

ПЛАМЕНКА БОРОВСКА

КОМПЮТЪРНИ СИСТЕМИ



КОМПЮТЪРНИ СИСТЕМИ

© 1999 by CIELA - COURT AND LAWYERS
© 1999 by CIELA - COURT AND LAWYERS
© 1999 by CIELA - COURT AND LAWYERS

ПЛАМЕНКА БОРОВСКА

КОМПЮТЪРНИ СИСТЕМИ

- © Пламенка Боровска, *автор*
© Любомир Пенев, *художествен дизайн на корицата*
© СИЕЛА – СОФТ ЕНД ПАБЛИШИНГ

София • 2003

ISBN 954-649-633-2



СЪДЪРЖАНИЕ

УВОД	7
------------	---

Глава 1

ТЕХНОЛОГИЧНИ И АРХИТЕКТУРНИ АСПЕКТИ НА КОМПЮТЪРНИТЕ СИСТЕМИ	9
---	---

Глава 2

АРХИТЕКТУРА НА ЦЕНТРАЛНИЯ ПРОЦЕСОР	15
2.1. Общ преглед	15
2.2. Представяне на данните	16
2.3. Система инструкции	25
2.4. Компютри със съкратен набор инструкции (RISC)	40
2.5. Регистри на процесора	47
2.6. Обработка на прекъсванията и изключенията	50
2.7. Инструкционни конвейери	56
2.8. Суперскаларни процесори	66
2.9. Архитектура на процесори с хипернишкова технология	79
2.10. Тенденции в развитието на съвременните процесори (обобщение)	89

Глава 3

АРХИТЕКТУРА НА КОМПЮТЪРНИТЕ СИСТЕМИ	92
3.1. Йерархия на паметта в компютърните системи	92
3.1.1. Полупроводникови памети	95
3.1.2. Главна памет	97
3.1.3. Кеш памет	99
3.1.4. Външна памет	112
3.2. Управление на паметта в компютърните системи	120
3.2.1. Сегментиране на паметта	120
3.2.2. Виртуална памет	130
3.2.3. Управление на кеш паметта	138
3.3. Защита	148

3.4. Архитектурна поддръжка на мултипроцесорни системи	154
3.5. Входно-изходна система	163
3.7. Шинна архитектура	176

Глава 4

ПРОИЗВОДИТЕЛНОСТ НА КОМПЮТЪРНИ СИСТЕМИ	185
--	-----

Глава 5

ПАРАЛЕЛНИ КОМПЮТЪРНИ СИСТЕМИ	197
------------------------------------	-----

5.1. Паралелни компютърни архитектури	197
5.2. Таксономия на паралелните <i>mimd</i> компютърни архитектури	204
5.2.1. Програмни модели за паралелни компютри	205
5.2.2. Архитектури с обща физическа памет: архитектура PRAM	209
5.2.3. MIMD архитектури с разпределена памет	211
5.2.4. Основни категории архитектури с разпределена памет	212
5.2.5. Архитектури с разпределена обща памет	215
5.2.6. Многонишкови архитектури	220
5.2.7. Архитектури с виртуална памет	222
5.3. Вътрешносистемни комуникационни мрежи за паралелни компютри	224
5.4. Комуникации в мултикомпютри	234
5.5. Структура и организация на мрежи за паралелни компютри с конвейеризиран трансфер	240
5.6. Производителност и ефективност на паралелни компютърни системи	266
5.6.1. Пространствени измерения на производителността на паралелните компютри	266
5.6.2. Производителност на вътрешносистемни комуникационни мрежи за паралелни компютри	273

ЛИТЕРАТУРА	285
------------------	-----

УВОД

В книгата са представени основните тенденции в развитието на микроархитектурата на съвременните процесори – суперскаларна обработка, хипернишкова технология, спекулативни предварителни изчисления, интензивно използване на кеш памети на няколко нива, RISC архитектури. Анализирани са проблемите на суперконвейеризираната микроархитектура на съвременните процесори, свързани с блокирането на конвейерите – грешно предсказани условни преходи, липси в кеша, прекъсвания и др. Разгледани са архитектурните аспекти на съвременните компютърни системи – виртуална памет, входно-изходна система, шинна архитектура. Особено внимание е отделено на паралелните компютърни архитектури като PRAM, многонишковите архитектури, архитектури с разпределена обща памет, паралелни архитектури с виртуална памет, мултикомпютри и др. Представената класификация на паралелните компютърни архитектурни форми се основава на използваните програмни модели и модели на комуникациите. Разгледани са структурата и организацията на комуникационните вътрешносистемни мрежи за паралелни компютри – топологии, комуникационни протоколи, маршрутизация, предотвратяване и елиминиране на мъртво блокиране, архитектура на комутаторите. Представени са методи и техники за оценка на производителността и ефективността на съвременните компютърни системи.

Книгата е предназначена за студентите от техническите специалности на университетите и техническите колежи, изучаващи дисциплините „Компютърни системи“ и „Компютърни архитектури“, а също така и за докторанти и специалисти, работещи в областта на компютърните архитектури.

Авторката д-р Пламенка Боровска е доцент в катедрата по Компютърни системи към Факултета по компютърни системи и управление на Техническия Университет – София.

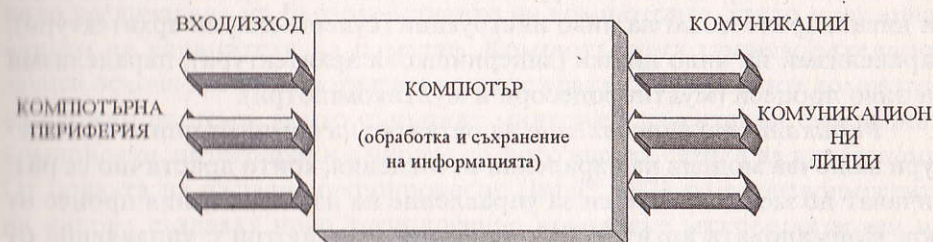
Глава 1

ТЕХНОЛОГИЧНИ И АРХИТЕКТУРНИ АСПЕКТИ НА КОМПЮТЪРНИТЕ СИСТЕМИ

Основните функции на компютърната система са информационна обработка, съхраняване на информацията, информационен трансфер и управление (фиг. 1.1). Компютърната система трябва да осъществява информационен обмен с външния свят. Операционната среда на компютърната система включва устройства, които представляват източници или консуматори на информация. Получаването или предаването на информация от/към компютъра се осъществява чрез т.нар. периферни устройства, директно свързани към компютъра, а процесът се нарича вход/изход (Input/Output). Когато информацията се получава от или се предава на отдалечено устройство, процесът се нарича комуникация.

Структурата на компютърната система включва четири базови градивни модула:

1. Централен процесор, осъществяващ информационната обработка;
2. Памет, осъществяваща съхраняването на информацията;
3. В/И (входно/изходна) система, осъществяваща информационния обмен между компютъра и външния свят;
4. Вътрешни системни връзки, осъществяващи комуникациите между централния процесор, паметта и входно-изходната система.



