentu lokalnej tablicy opisów LDT – numer w rejestrze LDTR
- segmentu globalnej tablicy opisów – adres bazowy w rejestrze GDTR
- segmentu tablicy wektorów przerwań – adres bazowy w rejestrze IDTR.

Dostęp do każdego segmentu systemowego w celu jego wypełnienia lub odczytu odbywa się przez jego tymczasowe redefiniowanie (z poziomu wywołania systemowego) jako segmentu danych (na podstawie opisu umieszczonego w GDT).

W każdym odwołaniu do lokalnej lub globalnej tablicy opisów lokalizację opisu wewnątrz segmentu LDT lub GDT wskazuje selektor właściwego segmentu programu (CS podczas adresowania kodu, DS, ES, FS lub GS podczas adresowania danych oraz SS podczas adresowania stosu). Lokalizację opisu w segmencie IDT wskazuje wektor przerwania przyjętego do obsługi.

Kontekst procesora, obejmujący także aktualne wskaźniki stosu na wszystkich poziomach uprzywilejowania, jest automatycznie składowany i odtwarzany z użyciem segmentów TSS podczas przełączania zadania. Opis w segmencie TSS jest zapisem stanu systemu w momencie wstrzymania (wywłaszczenia) zadania. Opis początkowy tworzy procedura systemu operacyjnego podczas działania programu ładującego kod zadania i jego dane do pamięci.

Opisy w tablicach GDT i LDT są trzech rodzajów i wskazują:

- segment pamięci (kod, dane)
- furtkę wywołania procedury lub zadania – sposób przekazania sterowania między poziomami ochrony – wskaźnikiem jest zawsze rejestr CS
- systemową strukturę danych – segment TSS lub LDT (przez furtkę zadania).

Opis w tablicy IDT wskazuje zawsze sposób przekazania sterowania między poziomami ochrony – furtkę wywołania procedury obsługi przerwania. Wynika to z wymagania obsługi wszelkich przerwań na wyższych (systemowych) poziomach ochrony (również sprzętowych przerwań wejścia / wyjścia).

W trybie wirtualnym adresowanie dowolnego obiektu następuje zawsze za pomocą opisu (deskryptora), zatem wszystkie adresy w programie są adresami pośrednimi. Rejestr selektora segmentu (lokalnego – CS, DS, ES, FS, GS, SS lub systemowego –TR, LDTR) w dowolnej chwili realizacji zadania wskazuje lokalizację w tablicy opisów lokalnych LDT lub globalnych GDT. Opis najczęściej dotyczy segmentu pamięci. Jego elementami są wtedy: adres bazowy (liniowy) i rozmiar segmentu oraz uprawnienia i sposób dostępu. Opis może też stanowić tzw. furtkę (*gate*) dostępu do innego zadania lub procedury utworzonej na innym poziomie ochrony niż zadanie wywołujące.

Furtka zawiera selektor docelowego deskryptora (bloku TSS lub pamięci), zaś selektor CS, uaktywniany rozkazem skoku międzysegmentowego i wskazujący opis furtki, jest selektorem pośrednim.

Aby przyśpieszyć obliczanie adresu w pamięci (liniowego lub fizycznego przy wyłączonym stronicowaniu) każdy rejestr identyfikatora segmentu ma przysłonięte rozszerzenie 8-bajtowe, niedostępne dla programisty i automatycznie uaktualniane (rodzaj bufora *cache*), w którym umieszczona jest kopia opisu wskazywanego przez identyfikator w tablicy opisów LDT lub GDT. Celowość takiego rozwiązania wynika z zasady lokalności.

Wskazanie obiektu na tym samym poziomie ochrony polega na dodaniu adresu wewnątrz segmentu (obliczonego zgodnie z trybem adresowania) do 32-bitowego adresu bazowego odpowiadającego użytemu selektorowi, a praktycznie pobranego z ukrytego przedłużenia rejestru selektora. Kontrola praw dostępu następuje tylko podczas zmiany segmentu (kodu lub danych). Odwołanie do danych na innym poziomie ochrony powoduje sprawdzenie praw dostępu i jest praktycznie wykluczone w programie użytkownika, realizowanym na najniższym poziomie ochrony.

<table>
<tr><td colspan="7">Adres bazowy segmentu ($A_{31...24}$)</td><td>G</td><td>D/B</td><td>0</td><td>AVL</td><td colspan="2">Rozmiar seg. ($L_{19...16}$)</td></tr>
<tr><td>P</td><td colspan="2">DPL</td><td>S</td><td>t</td><td>t</td><td>t</td><td>t/a</td><td colspan="6">Adres bazowy segmentu ($A_{23...16}$)</td></tr>
<tr><td colspan="14">Adres bazowy segmentu ($A_{15...0}$)</td></tr>
<tr><td colspan="14">Rozmiar segmentu ($L_{15...0}$)</td></tr>
</table>

t t t a – segment zwykły (S = 1)	*t t t t* – segment systemowy (S = 0)
0h/8h – segment danych, RWX = (r – –)	0h, 8h – nieokreślony
1h/9h – segment danych, RWX = (r w –)	2h – tablica LDT
2h/Ah – segment danych (stos), RWX = (r – –)	9h – segment TSS dostępny (IA-32)
3h/Bh – segment danych (stos), RWX = (r w –)	0Bh – segment TSS zajęty (IA-32)
4h/Ch – segment kodu, RWX = (– – x)	9h (1h) – segment TSS dostępny (80286)
5h/Dh – segment kodu, RWX = (r – x)	0Bh (3h) – segment TSS zajęty (80286)
6h/Eh – segment kodu zgodny, RWX = (– – x)	4h...7h – furtka 80286*
7h/Fh – segment kodu zgodny, RWX = (r – x)	0Ch...0Fh – furtka IA-32 (80386+)*

*) Format deskryptora furtki jest pokazany na następnym diagramie

Rys. A.xx Struktura deskryptora pamięci i deskryptora systemowego: P – bit obecności, DPL – poziom uprzywilejowania deskryptora, S – typ deskryptora, G – ziarnistość adresu (1 – rozmiar segmentu $L_{23...0}$ podany w stronach 4 kB, 0 – rozmiar $L_{15...0}$ podany w bajtach), AVL – dostępny, a – bit używalności, D/B – domniemany rozmiar adresu i operandu (1 – 32b, 0 – 16b)

Użycie bajtu przedrostka rozmiaru operandu (66h) lub bajtu przedrostka rozmiaru adresu (67h) lokalnie zmienia rozmiar nadany bitem D deskryptora.