Фиг. 1.1. Функционален модел на компютърната система

Централният процесор обединява две компоненти: аритметично-логическото устройство (АЛУ), отговорно за обработката на информа-

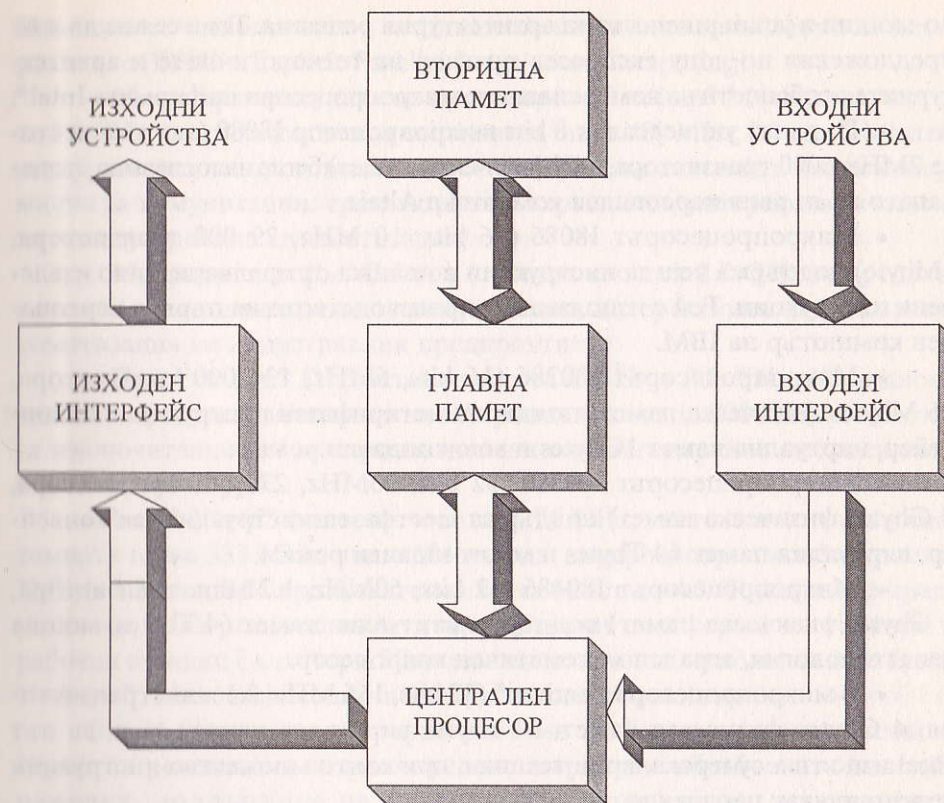
цията и управляващото устройство (УУ), координиращо дейността на компютърните ресурси.

Структурата на компютърната система обхваща базовите градивни модули и начина на тяхното свързване. Функционирането на компютъра включва действието на всеки компонент като част от структурата. Компютърната архитектура включва структурата на компютъра, модела на изчислителния процес и атрибутите, които директно влияят върху логическото изпълнение на програмата и системната производителност. Важни архитектурни белези са системата от инструкции, дължината на машинната дума, режимите за адресиране на паметта, механизмите за вход-изход и др.

Исторически първа се е появила класическата фон Нойманова архитектура (по името на американския професор von Neumann). Обобщеният структурен модел на компютър с класическа фон Нойманова архитектура е представен на фиг. 1.2. Той съдържа централен процесор, обединяващ АЛУ и УУ, памет (главна и вторична) за съхраняване на програмите и данните и входно-изходна система. Изчисленията се осъществяват под управлението на наредена последователност от инструкции (машинни команди), наречена компютърна програма. Характерно за класическата фон Нойманова архитектура е чисто последователният характер на изпълнението на програмата, т.е. изпълнението на $i+1$ инструкция започва едва след завършването на изпълнението на i -тата инструкция.

Компютърните архитектури се развиват в две основни направления: *еволюционно* и *радикално*. Еволюционното развитие на компютърните архитектури се основава на усъвършенстването на класическата фон Нойманова архитектура, като се запазва механизмът за управление на изчислителния процес от инструкции и се използват паралелизми на всички нива: паралелизъм на ниво инструкции (суперскаларни архитектури), паралелизми на ниво нишки (хипернишковки архитектури), паралелизми на ниво процеси (мултипроцесори и мултикомпютри).

Радикалното направление на развитие на компютърните архитектури включва модели на паралелни изчисления, които драстично се различават по механизмите си за управление на изчислителния процес от фон Ноймановата архитектура: паралелни компютри с управление от потока данни (data flow), редуccionни архитектури, логически паралелни компютри. При радикалните компютърни архитектури отпадат инструкциите, операторите за присвояване, използването на програмен брояч.



Фиг. 1.2. Обобщен структурен модел на компютър с класическа фон Нойманова архитектура

Цялостното развитие на компютърната техника води до непрекъснато увеличаване на бързодействието на компютрите, както и увеличаването на капацитета на паметта. Компютърната производителност зависи основно от технологията и архитектурата. Съвременните компютри са сложни системи, които съдържат милиони елементарни електронни компоненти, интегрирани в чипове с много висока степен на интеграция. От появата на първия микропроцесор (Intel®, 1971 г.) производителите на чипове създават ново технологично поколение микропроцесори на всеки три години. При производството на динамични памет, базовата технология за главната компютърна памет, капацитетът на чиповете памет се увеличава приблизително четирикратно за всяко ново поколение микропроцесори. Иновациите в технологията водят до въвеждането на

по-мощни и усъвършенствани архитектурни решения. Това се вижда и от предложения по-долу експресен преглед на технологичните и архитектурните особености на комерсиалните микропроцесори на фирмата Intel®:

- Първият универсален 8 bits микропроцесор I8080 (тактова честота 2MHz, 6000 транзистора, 64К физическа памет) беше използван за създаването на първия персонален компютър Altair.

- Микропроцесорът I8086 (16 bits, 10 MHz, 29 000 транзистора, 1Mbyte) поддържа кеш за инструкции и опашка от предварително извлечени инструкции. Той е използван за производството на първия персонален компютър на IBM.

- Микропроцесорът I80286 (16 bits, 6MHz, 134 000 транзистора, 16 Mbytes физическа памет) поддържа четирифазен инструкционен конвейер, виртуална памет 1Gbytes и многозадачен режим.

- Микропроцесорът I80386 (32 bits, 33MHz, 275 000 транзистора, 4 Gbytes физическа памет) поддържа шестфазен инструкционен конвейер, виртуална памет 64 Tbytes и многозадачен режим.

- Микропроцесорът I80486 (32 bits, 50MHz, 1.2 млн. транзистора, 4 Gbytes физическа памет) поддържа виртуална памет 64 Tbytes, мощна кеш технология, вграден математичен копроцесор.