Wywołanie zadania lub procedury na poziomie ochrony innym niż bieżący zawsze następuje za pomocą furtki (przejścia) wskazanej przez selektor CS w tablicy LDT lub GDT podczas wykonania skoku lub wywołania międzysegmentowego. Z treści opisu

furtki pobierany jest nowy selektor (docelowy), wskazujący w przypadku przełączania zadań lokalizację segmentu stanu zadania, a podczas wywołania procedury lokalizację nowego segmentu kodu i adres startowy procedury (przemieszczenie w segmencie). Może też nastąpić obniżenie poziomu ochrony, chyba że docelowy segment kodu jest segmentem zgodnym, którego wykonanie jest możliwe na każdym poziomie ochrony. Podczas przełączenia zadań do opisu TSS nowego zadania jest wpisany identyfikator zadania wywołującego. Wywołanie procedury, zarówno wewnątrzsegmentowej (NEAR), jak i pozasegmentowej (FAR – przez furtkę), powoduje umieszczenie adresu powrotu na stosie zadania wywołującego.

Zakończenie zadania i zwrotne przełączenie do zadania wywołującego (jeśli był ustawiony bit NT w rejestrze stanu procesora) realizuje instrukcja IRET. Powrót z procedury realizuje natomiast instrukcja RET (RETF), a jeśli procedurę realizowano na wyższym poziomie uprzywilejowania następuje też przełączenie stosu.

Przemieszczenie ($d_{31...16}$)										
P	DPL	S	t	1	t	a	0	0	0	Liczba słów przekazywanych
Selektor (wirtualny)										
Przemieszczenie ($d_{15...0}$)										

Przemieszczenie – adres wewnętrzny procedury w segmencie docelowym wskazanym przez selektor wirtualny (nieistotne gdy deskryptor jest furtką zadania), Liczba słów przekazywanych jest istotna tylko w deskryptorze furtki wywołania

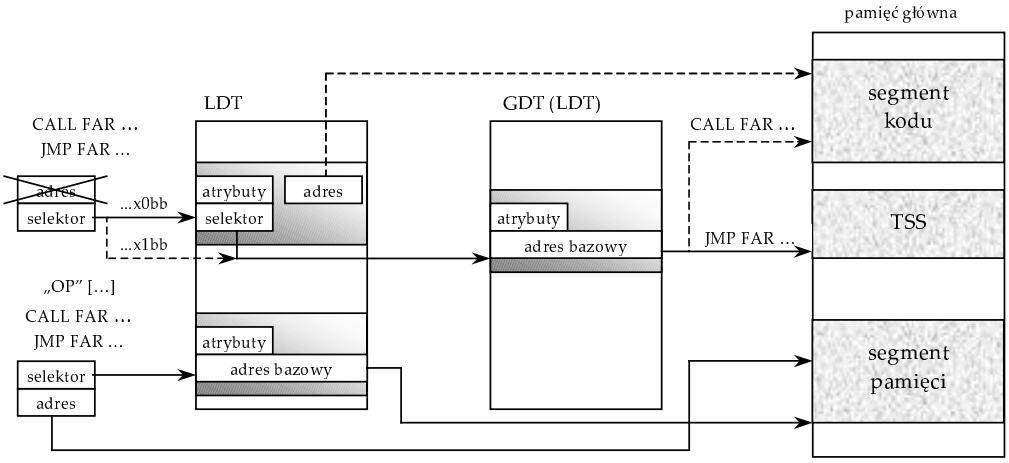
t 1 *t a* – deskryptor furtki (S = 0)	*t* 0 *t a* – segment danych kontekstowych
5h – furtka zadania (*task gate*)	0h, 8h – nieokreślony
0Ch (4h)– furtka wywołania (*cal gate*)	2h – tablica LDT
0Eh (6h) – furtka przerwania (*interrupt gate*)	9h (1h) – segment TSS dostępny
0Fh (7h) – furtka potrzasku (*trap gate*)	0Bh (3h) – segment TSS zajęty

Uwaga: W nawiasach kod dla trybu wirtualnego 80286

Rys. Struktura systemowego deskryptora furtki

W trybie wirtualnym są dwa sposoby dostępu do kodu procesu. Dostęp do kodu własnego (bez zmiany poziomu ochrony) jest wykonany za pośrednictwem zwykłego deskryptora pamięci, wskazanego za pomocą selektora CS (rozkaz JMP lub CALL). Deskryptor zawiera adres bazowy bloku pamięci, w którym kod jest umieszczony.

Dostęp wymagający zmiany poziomu ochrony jest dwuetapowy – najpierw selektor użyty w programie podczas wykonania rozkazu JMP lub CALL wskazuje położenie deskryptora furtki i dopiero potem umieszczony w tym deskryptorze *selektor docelowy* lokalizuje opis wskazujący segment pamięci, w którym umieszczono kod docelowy. Przy takim wywołaniu oryginalne przemieszczenie jest ignorowane, bo albo nie jest potrzebne (furtka zadania), albo jest zakodowane w deskryptorze (furtka procedury).



Rys. 8. Użycie furtki i deskryptora

Poziomy ochrony

Każdy rozkaz programu jest wykonywany na określonym poziomie ochrony, zwanym bieżącym poziomem uprzywilejowania CPL (*Current Privilege Level*). Poziom ten może być zmieniony tylko przy zmianie selektora CS, co jest możliwe tylko przez wykonanie skoku lub wywołania międzysegmentowego. Każdy selektor CS na dwóch najniższych bitach ma zapisany kod swojego poziomu uprawnień, zwany *poziomem uprzywilejowania segmentu żądającego* (*Requestor Privilege Level, RPL*).

Informacja o bieżącym poziomie uprzywilejowania jest porównywana z poziomem uprzywilejowania deskryptora zapisanym w polu DPL (*Descriptor Privilege Level*). Jeśli wskazywany jest segment pamięci, wtedy pole DPL określa minimalny poziom uprzywilejowania, uprawniający do dostępu do tego segmentu. Jeśli wskazywana jest furtka, to pole DPL wskazuje najwyższy dozwolony poziom uprzywilejowania, uprawniający do użycia tej furtki (aby zadania bardziej uprzywilejowane nie mogły korzystać z usług niższego poziomu). W takim przypadku zostają zignorowane bity RPL selektora docelowego, a procedura może być wykonana, jeśli poziom ochrony zapisany w polu DPL deskryptora segmentu kodu tej procedury nie jest niższy niż bieżący poziom uprzywilejowania.

Podczas dostępu do danych należy porównać DPL ze słabszym z dwóch CPL i RPL. Zasady zmiany poziomów ochrony są następujące:

1. Można wykonać procedurę o wyższym poziomie uprzywilejowania – nie można wykonać procedury mniej uprzywilejowanej (na niższym poziomie ochrony). Jeśli zadanie wykonywane na poziomie RPL korzysta z usługi na poziomie CPL to pole *rpl* selektora segmentu danych powinno zostać osłabione – istotny jest słabszy poziom.

2. Można uzyskać dostęp do danych na niższym poziomie ochrony – nie można uzyskać dostępu do danych bardziej uprzywilejowanych.