- В микропроцесора Pentium® (32 bits, 166 MHz, 3.1 млн. транзистора, 4 Gbytes физическа памет, 64 Tbytes виртуална памет) за пръв път Intel използва суперскаларни техники, при които множество инструкции се изпълняват паралелно.

- При Pentium® Pro (64 bits, 200 MHz, 5.5 млн. транзистора, 64 Gbytes физическа памет, 64 Tbytes виртуална памет) се усъвършенства суперскаларната архитектура с агресивно използване на преименуването на регистри, предсказване на преходи, анализ на потока данни и спекулативни изчисления, средства, поддържащи мултипроцесорни системи.

- Pentium® II (64 bits, 300 MHz, 7.5 млн. транзистора, 64 Gbytes физическа памет, 64 Tbytes виртуална памет) поддържа технологията MMX на Intel, която се използва ефективно за обработката на видео-, аудио- и графични данни.

- В Pentium® III (тактова честота 1.26 GHz, системна шина 133 MHz) са заложени допълнителни инструкции за поддръжка на софтуера за 3D графика.

- Intel® Itanium® 2 (тактова честота 1.5 GHz, тактова честота на системната шина 400 MHz с ширина 128bits, 6 Mbytes L3 кеш) представлява двупроцесорна конфигурация за изграждане на сървърни платформи за високопроизводителни технически, бизнес и индустриални приложения.

Използва се технологията с експлицитно паралелно изчисление на инструкциите (EPIC – Explicit Parallel Instruction Computing);

- Микроархитектурата на процесорите Pentium® M (тактова честота 1.6 GHz, тактова честота на системната шина 400 MHz, L2 кеш 1 Mbytes) осигурява висока производителност на вградени системи с ниска консумация за комуникации, транзакционни терминали, интерактивни клиенти и индустриална автоматизация;

- Микропроцесорът Celeron (тактова честота 2.5 GHz) е предназначен за вградени системи за комуникации, интерактивни клиенти и автоматизация на индустриални предприятия;

- При Pentium® IV (тактова честота 2 GHz – 3.2 GHz, 42 милиона транзистора, тактова честота на системната шина до 800 MHz) се постига изключително висока производителност благодарение на хипернишковата архитектура.

- Intel® Xeon™ (тактова честота 3.06 GHz, тактова честота на системната шина 533 MHz, чип с 604 извода, двупроцесорна конфигурация, 1Mbyte L3 кеш, В/И пропускателна способност до 4.3 Gbytes/sec) е предназначен за изграждане на мултипроцесорни платформи на сървъри и работни станции с клъстери. Използва иновационни технологии като Intel® Netburst™ Microarchitecture и хипернишкова технология.

Съвременните компютърни системи трябва да осигурят високоскоростна реализация на приложения, свързани с обработката на изображения, поддържане на видеоконферентни връзки, мултимедия, разпознаване и синтез на говор, обработка на транзакции за електронен бизнес, поддръжка на големи бази от данни, научни и технически приложения с голям обем на изчисленията, поддръжка на компютърни мрежи клиент-сървър.

Съвременните технологични платформи включват основно силициевите нанотехнологии, осигуряващи възможност за въвеждането на редица архитектурни иновации в микроархитектурата на микропроцесорите. Основните тенденции в развитието на микроархитектурата на съвременните микропроцесори са усъвършенстване на хипернишковите технологии в нейните три разновидности: многонишкови изчисления в едночипови мултипроцесори, симултантни многонишкови изчисления и динамични многонишкови изчисления. Също така се правят изключително интензивни изследвания при усъвършенстването на техниките за предсказване на преходите, анализа на потока данни, свързан с планирането на изпълнението на инструкциите и спекулативните изчисления с цел максимално натоварване на изчислителните ресурси на централния процесор.

Развитието на съвременните компютърни системи е неизбежно свързано с усъвършенстването на структурата и организацията на йерархията на паметта. Плътноста на чиповете динамична RAM памет, базовата технология на главната компютърна памет, расте с едновременно намаляване на времето за достъп. На фона на непрекъснатото увеличаване на бързодействието на съвременните микропроцесори и памети критичното място за производителността на компютърната система остава пропускателната способност на тракта централен процесор-главна памет, по който се поддържа постоянен поток от инструкции и данни между чиповете памет и чипа на централния процесор. За преодоляване на това тясно място се правят непрекъснати усъвършенствания на системната архитектура в следните направления:

- Разширяване на ширината на информационните пътища на тракта централен процесор-главна памет и съответно „разширяване“ на информационните единици в чиповете памет;
- С цел повишаване на ефективността на интерфейса с паметта се използва кеширане или буфериране в чиповете памет;
- За да се редуцира честотата на обръщенията към паметта се осигурява кеширане на няколко нива между процесора и паметта, което включва кешове в рамките на процесорния чип и кешове в непосредствена близост до него.
- Увеличаване на пропускателната способност на тракта централен процесор-главна памет чрез високоскоростни шини с йерархична структура за буфериране и структуриране на потока данни.

За достигане на висока производителност на компютърната система е от съществено значение ефективността на входно-изходната система. В рамките на компютърната система могат да бъдат включени широк спектър периферни устройства с високи изисквания към пропускателната способност на входно-изходната система. Стратегиите включват кеширане, буфериране и многоетажни структури на шините, като етажите са функционално обособени в съответствие със скоростните характеристики на свързаните устройства.

Глава 2

АРХИТЕКТУРА НА ЦЕНТРАЛНИЯ ПРОЦЕСОР

2.1. ОБЩ ПРЕГЛЕД

Централният процесор е изграден от елементарни електронни устройства, които съхраняват информацията в двоична (бинарна) форма и изпълняват прости логически операции над данните. Логическите променливи са такива променливи, които могат да приемат две стойности: *истина* („true“) и *лъжа* („false“). Действията с логически променливи са обект на разглеждане в Булевата алгебра. В компютъра преобразуването на логическите променливи се осъществява от комбинационни логически схеми. Те представляват многоетажни структури от логически елементи, които осъществяват преобразуването на логическите променливи съобразно логическите функции, които те реализират. Изходните сигнали на комбинационните логически схеми закъсняват спрямо входните сигнали с време, равно на закъснението за разпространение на сигналите в схемата. Те са активни дотогава, докато са активни входните сигнали на логическата схема, т.е. комбинационните логически схеми не съхраняват информация, а само я преобразуват. Съхраняването на информацията се осъществява в електронни елементи памет, наречани тригери. Тригерът представлява логическа схема с две устойчиви състояния: логическа 1 и логическа 0. Той съхранява един бит информация. Регистърът представлява набор от тригери, броят на които в общия случай се определя от дължината на машинната дума, т.е. от разрядността на микропроцесора. При 64-битовите процесори дължината на регистрите е 64 бита, т.е. те съдържат 64 тригера. Процесорите могат да съдържат регистри с дължина двойна или четворна машинна дума.

Информационните единици в компютрите са както следва:

- 1 байт съдържа 8 бита ($1\text{ byte} = 8\text{ bits}$);
- 1 килобит съдържа 2^{10} бита или 1Кбита ($1\text{ Kbits} = 1024\text{ bits} = 2^{10}\text{ bits}$)
- 1 килобайт съдържа 2^{10} байта или 1Кбайта ($1\text{ Kbytes} = 1024\text{ bytes} = 2^{10}\text{ bytes}$)
- 1 Mbytes съдържа 1024 Kbytes
- 1 Gbytes съдържа 1024 Mbytes
- 1 Tbytes съдържа 1024 Gbytes

Дължината на машинната дума на 64-битовите процесори е 8 байта.

Централният процесор на съвременен компютър включва богат спектър от функционални блокове: шинно интерфейсно устройство, изпълнително устройство, устройство за инструкции, адресно устройство, странично и сегментно устройства, управляващо устройство, кеш за инструкции, кеш за данни.

Изпълнителното устройство включва аритметично-логическото устройство (АЛУ), което осъществява аритметичните и логическите операции над данните. Операндите и резултатите от аритметичните и логическите операции се съхраняват в регистри, които осигуряват временното им съхраняване за подаване на операндите към АЛУ и поемане на резултата след изпълнението на операцията от АЛУ. Входните сигнали за АЛУ са операндите от регистрите и сигнали от управляващото устройство на централния процесор, а изходните сигнали са резултатите от изпълнените операции, съхранявани временно в регистри и флагове, отразяващи вида на получените резултати.

Централният процесор функционира в два основни режима: *системен режим* (S – System) и *потребителски режим* (U – User). Системният режим (нарича се още режим супервайзор) е привилегирован и при него е активна операционната система. При него всички инструкции от набора инструкции на процесора са разрешени за изпълнение, както и всички ресурси на компютърната система са достъпни. В режим потребител се изпълняват приложните програми. При него не са разрешени за изпълнение т.нар. привилегировани инструкции (разрешени за изпълнение само в режим супервайзор), както и не са достъпни всички ресурси на компютърната система.

2.2. ПРЕДСТАВЯНЕ НА ДАННИТЕ

Общите категории данни в компютрите са *числа, адреси, символи, логически данни*. По своята същност адресите са числа и в компютъра те се третират като цели числа без знак. В някои компютри са заложили по-сложни типове данни като например структури от данни или стринг от символи. Числовите данни в компютрите са три типа: *цели числа (числа с фиксирана точка), числа с плаваща точка или пакетираните десетични числа*.

Бройни системи. В човешката практика се използват два вида бройни системи: *непозиционни* и *позиционни*. При *непозиционните* бройни

системи теглото на цифрата не зависи от нейната позиция. Пример за *непозиционна* бройна система е римската бройна система.

При *позиционните* бройни системи теглото на всяка цифра в числото зависи от нейната позиция. Така например цифрата 5 има различно тегло в числото 5555 в десетичната бройна система, която се използва в нашето ежедневие. Първата цифра 5 има тегло 1000, втората – 100, третата – 10, а четвъртата – 1.

При позиционните бройни системи числата се представят като полином от вида:

$$A = \sum_i a_i \cdot S^i, \text{ за } i \in [-m, +n], \quad (2.1)$$

където a_i е цифрата в i -тата позиция на числото, S – основата на бройната система, а индексът i показва позицията на цифрата в числото.

Основата на бройната система показва броя на различните цифри, които се използват за представяне на числата. Така например при десетичната бройна система (основа 10) за представянето на числата се използват 10 различни цифри: цифрите от 0 до 9. Представянето на числото 5555, 55 във вид на полином в десетична бройна система ще има следния вид:

$$5555, 55_{(10)} = 5 \cdot 10^3 + 5 \cdot 10^2 + 5 \cdot 10^1 + 5 \cdot 10^0 + 5 \cdot 10^{-1} + 5 \cdot 10^{-2} \quad (2.2)$$

При двоичната бройна система (основа две) се използват само две цифри за представянето на числата: 0 и 1. Представянето чрез полином в двоична система ще има следния вид:

$$11010_{(2)} = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 \quad (2.3)$$

При шестнайсетичната бройна система се използват 16 символа за означаване на числата: цифрите от 0 до 9 и символите A – за 10, B – за 11, C – за 12, D – за 13, E – за 14 и F – за 15. Представянето на число чрез полином в шестнайсетична система ще има следния вид:

$$13A4C_{(16)} = 1 \cdot 16^4 + 3 \cdot 16^3 + 10 \cdot 16^2 + 4 \cdot 16^1 + 12 \cdot 16^0 \quad (2.4)$$

Във всяка бройна система представянето на основата е $S=10$, защото $S=1 \cdot S^1 + 0 \cdot S^0$. В двоична бройна система $S=10_{(2)} = 1 \cdot 2^1 + 0 \cdot 2^0 = 2_{(10)}$. В шестнайсетична бройна система $S=10_{(16)} = 1 \cdot 16^1 + 0 \cdot 16^0 = 16_{(10)}$.

Преобразуването на представянето на числата от произволна бройна система в десетична бройна система става чрез развиването му в полином при представянето на цифрите в десетична бройна система (вж. 2.3 и 2.4). Преобразуването на представянето на числата от десетична

бройна система в двоична бройна система се осъществява, като числото се разделя последователно на две до получаването на частно 0. Впоследствие остатъците се записват обратно на реда на тяхното получаване. Преобразуването на числото 13 от десетична в двоична система 1101 е показано на фиг. 2.1. За да проверим верността на преобразуването, развиваме полученото двоично число в полином с основа 2, т.е.

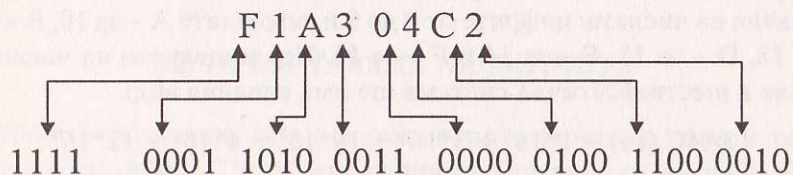
$$1101_{(2)} = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 1 = 13_{(10)} \quad (2.5)$$

При преобразуването на числата от двоична в шестнайсетична бройна система и обратно се използва фактът, че една шестнайсетична цифра се представя с 4 бита, т.нар. тетрада. При преобразуването на числата от шестнайсетична в двоична бройна система всяка шестнайсетична цифра се представя чрез съответната ѝ тетрада в двоичен код (фиг. 2.2).