Instrukcje we i wy mogą być wykonane tylko przez program, którego CPL jest nie niższy od wskazanego w polu IOPL (może być inne dla każdego zadania) lub tablicy zezwoleń we/wy (część opisu w segmencie TSS).

Dostęp do pamięci jest zawsze realizowany

- przez deskryptor pamięci – do danych lub kodu własnego
- przez furtkę (zadanie, podprogram, przerwanie, pułapka) – do kodu innego zadania lub jego udostępnionej części (procedury).

Zmiana trybu pracy

Po załączeniu lub inicjalizacji sygnałem RESET procesor działa zawsze w trybie rzeczywistym, a pierwsza instrukcja (zwykle skok) jest pobierana z lokacji 0FFF0h w ostatnim segmencie standardowym (64kB) w przestrzeni adresowej procesora. *Automatyczna zmiana na tryb rzeczywisty* następuje wskutek zgłoszenia przerwania SMI, powodującego przejście do trybu systemowego SM.

Przejście do trybu wirtualnego następuje przez ustawienie (=1) bitu PE w rejestrze sterującym CR0. Konieczne jest też natychmiastowe opróżnienie kolejki rozkazów, należy zatem wykonać instrukcję skoku wewnątrzsegmentowego. Stan selektorów segmentów zostaje automatycznie zmieniony, więc wskazują one nadal te fizyczne segmenty, które były wskazywane przed przełączeniem. Przed zmianą selektora trzeba zatem zadbać o zainicjowanie tablic i rejestrów systemowych, w tym także IDT, która w trybie wirtualnym zawiera furtki a w trybie rzeczywistym adresy procedur. Jest też konieczne przeładowanie rejestru IDTR. Aby wykluczyć błędną obsługę przerwania, podczas przełączania trybów i ładowania rejestru IDTR konieczne jest zablokowanie przerwań. Należy też dokładnie sprawdzić program aby wykluczyć wystąpienie wyjątku. Podczas wykonania procedury przełączania musi być także przygotowany segment TSS nowego zadania i środowisko dla stronicowania (jeśli będzie używane).

Powrót do trybu rzeczywistego w procesorach od 80386 począwszy, może nastąpić przez wyzerowanie bitu PE w rejestrze sterującym CR0. Przedtem należy wyłączyć stronicowanie, przekazać sterowanie do segmentu standardowego (64kB), załadować do rejestrów segmentów selektory wskazujące segmenty 64kB o adresach bazowych odpowiadających wynikowi translacji selektora na adres w trybie rzeczywistym oraz zablokować przerwania. Po wyzerowaniu bitu PE należy wykonać instrukcję JMP i instrukcję LIDT w celu wskazania położenia tablicy wektorów przerwań dla trybu rzeczywistego.

A.3.1. Stronicowanie

W architekturze IA-32 stronicowanie jest wtórnym mechanizmem adresowania wirtualnego. Segmentacja może być wyłączona tylko pozornie przez ustalenie dla

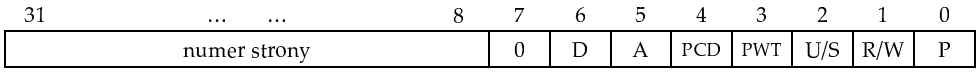
każdego segmentu zerowego adresu bazowego i maksymalnego rozmiaru. Metoda ta jest określana jako płaski model pamięci (*flat memory*).

Stronicowanie zostaje włączone po ustawieniu bitu 31 (PG) w rejestrze sterującym CR0. Rozmiar strony może być standardowy (4kB) lub rozszerzony (4GB). Dodatkowo ustawienie bitu rozszerzenia adresu fizycznego w rejestrze CR4 (b_5=PAE – *Physical Address Extension*) umożliwia użycie stron o rozmiarze 2MB.

W podstawowym trybie 10 najwyższych bitów adresu liniowego wskazuje katalog tablic stron a 10 kolejnych aktualną tablicę stron. W trybie rozszerzonym najwyższych 10 bitów adresu liniowego wskazuje tablicę stron a pozostałe adres na stronie. Każdy deskryptor strony jest 32-bitowy i zawiera 20- lub 10-bitowy numer strony fizycznej.

W opcji z rozszerzonym 36-bitowym adresem fizycznym 2 najwyższe bity służą do wyboru jednego z 4 katalogów. W trybie podstawowym kolejne grupy po 9 bitów wskazują katalog stron i aktywną tablicę, a każdy deskryptor umieszczony w tablicy zawiera 24-bitowy numer strony standandardowej, a w trybie rozszerzonym 9 bitów wskazuje katalog stron o rozmiarze 2GB każda. Rozmiar deskryptora jest rozszerzony do 64 bitów, aby umożliwić wytwarzanie 36-bitowego adresu bazowego strony.

Lokalizacja katalogu stron jest zapisana w rejestrze sterującym CR3. Deskryptor tablicy stron w katalogu oraz strony w tablicy mają taką samą strukturę. Najniższe 8 bitów deskryptora zawiera informacje o sposobie dostępu do strony (rys A.), wyższe 20-24 bitów są numerem (adresem) strony fizycznej (rys.). Dla strony można określić poziom uprzywilejowania (U/S=1 – poziom użytkownika, odpowiadający najniższemu poziomowi uprzywilejowania w segmentacji, lub systemowy – wszystkie pozostałe)



Rys. Struktura deskryptora strony

W obszarze strony można lokalnie wyłączyć (PCD=1) użycie wewnętrznej pamięci podręcznej oraz ustalić tryb współpracy z zewnętrzną pamięcią podręczną (PWT). Ustawiony bit D wskazuje, że w obszarze strony dokonano zapisu, P jest bitem obecności strony w pamięci głównej.