Деление на 2	Частно	Остатък	Двоично представяне
13/2	6	1	13 ₍₁₀₎ = 1101 ₍₂₎
6/2	3	0	
3/2	1	1	
1/2	0	1	

Фиг. 2.1. Преобразуване на числото 13 от десетична в двоична система

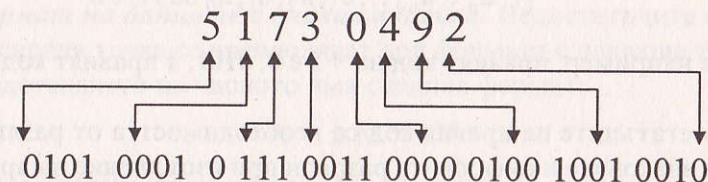
За приложения с интензивен вход/изход и сравнително малко и прости изчисления се предпочитат числата да се съхраняват и обработват в *пакетиран десетичен код* (packed decimal). Този код е по-малко компактен от директното бинарно представяне, но за сметка на това се избягват допълнителните разходи за преобразуването.



Фиг. 2.2. Преобразуване на числата от шестнайсетична бройна система и обратно

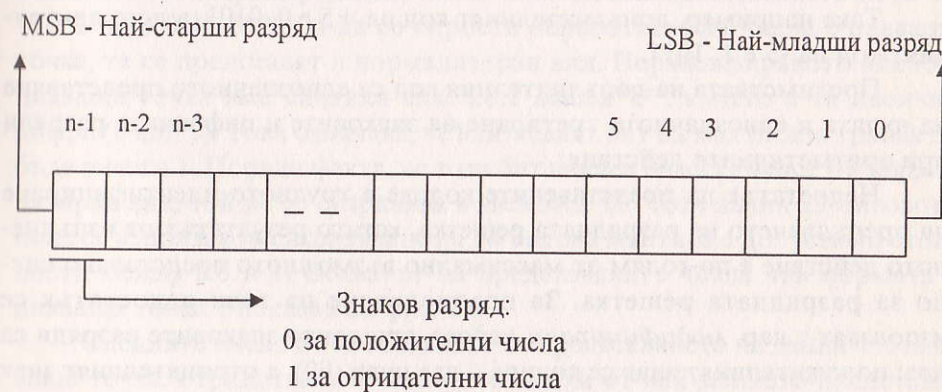
Най-голямата цифра в десетична бройна система е 9, чието представяне изисква една тетрада (1101). Следователно за двоичното представяне на десетично число са необходими толкова тетради, колкото са цифрите на десетичното число. Във всяка тетрада се записва двоичният

еквивалент на съответната десетична цифра на числото. При преобразуването на дадено число от двоична в десетична бройна система двоичното число се разделя на тетради и за всяка тетрада се записва десетичният еквивалент (фиг. 2.3).



Фиг. 2.3. Преобразуване на число от двоична в десетична бройна система и обратно при пакетирано десетично кодиране

Формат на данните с фиксирана точка. При това представяне обхватът на представяните числа зависи от дължината на машинната дума (разрядността на решетката). При 64-битовите микропроцесори разрядната решетка съдържа 64 бита. Битовете се номерират от дясно на ляво (фиг. 2.4), като най-левият бит е най-старшият и се означава с MSB (Most Significant Bit), а най-десният бит е най-младшият бит и се означава с LSB (Least Significant Bit). Най-левият бит съдържа знака на числото, като положителните числа имат знаков разряд 0, а отрицателните числа – знаков разряд 1. При дължина на разрядната решетка n бита, максималното число, което може да бъде представено, е $2^{n-1}-1$, а най-малкото отрицателно представяно число е -2^{n-1} .



Фиг. 2.4. Формат на данните с фиксирана точка

Представянето на данните в *прав код* има следния вид:

$$A_{\text{ПК}} = \begin{cases} 0, a_{n-2}, a_{n-1} \dots a_3, a_2, a_1, a_0 & \text{за } A \geq 0 \\ 1, a_{n-2}, a_{n-1} \dots a_3, a_2, a_1, a_0 & \text{за } A < 0 \end{cases} \quad (2.6)$$

Така например, правият код на +5 е 0, 0101, а правият код на -5 е 1, 0101.

Недостатъците на правия код са необходимостта от различно третиране на знаковите и цифровите разряди при изпълнение на аритметичните действия и двойственото представяне на нулата (0, 0000000000000000 и 1, 0000000000000000).

Представянето на данните в *обратен код* има следния вид:

$$A_{\text{ОК}} = \begin{cases} 0, a_{n-2}, a_{n-1} \dots a_3, a_2, a_1, a_0 & \text{за } A \geq 0 \\ 1, \overline{a_{n-2}}, \overline{a_{n-1}} \dots \overline{a_3}, \overline{a_2}, \overline{a_1}, \overline{a_0} & \text{за } A < 0 \end{cases} \quad (2.7)$$

Така например, обратният код на +5 е 0, 0101, а обратният код на -5 е 1, 1010.

Представянето на данните в *допълнителен код* има следния вид:

$$A_{\text{ДК}} = \begin{cases} A_{\text{ПК}} & \text{за } A \geq 0 \\ A_{\text{ОК}} + 0,000\dots 01 & \text{за } A < 0 \end{cases} \quad (2.8)$$

Така например, допълнителният код на +5 е 0, 0101, а допълнителният код на -5 е 1, 1011.

Предимствата на допълнителния код са еднозначното представяне на нулата и еднозначното третиране на знаковите и цифровите разряди при аритметичните действия.

Недостатък на представените кодове е трудното идентифициране на препълването на разрядната решетка, когато резултатът от изпълненото действие е по-голям от максимално възможното представяно число за разрядната решетка. За преодоляване на този недостатък се използват т.нар. *модифицирани кодове*, при които знаковите разряди са два: положителният знак се кодира с две нули (00), а отрицателният знак – с две единици (11). Получаването на 10 или 01 в знаковите разряди на резултата от аритметичното действие показва възникнало препълване на разрядната решетка. Всички аритметични действия над числата с фиксирана точка се извършват в допълнителен код.

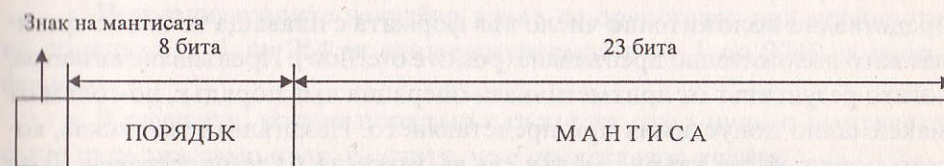
Основен недостатък на формата на данните с фиксирана точка е ограниченият обхват на представяните числа. Много малките числа се третират като машинна нула. Представянето на числа извън представяния формат, изисква мащабиране.

Формат на данните с плаваща точка. Недостатъците на формата с фиксирана точка се преодоляват при формата с плаваща точка. При него представянето на числото има следния формат:

$$A = \pm M * B^{\pm E} \quad (2.9)$$

където A е представяното число, M – мантиса, E – порядък, B – основа, която не се задава в явен вид, тъй като е една и съща за всички числа.

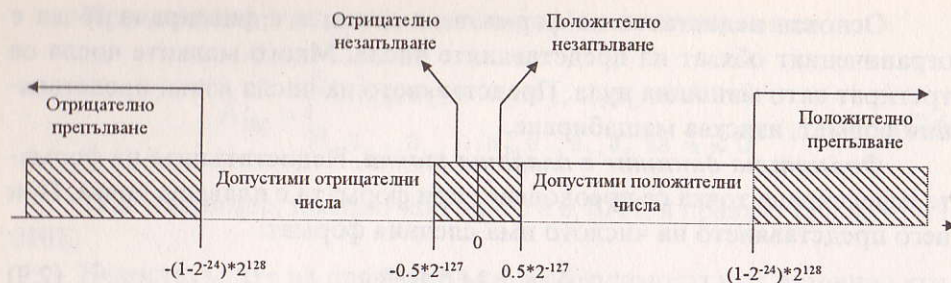
На фиг. 2.5 е показан типичен 32-битов формат с плаваща запетая. Най-левият бит съдържа знака на числото (0 – за положителни числа, 1 – за отрицателни числа). Стойността на порядъка се съдържа в следващите 8 бита. Останалите 23 бита съдържат мантисата.



Фиг. 2.5. 32-битов формат на данните с плаваща запетая

Числата с плаваща запетая могат да имат различни форми на представяне. Така например, дадено число може да има представянето $0.111 * 2^5$, $111 * 2^2$ или $0.0110 * 2^6$. За да се опрости обработката на числа с плаваща точка, те се представят в нормализиран вид. Нормализираното число с плаваща точка има следния вид: $\pm 0.1 a a a \dots a * 2^{\pm E}$, където a са двоични цифри (0 или 1). Това означава, че най-левият бит на мантисата трябва да бъде винаги 1. Поради факта, че този бит винаги има стойност 1 в нормализиран вид, той не се съхранява в явен вид. По този начин 23-битовото поле се използва за съхраняване на 24-битова мантиса с допустими стойности между 0.5 и 1. Обхватът на представяните числа във формата с плаваща точка е показан на фиг. 2.6.

Следните числа не са обхванати от представянето на данните с плаваща точка: отрицателните числа, по-малки от най-малкото представяно отрицателно число, се третират като отрицателно препълване (negative overflow). Отрицателните числа, по-големи от най-голямото представяно отрицателно число в обхвата и по-малки от 0, се третират като



Фиг. 2.6. Обхват на представяните числа с 32-битов формат с плаваща точка

отрицателно незапълване (negative underflow). Положителните числа, по-малки от най-малкото представяно положително число във формата с плаваща точка, се третират като положително незапълване (positive underflow). Положителните числа, които са по-големи от най-голямото представяно положително число във формата с плаваща точка, се третират като положително препълване (positive overflow). Препълване възниква, когато резултатът от аритметичната операция има порядък, по-голям от максимално допустимия при представянето. Незапълване възниква, когато порядъкът е твърде малък, за да може да бъде представен. Този случай е по-малко сериозен от препълването, защото поради малките стойности на порядъците на числата може да се приеме със задоволителна точност, че тези числа имат стойност приблизително 0. Нулата също не може да бъде представена във формата с плаваща точка. За тази цел се използва специално предназначена комбинация.

При формата на данните с плаваща точка броят на представяните числа не е по-голям от този при формата с фиксирана точка. Представяните числа с плаваща точка не са разпределени равномерно по числовата ос както при формата с фиксирана точка. Възможните стойности на представяните числа с плаваща точка са съгъстени близо до нулата и се разреждат с отдалечаването от нея. Това е и един от основните недостатъци на представянето с плаваща запетая. Резултатите, които се получават при изчисленията, не са точни и те трябва да бъдат закръглявани до най-близкото представяно число. Съществува зависимост между обхвата на представяните числа и точността на изчисленията. За да се увеличат както обхватът, така и точността е необходимо да се използват повече битове във формата с плаваща точка. Повечето компютри осигуряват числа с единична точност и с двойна точност.

Използването на по-голямо поле за порядъка в рамките на формата с плаваща запетая води до разширяването на обхвата на представяните числа за сметка на намаляването на точността.

Почти при всички съвременни компютри за представянето на данните във формат с плаваща точка се прилага стандартът на Института на електроинженерите и инженерите по електроника **IEEE Standard 754**, който е разработен с цел осигуряването на преносимостта на програмите и стимулирането на разработването на програми за интензивни числени приложения. Този стандарт дефинира единичен 32-битов и 64-битов двоен формат с плаваща точка. Стандартът дефинира също така два разширени формата, единичен и двоен, които се използват при междинни изчисления. По-голямата точност на разширените формати дава възможност да се намалят грешките при закръгляване. Особените случаи възникват, когато полето на порядъка съдържа само единици (255 за единичен формат и 2047 за двоен формат) или само нули:

- Нормализираните ненулеви числа се представят при стойности на порядъка от 1 до 254 за единичен формат и от 1 до 2046 за двоен формат.
- В случаите, когато порядъкът съдържа само нули и мантисата също съдържа само нули, се счита, че е представена нулата.
- Когато порядъкът съдържа само единици, а мантисата само нули, се счита, че е представена безкрайност, положителна или отрицателна, в зависимост от стойността на знаковия бит.
- Когато порядъкът съдържа само нули, а мантисата е ненулева, то е представено денормализирано число.
- Когато порядъкът съдържа само единици, а мантисата е ненулева, представянето се счита, че не е на число, NaN (Not a Number) и се използва за сигнализиране на изключения.

Обработка на числа с плаваща точка. Събирането и изваждането на числа с плаваща точка обхваща следните стъпки:

1. Изравняване на порядъците (чрез съответното изместване на мантисите);
 2. Сумиране (изваждане) на мантисите;
 3. Нормализиране на резултата.
- Умножението (делението) на числа с плаваща точка обхваща следните стъпки:
1. Умножение (деление) на мантисите;
 2. Събиране (изваждане) на порядъците;
 3. Нормализиране на резултата.

Проблеми могат да възникнат в следните случаи:

- Препълване на полето на порядъка на резултата (положителната стойност на порядъка препълва полето на порядъка). В някои системи ситуацията се означава като $+\infty$ или $-\infty$.

- Незапълване на полето на порядъка на резултата (отрицателната стойност на порядъка е прекалено малка, за да бъде представена) и числото се представя като 0.

- Незапълване на мантисата. При изместването на мантисата с цел изравняване на порядъците, цифрите препълват решетката и излизат извън най-десния разряд. В този случай е необходимо закръгляване.

- Препълване на мантисата. При сумиране на мантиси с еднакви знаци може да възникне пренос от най-старшия бит. В този случай е необходимо изместване на мантисата на резултата.