Rys. Stronicowanie w trybie rozszerzonych adresów fizycznych

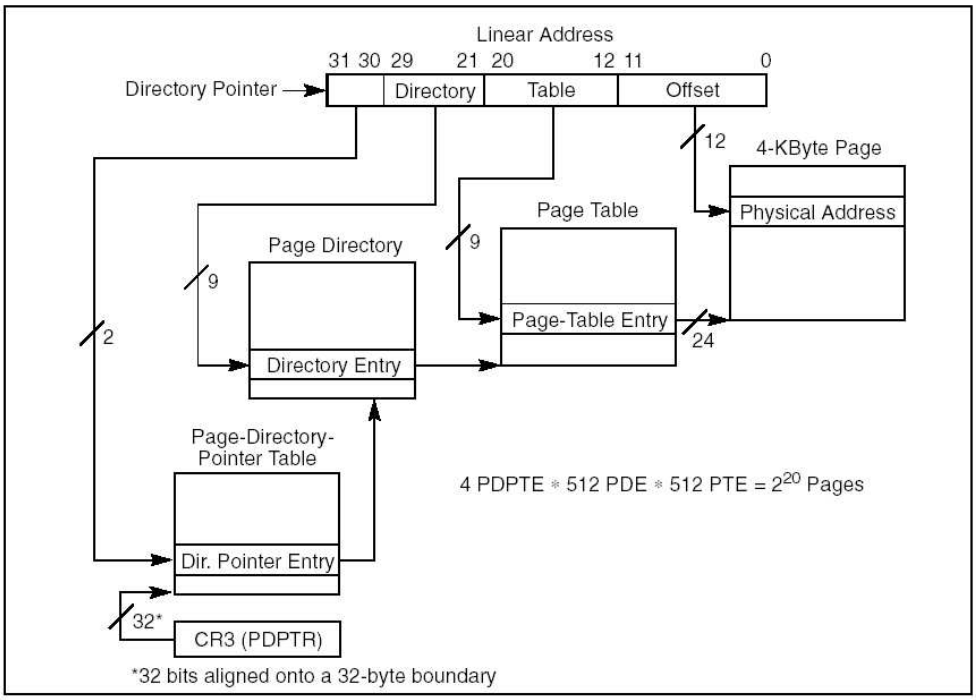


Figure 3-18. Linear Address Translation With PAE Enabled (4-KByte Pages)

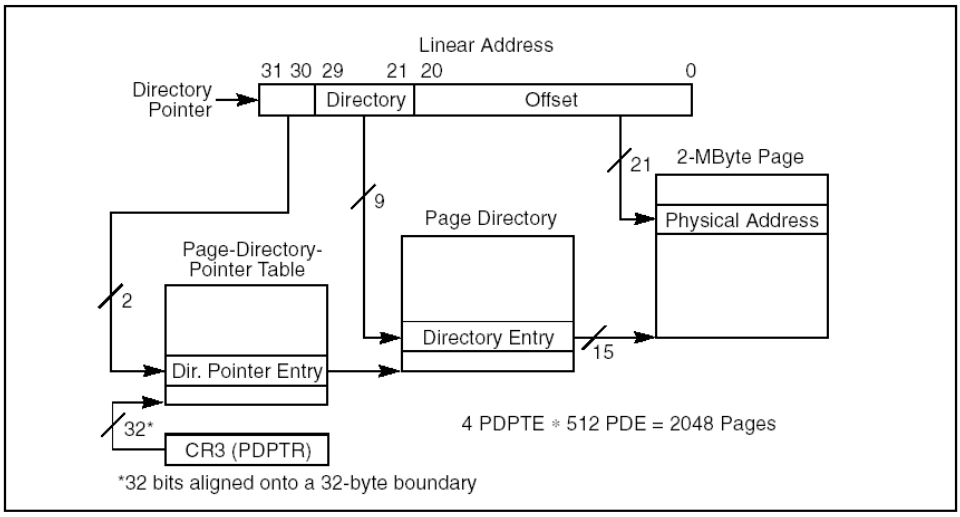


Figure 3-19. Linear Address Translation With PAE Enabled (2-MByte Pages)

A.4. System przerwań

System przerwań procesora Pentium obejmuje zestaw przerwań systemowych najwyższego priorytetu (RESET, BUSCHK#, R/S#, FLUSH#, SMI#, INIT), z których każde ma osobne wejście zgłoszenia, oraz przerwania standardowe, przeznaczone głównie do obsługi urządzeń peryferyjnych. Domniemanym sposobem identyfikacji tych przerwań jest wektoryzacja. Rozstrzyganie priorytetów i wysyłanie wektora jest funkcją sterownika przerwań.

A.4.1. Przerwania systemowe i tryb systemowy SMM

Realizacja wielu funkcji zarządzania systemem powinna przebiegać niezależnie od trybu pracy procesora. Szczególnym rodzajem takich funkcji jest zarządzanie poborem mocy, a zwłaszcza czasowe odłączanie i ponowne załączanie nieaktywnych urządzeń.

Wspomaganie mechanizmów zarządzania na poziomie architektury procesora, w sposób przeźroczysty dla systemu operacyjnego i zadań użytkowników, upraszcza projektowanie systemu. Cechę przeźroczystości można uzyskać przez umożliwienie realizacji procedur w przestrzeni adresowej, niedostępnej (przysłoniętej) podczas normalnej pracy, uaktywnianej zgłoszeniem specjalnego przerwania. W procesorach Pentium służy temu tryb systemowy SMM (*system management mode*), uruchamiany przerwaniem (impulsem) na wejściu SMI# (powrót rozkazem RSM).

Typowe funkcje realizowane przez procedury obsługi (*handlers*) SMM to:

- przechowanie stanu urządzenia
- wyłączenie urządzenia nieaktywnego (*idle*)
- uaktywnienie urządzenia wyłączonego
- zatrzymanie lub zwolnienie oscylatora i zegarów
- przechowanie stanu systemu w pamięci nieulotnej (*non-volatile*) (np. CMOS RAM) i wyłączenie zasilania
- obsługa zagrożeń bezpieczeństwa (*security protection*).

Żądanie obsługi zdarzenia z kategorii zarządzania systemem (w tym zarządzania poborem mocy) jest zgłaszane przerwaniem sprzętowym (zbocze opadające) na wejściu SMI# (*system management interrupt*). Wyższy priorytet mają tylko sygnały testu magistrali (BUSCHK#), opróżnienia pamięci podręcznej (FLUSH#) oraz przejście do stanu testowania procesora (R/S#). Przyjęcie zgłoszenia SMI powoduje zainicjowanie następującego ciągu akcji:

- wysłanie potwierdzenia na wyjście SMIACT# (*SMI accept*)
- przełączenie pamięci – załączenie przysłoniętej pamięci systemu zarządzania SMRAM i zablokowanie (przysłonięcie) dostępu do pamięci standardowej RAM w obszarze pokrywającym SMRAM
- przechowanie kontekstu (stanu) procesora w SMRAM

- inicjalizacja rejestrów do stanu wymaganego przez SMM (CS:= 3000h – adres bazowy 00030000h, inne selektory segmentów i ich adresy bazowe = 0, zakres segmentów 2^{32}, (pamięć „płaska"), EIP:= 00008000h, EFLAGS = 00000002h – (bit 1 zawsze jest „1")
- rejestr stanu procesora CR0 – PE=0, EM=0, TS=0, TRAP=0, IE=0, PG=0
- identyfikacja źródła przerwania (przeglądnięcie rejestrów stanu urządzeń)
- obsługa (przechowanie stanu urządzenia i jego wyłączenie)
- powrót z obsługi SMI – rozkaz RSM, dostępny tylko w trybie SMM
- odtworzenie stanu procesora, przełączenie SMRAM na RAM i powrót do zadania przerwanego.

Funkcje SMM są realizowane zawsze w trybie rzeczywistym. Segment pamięci SMRAM ma rozmiar 64KB i jest domyślnie umieszczony od adresu bazowego 30000h. Domniemany względny adres początkowy procedury określa zainicjowany IP (00008000h). Kontekst procesora jest umieszczany w ostatnich 512 bajtach segmentu (od adresu względnego 0FE00h). W obszarze kontekstu może być też przechowany adres we/wy, które zostało wyłączone.

Domniemane położenie segmentu SMRAM może być w ograniczonym zakresie zmienione przy wejściu do tego obszaru. Segment SMRAM musi być umieszczony w obszarze adresowym trybu rzeczywistego i na granicy bloku 32kB (11 mniej znaczących bitów selektora CS musi być zerami). Powinno być to zrealizowane podczas wykonaniu procedur inicjalizujących POST (*power-on self-test*) przez zapis nowego adresu bazowego pamięci (segmentu) SMRAM w obszarze kontekstu SMM (adres względny od 7E00h do 7FFFh). Jeśli tryb SMM jest używany, to procedura POST musi mieć dostęp do przysłoniętej SMRAM w celu wpisania tam procedur obsługi SMM. Procedury SMM mają natomiast dostęp do całego pozostałego obszaru RAM (oprócz segmentu przysłoniętego przez SMRAM). Umożliwia to umieszczenie pamięci SMRAM w nieużywanym obszarze RAM.

Pojedyncze zgłoszenie SMI, które nie może być natychmiast obsłużone, a także ponowne zgłoszenie SMI podczas realizacji procedury SMM jest zapamiętywane. W takim przypadku zostaje zignorowana instrukcja RSM i następuje podjęcie kolejnej obsługi SMI#. Kolejne zgłoszenie SMI przed podjęciem obsługi zgłoszenia odłożonego jest ignorowane.

Jeśli obszar SMRAM jest nałożony na obszar RAM, który może być odwzorowany w pamięci podręcznej, to przed przyjęciem SMI należy opróżnić pamięć podręczną danych i kodu oraz bufory zapisu. Można to łatwo wykonać wymuszając w chwili zgłoszenia SMI# sygnał przerwania FLUSH#, którego priorytet jest wyższy. Ponieważ EM=0, ignorowane są przerwania jednostki zmiennoprzecinkowej (FERR#, IGNNE#).

Błąd krytyczny w trybie SMM (adres bazowy segmentu niepodzielny przez 2^{15}, nielegalny stan CR0, CR4: $b_{7:31}$=1 powoduje wyłączenie komputera (*shutdown*).

Priorytet przerwania SMI# jest wyższy niż priorytet zgłoszeń NMI oraz INTR a także wyższy niż priorytet przerwania inicjującego INIT (częściowy RESET, podczas

którego nie zostają unieważnione pamięci podręczne ani bufory zapisu, nie zmienia się stan rejestrów stosu zmiennoprzecinkowego ani rejestrów specyficznych, nie ulega też zmianie ustawienie bitów CD (*cache disable*) i NW (*not write-through*) w rejestrze CR4).

Wyższy priorytet niż SMI# mają w kolejności ważności:

- RESET – pełny restart, ogólne wznowienie
- BUSCHK# – błąd magistrali (*bus check error*)
- R/S# – (*run/stop#*) przełączenie procesora w tryb pracy z kontrolą wszystkich wyprowadzeń układu procesora, wykorzystywany podczas testowania
- FLUSH# – wymuszenie zapisu zwrotnego z *cache* i buforów zapisu a następnie unieważnienie wszystkich linii pamięci podręcznej.

Stan wejścia BUSCHK jest testowany podczas aktywności sygnału potwierdzenia gotowości transmisji BRDY#. W razie zgłoszenia stan magistrali jest zapamiętany w rejestrze adresu MCAR (*Machine Check Address Register*) i rejestrze statusu magistrali MCTR (*Machine Check Type Register*). Stan rejestrów tych rejestrów może być odczytany za pomocą rozkazu RDMSR

Jako reakcja na BUSCHK# jest generowany wyjątek, ale tylko wtedy gdy w rejestrze sterującym CR4 jest ustawiony bit MCE (*Machine Check Exception*). Wyjątek MCE jest generowany w razie wystąpienia błędu parzystości adresu, błędu parzystości podczas zapisu, a także w razie wystąpienia błędu parzystości podczas odczytu danych przy aktywnym stanie PEN# (*Parity Enable*).

A.4.2. Przerwania standardowe

Procesor Pentium w wersji podstawowej, oprócz zgłoszeń przerwań najwyższego priorytetu (RESET, BUSCHK#, R/S#, FLUSH#, SMI#, INIT) ma 2 standardowe wejścia przerywające: NMI i INTR. Przerwanie zgłoszone poziomem na wejściu INTR jest przyjmowane na granicy kolejnej instrukcji i wymusza potwierdzenie przyjęcia przerwania na magistrali procesora (dwa cykle *back-to-back* – „ramię w ramię"), o ile procesor jest przygotowany do odbioru zgłoszenia (ustawiony bit IF w rejestrze flag). Wszystkie przerwania oprócz krytycznych są precyzyjne – podjęcie ich obsługi wymaga zakończenia wszystkich instrukcji w potoku.

Do pracy w takim trybie jest przewidziany programowalny sterownik przerwań Intel 8259A (*Programmable Interrupt Controller*, PIC), umożliwiający zgłoszenie ośmiu, zaś w trybie kaskadowym 8x8=64 zgłoszeń przerwań wektorowych. Sterownik umożliwia też dynamiczną rotację priorytetów zgłoszeń, maskowanie zgłoszeń i różne tryby zagnieżdżania zgłoszeń.

Sterownik 8259A jest przeznaczony do współpracy tylko z jednym procesorem, także w asymetrycznych systemach wieloprocesorowych, w których jeden procesor wykonuje system operacyjny. Sterownik nie może być wykorzystany w symetrycznych

systemach wieloprocesorowych. Ograniczone możliwości redystrybucji zgłoszeń przerwań ma sterownik 82489DX, stosowany w systemach z procesorem 80486.

Dla systemów wieloprocesorowych budowanych w oparciu o procesory Pentium dostępne są wersje procesora zawierające rozbudowany układ obsługi zgłoszeń przerwań, przystosowany nie tylko do obsługi standardowych zgłoszeń przerwań na wejściu INT (ze sterownikiem 8259A) ale także do współpracy z rozbudowanym sterownikiem przerwań (*Advanced Programmable Interrupt Controller*, APIC), podobnym funkcjonalnie do 82489DX. Lokalny blok APIC procesora zostaje wyłączony (tryb *bypass*), jeśli podczas zerowania procesora (RESET) na linii PICD0/APICEN# jest stan wysoki (APICEN = 0). W takim stanie procesor obsługuje przerwania w trybie 8086.