Представяне на символи. Текстовата форма на данните е много удобна за човека, но не е удобна за дигитални системи, които използват бинарно представяне. Затова символите се представят чрез последователности от битове. Най-популярният код за представяне на символни данни е американският стандартен код за информационен обмен ASCII (American Standard Code for Information Interchange). При този код всеки символ се представя с последователност от 7 бита, при което е възможно използването на 128 комбинации. Една част от кодовете се използва за представяне на символите, останалата част се използва за кодиране на управляващи символи, например за управление на отпечатването на страница или за комуникационни процедури. Кодираните символи се съхраняват и се предават в 8-битов формат, като осмият бит може да съдържа 0 или контролен бит за проверка по четност. Друг код, който се използва за представяне на символи, е EBCDIC (Extended Binary Coded Decimal Interchange Code). Както ASCII кодът, така и кодът EBCDIC са съвместими с пакетираното десетично кодиране.

Представяне на логически данни. Информационните единици, които могат да се адресират индивидуално в компютъра като байт, дума, полудума и т.н., се третираат като цялостни информационни единици. Логическите операции в процесора, обаче, са побитови, т.е. операциите се извършват над битове с еднакви позиции в операндите. Следователно представянето на логическите данни в компютъра трябва да бъде бит-ориентирано, т.е. данните могат да приемат само две стойности: 1 (true) и 0 (false). Бит-ориентираното представяне на данните е необходимо също така при изместването на мантисата при нормализацията или изравняването на порядъците при числата с плаваща точка, при трансформирането на ASCII код в пакетиран десетичен код и др. Едни и същи данни могат да бъдат третирани като числови, логически или символни. Типът на данните се определя от операциите, които се извършват с тях.

2.3. СИСТЕМА ИНСТРУКЦИИ

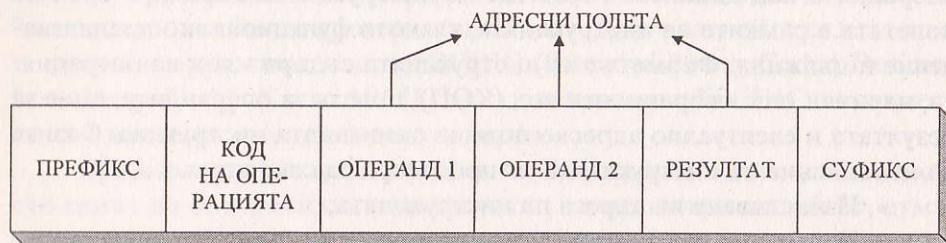
Формат на инструкциите. Системата инструкции е изключително важен архитектурен белег на всеки процесор. Компютърът изпълнява програма, която представлява наредена последователност от инструкции. Чрез системата инструкции програмистът управлява работата на компютърната система. Компютърните програми се съставят на езици от високо ниво, в които представянето на информационната обработка е във вид удобен за програмиста, но не и за компютъра като дигитална система. Затова компютърната програма се превежда на машинен език, т.е. в машинни инструкции. Машинните инструкции имат бинарно (двоично) представяне, т.е. те съдържат само нули и единици. Действието на процесора зависи от инструкцията, която той изпълнява. Множеството машинни инструкции, които даден процесор може да изпълнява, се нарича *система инструкции на процесора*. Системата инструкции на процесора трябва да бъде *функционално пълна*, т.е. всеки оператор от езиците за програмиране на високо ниво да може да бъде реализиран чрез машинните инструкции на процесора. Всяка машинна инструкция трябва да съдържа необходимата информация за процесора за изпълнението на операцията над данните. Форматът на инструкцията дефинира броя на полетата в рамките на инструкцията, тяхното функционално предназначение и дължина. Форматът на инструкцията съдържа код на операцията, наричан още операционен код (КОП), полета за операндите, поле за резултата и евентуално адресно поле за следващата инструкция. Фазите на изпълнение на инструкцията от процесора са следните:

- Изчисляване на адреса на инструкцията;
- Извличане на инструкцията от паметта и зареждането ѝ в регистъра на инструкцията (IR);
- Декодиране на операцията;
- Изчисляване на адресите на операндите;
- Четене на операндите от паметта;
- Изпълнение на операцията;
- Изчисляване на адреса на резултата;
- Запис на резултата.

Форматът на триадресна компютърна инструкция е показан на фиг. 2.7.

Полетата префикс (обикновено с дължина 1 байт) и суфикс служат за указване на специални операции. Така например префиксът ESCAPE указва операция на данни с формат плаваща точка, префиксът LOCK осигурява ексклузивно използване на общата памет в мултипроцесори, префиксът REP указва повторение на операцията върху последователните елементи на стринг и т.н.

При процесорите съществуват два варианта за указване на следващата инструкция: естествен ред на изпълнение на инструкциите и принудителен ред на изпълнение на инструкциите. Съвременните процесори масово използват естествения ред за изпълнение на инструкциите. При тях адресът на текущата изпълнявана от процесора инструкция се съдържа в специален адресен регистър, наречен програмен брояч. След приключване на изпълнението ѝ, програмният брояч автоматично увеличава съдържанието си, като по този адрес се извлича следващата инструкция, подлежаща на изпълнение. Следователно при естествен ред на изпълнение на инструкциите не е необходимо да се използва допълнително поле във формата на инструкцията за указване на адреса на следващата инструкция. В случаите, когато процесорите използват принудителен ред на изпълнение на инструкциите, последното поле във формата на инструкцията съдържа адреса на следващата инструкция. Съдържанието на това поле се прехвърля в програмния брояч и по този адрес се извлича от паметта следващата инструкция.



Фиг. 2.7. Триадресен формат на компютърна инструкция

Поради трудностите на програмиста да се справя с бинарното представяне на компютърните инструкции е прието те да се представят в символна форма. Кодовете на операциите се представят чрез съкращения, наричани мнемонични означения (ADD – за сумиране, SUB – за изважда-

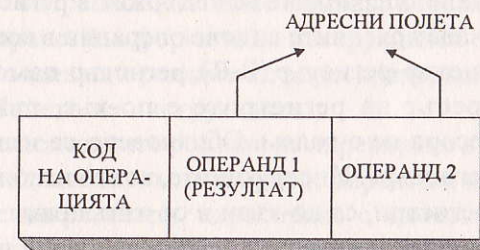
не, MUL – умножение, DIV – за деление и т.н.). Операндите също се представят в символна форма с имена на променливи (A, B, X, Y и др.). Така например инструкцията

ADD R1, A

показва, че съдържанието на клетка от паметта с адрес A трябва да се прибави към съдържанието на регистър R1.

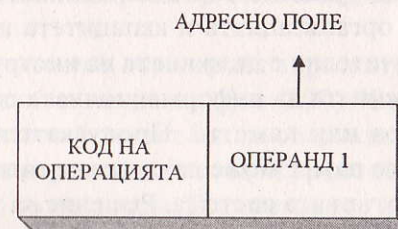
По този начин е възможно написването на програми на машинен език в символна форма, която се конвертира в бинарна форма (компютърни инструкции).