Przyjmowanie zgłoszeń przerwań przy aktywnym lokalnym sterowniku APIC wymaga użycia zewnętrznego koncentratora zgłoszeń APIC I/O, przyłączonego do magistrali przerwań PIC. Układ APIC I/O ma 16 wejść zgłoszeń przerwań, z których najwyższy priorytet sprzętowy ma wejście INTIN0 a najniższy INTIN15. Koncentrator I/O APIC testuje zgłoszenia na liniach INTIN# i przekazuje zgłoszenie o najwyższym priorytecie do lokalnego APIC, identyfikowanego adresem logicznym lub fizycznym.

W trybie zwykłym lokalny APIC każdego procesora może przyjmować 2 zgłoszenia lokalne LINT0 (INTR) i LINT1 (NMI) od urządzeń przyłączonych do lokalnej magistrali procesora. Odpowiadające im wektory przerwań są określone podczas programowania lokalnego APIC. Podsystem APIC obsługuje 4 kategorie przerwań:

- systemowe przekazywane przez APIC I/O
- NMI przekazywane przez APIC I/O
- lokalne (LINT0/1, Timer, Error)
- międzyprocesorowe IPI (*inter-processor interrupt*), w tym: STARTUP, INIT, SMI.

Wyróżnia się tylko 2 typy transferu: standardowy oraz EOI (zakończenie obsługi przerwania). Transfer EOI jest nadrzędny i w razie zgłoszenia transakcji EOI agent realizujący transfer standardowy musi zwolnić magistralę PIC.

Każdy transfer następuje zgodnie z protokołem na 3-liniowej magistrali przerwań PICD0, PICD1, PICCLK ($f_{PICCLK} < 16{,}67$ MHz). Protokół definiuje rodzaj przesyłanej informacji i zawiera identyfikator procesora (lokalnego APIC) lub modułu APIC I/O i wektor przerwania. W każdym cyklu przesyłane są dwa kolejne bity wiadomości. Protokół transferu na magistrali przerwań składa się z 14 (EOI), 21 lub 39 cykli zegara (linia PICCLK). W razie jednoczesnego rozpoczęcia transferów następuje arbitraż. Każdy transfer rozpoczyna się od 5 cykli – identyfikacji typu i arbitrażu:

- cykl 1 – określa typ transferu (11 – EOI, *end of interupt*)
- cykl 2-5 – arbitraż źródeł transferu (PICD0 = 0:0:0:0, PICD1 – D3:D2:D1:D0)

Każdy układ APIC (lokalny, I/O) ma unikatowy identyfikator 4-bitowy, nadawany na zboczu opadającym sygnału RESET jako stan wejść BE3#:BE0# (w systemach wieloprocesorowych, stan BE0# jest ignorowany a najniższy bit jest automatycznie przypisany każdemu z pary procesorów). Podczas pierwszego transferu identyfikator ten, jako identyfikator arbitrażu, jest przesyłany inwersyjnie na linie PICD0 i PICD1. Są

one wykonane jako „*open-drain*", czemu odpowiada realizacja iloczynu logicznego przesłanych jednocześnie sygnałów (*dotted-AND*, „iloczyn na drucie"). Każdy układ sprawdza w kolejnych cyklach 2-5 zgodność wynikowego bitu arbitrażu z kolejnym bitem swojego identyfikatora i w razie stwierdzenia niezgodności przerywa transfer. Po każdym arbitrażu identyfikator zwycięzcy jest zerowany a identyfikatory arbitrażu pozostałych układów są zwiększane o 1. Zapewnia to rotację priorytetów zgłoszeń.

Zakończenie każdego transferu jest również standardowe i obejmuje 5 cykli

- cykl *n*–4 – przesłanie sumy kontrolnej dla cykli od 6 do (*n*–5)
- cykl *n*–3– zerowy (szczelina czasowa dla obliczenia sumy kontrolnej)
- cykl *n*–2 – status sumy kontrolnej (00 – OK)
- cykl *n*–1 – status transferu (udany, błędny, powtórzenie)
- cykl *n* – 00 – koniec transferu.

Każdemu wejściu INTIN jest przypisany 8-bitowy wektor przerwania, który zostaje umieszczony w rejestrze skierowań (*redirection register*). Z każdym rejestrem są także skojarzone *atrybuty* przerwania:

- tryb zgłoszenia (*trigger mode*) – zbocze / poziom
- tryb identyfikacji celu (*destination mode*) – sposób adresowania lokalnego APIC
- identyfikator celu (*destination field*) – wskazanie lokalnego (lokalnych) APIC
- tryb doręczenia (*delivery mode*).

W fizycznym trybie identyfikacji wysyłany jest fizyczny identyfikator lokalnego APIC (procesora), dla którego przeznaczona jest wiadomość albo kod 1111 oznaczający wszystkie procesory i wtedy doręczanie następuje w trybie najniższego priorytetu. W trybie logicznym używany jest (zapisany w rejestrze LDR) 8-bitowy identyfikator logiczny, zawierający tylko jedną jedynkę, interpretowany zależnie od modelu identyfikacji (zapisanego w rejestrze formatu doręczania DFR). W modelu *płaskim* (*flat*) adresatami wiadomości są procesory wskazane przez jedynki w adresie logicznym (maksymalnie 8 procesorów). W modelu *klastrowym* (*cluster*) *płaskim* tworzone są grupy maksimum 4 procesorów, których 4 wyższe bity identyfikatora są jednakowe, zaś 4 niższe traktowane tak jak w modelu płaskim. Model *klastrowy hierarchiczny* pozwala na utworzenie sieci *klastrów* i wymaga dodatkowego sprzętu i oprogramowania.

Tryb doręczenia określa sposób przyjmowania przerwania. Wyróżnia się:

- tryb statyczny – zgłoszenie akceptowane przez fizycznie zidentyfikowany APIC
- tryb najniższego priorytetu (*lowest priority*) – zgłoszenie jest identyfikowane przez wszystkie lokalne układy logicznie zaadresowane
- tryb zewnętrzny – zgłoszenie pochodzi od sterownika 8259A, a wskazany fizycznie lokalny APIC wymusza 2 cykle INTA (*back-to-back*) w celu odczytu wektora przerwań z magistrali danych.

Protokół normalnego zgłoszenia przerwania zawiera 21 cykli w tym:

- cykl 6-7 – tryb identyfikacji (DM = 0 – fizyczny, 1 – logiczny) i doręczania (M2:M0 = 000 (statyczny), 001 (najniższy priorytet), 111 (zewnętrzny)

- cykl 8 – (w cyklu IPL INIT) definicja poziomu i sposobu zgłoszenia (zbocze – 0, poziom – 1), niezbędna dla wysłania właściwego sygnału zakończenia EOI
- cykl 9-12 – wektor przerwania
- cykl 13-16 – identyfikator celu (fizyczny tylko 4-bitowy, logiczny 8-bitowy)

Protokół zakończenia obsługi przerwania (EOI) zawiera 14 cykli – w cyklach 6-9 jest wysłany wektor przerwania.

Protokół przesłania zawartości 32-bitowego rejestru sterującego lokalnego APIC do innych APIC obejmuje 39 cykli (wraz z nagłówkiem i zakończeniem).

Rejestry lokalnego APIC są implementowane w postaci słów w pamięci o adresach z zakresu 0FEE000xxh. Każdy bit 256-bitowego rejestru odpowiada pozycji w tablicy przerwań. Zgodnie z zastrzeżeniem producenta, pierwsze 16 lokacji zarezerwowano dla obsługi wyjątków lub przerwań wewnętrznych.

W układzie sterownika APIC, podobnie jak w 8259A, szczególną rolę pełnią:

- rejestr zgłoszeń IRR (*interrupt request register*) – bit ustawiany w chwili akceptacji zgłoszenia
- rejestr obsług ISR (*in-service register*) – bit ustawiany w wewnętrznym cyklu INTA (do ISR jest kopiowany bit o najniższym numerze z IRR; zerowanie ISR następuje w cyklu EOI), jeśli zgłoszenie zostało przyjęte do obsługi
- rejestr trybu zgłoszenia TMR (*trigger mode register*) – ustawiany wraz z IRR aby wskazać rodzaj akcji lokalnego APIC w cyklu EOI.

Cztery najbardziej znaczące bity wektora przerwań określają priorytet zgłoszenia, zatem każda grupa 16 wektorów ma przypisany ten sam priorytet (0Fh najwyższy, 01h - najniższy). Z każdym poziomem priorytetu jest skojarzony dwupoziomowy bufor FIFO, co umożliwia zapamiętanie zgłoszenia o tym samym priorytecie jak aktualnie obsługiwane. Podjęcie przez procesor obsługi przerwania (o najwyższym priorytecie) powoduje wyzerowanie bitu tego przerwania w IRR i ustawienie tego bitu w ISR.

W systemie wieloprocesorowym możliwe jest przekazanie zgłoszenia do procesora wskazanego na podstawie aktualnych priorytetów sprzętowych. Lokalny APIC ma dwa rejestry priorytetu: TPR (*task priority*), w których zapisany jest priorytet aktualnie realizowanego zadania, programowo ustalany przez system operacyjny oraz PPR (*processor priority*), przechowujący priorytet aktualnego zadania (przerwania). Przy braku przerwań PPR = TPR, zgłoszenie przerwania o priorytecie > PPR powoduje jego uaktualnienie PPR oraz blokadę obsługi zgłoszeń o niższych priorytetach.

Zakończenie obsługi przerwania powoduje:

- przesłanie EOI (zera) do rejestru lokalnego APIC (0FEE000B0h),
- wyłączenie bitu najwyższego priorytetu w ISR,
- sprawdzenie TMR i w przypadku gdy zapisano zgłoszenie poziomem przesłanie protokołu EOI do I/O APIC; I/O APIC zeruje wówczas odpowiedni bit zgłoszenia (przy zgłaszaniu zboczem skasowanie następuje automatycznie po przekazaniu zgłoszenia do lokalnego APIC).

A.5. Ewolucja listy rozkazów i80x86/Pentium

A.5.1. Rozszerzenia i modyfikacje architektury listy rozkazów i80386

(składnia standardowa Intel: *operacja* dst, src, [src2])
rozszerzone tryby adresowania: [er + skala*er (!esp)+disp]

BS% r-dest, r-src	przeglądanie rejestru r-src do pierwszej napotkanej jedynki w przód (*forward*, %=F) lub w tył (%=R), jeśli ZF=1 to numer bitu w rejestrze r-dest
BT% r/[..], r/imm	tylko testowanie (%=) lub testowanie z jednoczesnym ustawianiem (S), zerowaniem (R) albo dopełnianiem (C) bitu w r/[..] wskazanego przez r/imm, wynik testu w CF
CDQ / CWDE	rozszerzenie znakowe akumulatora EAX→EDX:EAX / AX→EAX
MOV%X r-dst, r-src	kopiowanie rejestru krótkiego r-src ($r\text{-}src_{7\text{-}0}$ / $r\text{-}src_{15\text{-}0}$) z rozszerzeniem znakowym (%=S) lub zerowym (%=Z) do rejestru o podwójnym rozmiarze r-dst ($r\text{-}src_{15\text{-}0}$ / $r\text{-}src_{31\text{-}0}$)
J*cc etykieta*	zasięg (-2^{15}, $2^{15}-1$) (*etykieta* NEAR) lub (-2^{7}, $2^{7}-1$) (*etykieta* SHORT)
SET*cc*	ustawianie stanu warunku *cc* w rejestrze jako **true** (0..01) lub **false** (0..00)
LOOP[*c*]	licznik specyficzny dla trybu (CX dla 16-bitowego, ECX dla 32-bitowego), [*c*] = Z|NZ
LOOP[*c*]%	[%=W] licznik zawsze w CX, [%=D] licznik zawsze w ECX, niezależnie od trybu
PUSH[A/F]%	złożenie na stos operandu (-ów) o rozmiarze specyficznym dla trybu ([%=]) lub wskazanym 16-bitowym ([%=W], oprócz PUSHA/PUSHF) albo 32-bitowym ([%=D])
POP[A/F]%	zdjęcie ze stosu operandu (-ów) o rozmiarze specyficznym dla trybu ([%=]) lub narzuconym 16-bitowym ([%=W] , oprócz POPA/POPF) albo 32-bitowym ([%=D])
SHLD r/[..], r, CL/*im*	przesunięcie logiczne złożenia operandów {r/[..] : r} 16- lub 32-bitowych, w lewo
SHRD r/[..], r, CL/*im*	lub w prawo, SF/ZF/PF odpowiednio do wyniku, CF - ostatni „wyrzucany" bit, OF=1 dla przesunięcia o 1, gdy jest zmiana znaku, poza tym nieokreślone

A.5.2. Rozszerzenia i modyfikacje architektury listy rozkazów i80486

BSWAP	odwrócenie kolejności bajtów w rejestrze 32b (LE↔BE), dostęp do bajtów HH i HL
CMPXCHG [..], r	jeśli ac = [..], to [..] = r, jeśli nie to [..] = ac, operandy 8-, 16- lub 32-bitowe, może być poprzedzone przedrostkiem LOCK (ac – akumulator al, ax lub eax)

```
        cmp eax, dest
        jne diff
        mov dest, reg
Diff:   jmp done
        mov eax, dest
done:
```

```
        cmp eax, dest
        mov eax, dest
        jne diff
        mov dest, reg
diff:
```

```
cmp eax, dest
mov eax, dest
cmoveq dest, reg
```

XADD r/[..], r	temp ← src+dst, src ← dst, dst ← tmp, (src=r/[..], dst=r) F jak w dodawaniu, może być poprzedzone przedrostkiem LOCK, operandy 8-, 16- lub 32-bitowe,

WBINVD (**0F 08h**)	zapis zwrotny (WB) i unieważnienie całej zawartości wewnętrznej pamięci podręcznej
INVD (**0F 09h**)	unieważnienie całej zawartości wewnętrznej pamięci podręcznej
INVLPG	unieważnienie deskryptora stronicowania w buforze TLB (INValid Line PagGe)

A.5.3. Rozszerzenia i modyfikacje architektury listy rozkazów Pentium, Pentium MMX i Pentium II

od późniejszych wersji 80486 - implementację wskazuje bit ID rejestru znaczników ($EFLAGS_{21}$)

CPUID (**0F A2h**) odpowiednio do zawartości rejestru EAX zwraca w rejestrach (EAX=0) – EAX - max wartość EAX akceptowana przez CPUID (0, 1, 2)
EBX:EDX:ECX = napis ASCII „Genu-ineI-ntel" (procesor AMD – „Auth-enti-cAMD" lub „Virt-uall-yAMD) w konwencji Little Endian ((BL)='G', (BH)='e', (EBX_{23-16})='n', (EBX_{31-24})='u', (DL)='i', (DH)='n', itd.)
(EAX=1) – EAX - kod typu, EDX - cechy procesora ((EAX_{0-3})= S, identyfikator ... (*Stepping ID*), (EAX_{4-7})= M, numer modelu (*Model*), (EAX_{11-8})= F, rodzina procesorów (*Family*), (EAX_{13-12})= T, typ (*Processor type*: 00-*original OEM processor*, 01 - *Intel overdrive processor*, 10 - *dual processor*) , przykładowo pierwszy PentiumPro: M= 0001_2, F= 0110_2, T= 00_2

zawartość EDX

bit	bit		zawartość
0	FPU	*FPU on chip*	wbudowana jednostka zmiennoprzecinkowa
1	VME	*Virtual 8086 Mode Enhancement*	implementacja trybu wirtualnego 8086
2	DE	*Debugging Extensions*	dodatkowe funkcje uruchomieniowe dla we/wy
3	PSE	*Page Size Extensions*	rozszerzenia rozmiaru strony – strona 4kB lub 4MB
4	TSC	*Time Stamp Counter*	licznik marki czasowej
5	MSR	*RDMSR and WRMSR Support*	wspomaganie odczytu i zapisu rejestrów specyficznych dla modelu
6	PAE	*Physical Address Extensions*	rozszerzenia adresu fizycznego (36 bitów adresowych)
7	MCE	*Machine Check Exception*	wyjątek „testowanie procesora"
8	CX8	*CMPXCHG8B Instruction*	instrukcja niepodzielna porównania /wymiany 8-bajtowej CMPXCHG8B
9	APIC	*APIC on Chip*	wbudowany inteligentny sterownik przerwań
12	MTRR	*Memory Type Range Register*	rejestr zakresów typów pamięci dostępnych na różnych magistralach
13	PGE	*PTE Global Bit*	ogólny bit PTE
14	MCA	*Machine Check Architecture*	architektura wspomagająca wykrywanie i obsługę wyjątków
15	CMOV	*Conditional Move Instruction*	implementowane instrukcje CMOV, FCMOV oraz
23	MMX	*MMX Technology*	implementowane rozszerzenia MMX

(EAX=2) - EAX:EBX:ECX:EDX - informacja o wewnętrznej cache i TLB

CMPXCHG8B [..]	działanie analogiczne jak cmpxchg na argumentach 8B (rejestry EDX:EAX, ECX:EBX)
CMOV*cc* r, r/[..]	warunkowe kopiowanie danych, argumenty 16- / 32-bitowe, warunki jak przy skokach

(**0F 4xh („AMB") ...**)	(kod drugiego półbajtu analogiczny jak w rozkazach j*cc* i s*cc*, kolejne zależnie od trybu adresowania zapisanego w bajcie trybu AMB)
FCMOV*cc*	warunkowe kopiowanie argumentów zmiennoprzecinkowych
RDTSC (**0F 31h**)	odczyt licznika cykli (*time stamp counter*, TSC) do rejestru EDX:EAX (*tryb chroniony*)
RDMSR (**0F 32h**)	odczyt rejestru specyficznego modelu (*model specific register*) do rejestru EDX:EAX, argument w ECX (0 – MCA, *machine check address* - adres instrukcji wywołującej wyjątek, 1 – MCT, *machine check type* - typ wyjątku) (? *tryb chroniony* ?)
WRMSR (**0F 30h**)	zapis rejestru specyficznego modelu (*model specific register*) do rejestru EDX:EAX, argument w ECX (0 – MCA, 1 – MCT – !! *tylko tryb chroniony* !!
RSM (**0F AAh**)	powrót z trybu systemowego SMM (wejście przez impuls zewnętrzny SMI#)
RDECR (**0F 36h**)	odczyt rejestru sterującego (tryb chroniony, poziom nadzoru)
WRECR (**0F 36h**)	zapis rejestru sterującego (tryb chroniony, poziom nadzoru)

A.5.4. Lista rozkazów MMX

Jednostka MMX operuje na argumentach 64-bitowych i używa rejestrów pliku zmiennoprzecinkowego, lecz w trybie dostępu swobodnego a nie sekwencyjnego jak FPU (rejestry tworzą stos cykliczny, rejestr szczytu stosu ma zawsze numer ST(0), stos zawiera zawsze 8 rejestrów).

Ponieważ przy wykonywaniu działań zmiennoprzecinkowych rejestry stosu są etykietowane między innymi każdemu przypisana jest cecha ważności (aktualności), to po wykonaniu działań MMX konieczne jest unieważnienie zawartości wszystkich rejestrów (rozkazem EMMS). Przełączenie na tryb MMX następuje automatycznie przy wykonaniu rozkazu kopiowania MMX (MOVQ lub MOVD).

Przy wykonywaniu działań MMX rejestry stosu są nazywane **mm**# (numery od 0 do 7), dla każdego działania jeden z operandów (dla działań arytmetycznych tylko operand źródłowy) może być umieszczony w pamięci.

Wykluczone jest jednoczesne użycie tych rejestrów jako stosu zmiennoprzecinkowego a po zakończeniu operacji MMX konieczne jest unieważnianie rejestrów (w kontekście FPU ich zawartość nie jest określona).

Rozkazy MMX operują na 8-, 16- lub 32-bitowych rekordach słowa podwójnej długości (64-bitowego), wykonując wskazane działanie współbieżnie na wszystkich rekordach. Argumenty rozkazów MMX w poszczególnych rekordach są zawsze traktowane jako stałoprzecinkowe (całkowite lub naturalne)

Metody formowania wyniku działań arytmetycznych

- *modulo* rozmiar bajtu / półsłowa / słowa
- *nasycanie jednokierunkowe* –spłaszczanie wyniku od góry - liczby naturalne (U)
- *nasycanie dwukierunkowe* – symetryczne spłaszczanie wyniku względem zera - liczby całkowite (S)