Триадресният формат на инструкциите не се използва в много процесори, тъй като изисква голяма дължина на инструкцията, за да съдържа трите адресни полета. При двадресен формат инструкцията съдържа две адресни полета, като едно от адресните полета трябва да съдържа както адреса на единия операнд, така и адреса на резултата (фиг. 2.8). Инструкциите с двадресен формат имат по-малка дължина, но предизвикват известно неудобство. Преди изпълнението на операцията е необходимо единият операнд, на чийто адрес ще бъде записан резултата, да бъде записан във временна клетка от паметта или временен регистър.



Фиг. 2.8. Двадресен формат на компютърната инструкция

В съвременните процесори се използва и едноадресният формат на инструкциите (фиг. 2.9). При този формат вторият адрес трябва да бъде



Фиг. 2.9. Едноадресен формат на инструкциите

зададен в неявен вид. Обикновено неявно заложеният адрес е на регистър в процесора, наречен акумулатор. Акумулаторът съдържа единия адрес на операнда и се използва за запис на резултата. При едноадресен формат на инструкциите дължината на програмите се увеличава, тъй като са необходими допълнителни инструкции, които да заредят операндите в акумулатора.

Броят на адресните полета във формата на инструкцията влияе съществено върху архитектурата на процесора. По-малък брой на адресните полета в инструкцията водят до по-примитивни инструкции, които от своя страна опростяват структурата на процесора. При едноадресни инструкции програмистът разполага само с един универсален регистър, акумулатора. При инструкции с повече адреси се използват множество универсални регистри. Това дава възможност някои операции да се изпълняват само с регистри. Достъпът до регистрите е много по-бърз от достъпа до паметта, така че изпълнението се ускорява. За да се постигне по-голяма гъвкавост и да се ускори изчислителният процес, съвременните процесори използват както дваадресни, така и триадресни инструкции.

Според това дали операндите се съдържат в регистри или в клетки в паметта се различават следните видове операции в компютърната система: операции регистър-регистър (R-R), регистър-памет (R-M) и памет-памет (M-M). Адресът на регистрите е по-къс, тъй като броят на регистрите в процесора не е голям. Обикновено се използват 4 бита за адресиране на 16 регистъра. Инструкциите, които използват за операнди съдържанието на регистри, са по-къси и се изпълняват по-бързо. В този аспект най-бързи са операциите регистър-регистър, а най-бавни са операциите памет-памет, тъй като изискват достъп до паметта. Операциите от типа регистър-памет заемат междинно място по бързодействие. Програмите, съдържащи операции регистър-регистър, са най-дълги, а програмите, съдържащи операции памет-памет, са най-компактни.

Дължината на инструкцията определя в значителна степен архитектурата на процесора, организацията и капацитета на паметта, шинната архитектура. Препоръчително е дължината на инструкцията да бъде равна или кратна на дължината на информационната единица, която се обменя между процесора или паметта. Пропускателната способност на тракта процесор-главна памет може да се окаже тясно място за производителността на компютърната система. Решение на този проблем е да се използват къси инструкции и кеш за инструкции. От друга страна трябва да се има предвид, че дължината на инструкцията трябва да бъде кратна

на дължината на числата с фиксирана точка, както и на дължината на символите (8 бита). Съвременните процесори допускат използването на инструкционни формати с различна дължина.

Дължината на инструкцията зависи от дължината на операционния код и дължините на адресните полета. Големият брой операции в системата инструкции изисква по-голямо поле за операционния код, което при фиксиран инструкционен формат води до скъсяване на адресните полета. Дължината на инструкцията зависи също така и от броя на операндите. Съвременните процесори реализират най-често операции с два операнда. Дължината на адресните полета зависят от режимите за адресация на данните в процесора. При явно задаване на режимите на адресация са необходими няколко бита във формата на инструкцията за указване на използвания режим за адресиране. Дължината на адресните полета зависи и от размера на адресния обхват на компютърната система, т.е. от дължината на физическия адрес в паметта.

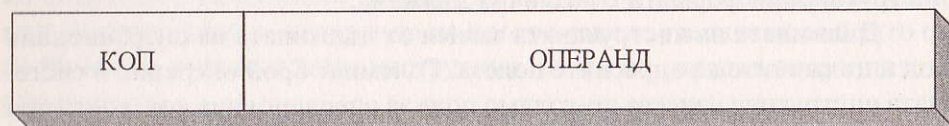
Режими на адресиране. Система инструкции, при която всеки операционен код може да бъде съвместен с произволен режим на адресиране, се нарича *ортогонална система инструкции*. При повечето процесори обаче, различните операционни кодове изискват използването на различни режими за адресиране. Във формата на инструкцията са необходими един или повече бита, задаващи режима на адресиране, който ще бъде използван при изпълнението на инструкцията.

Най-често използваните режими на адресиране в съвременните процесори са непосредствено, имплицитно, директно, индиректно адресиране, адресиране на регистри, адресиране с изместване (адресиране с индексирание, релативно адресиране, адресиране с базиране).

При режимите на адресиране се използва ефективен адрес. При системи без виртуална памет ефективният адрес представлява адрес в паметта или регистър. При компютърни системи с виртуална памет ефективният адрес представлява виртуален адрес или регистър. Преобразуването на виртуалния адрес във физически се реализира прозрачно за програмиста.

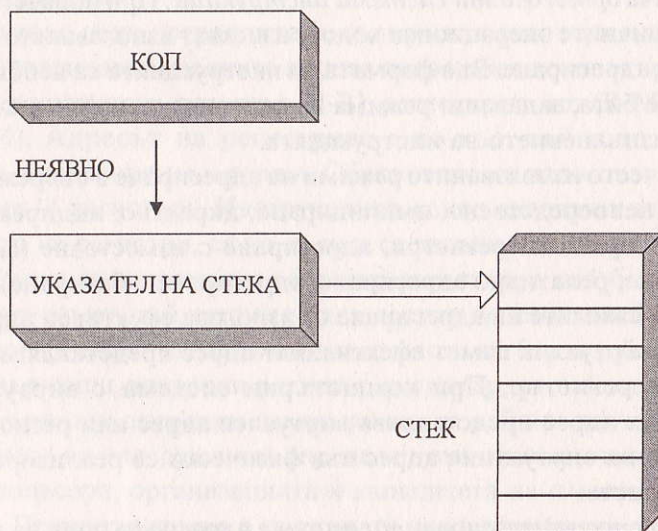
При *непосредственото адресиране* адресното поле на инструкцията съдържа стойността на операнда (фиг. 2.10). Числото е представено в допълнителен код, като най-левият бит на операнда съдържа неговия знак. Предимството на непосредственото адресиране е, че се спестява достъп до паметта за четене на операнда. Този режим се използва за задаване на константи или начални стойности на променливи. Негов недостатък е, че дължината на операнда се ограничава от дължината на адресното поле

на инструкцията, което при повечето процесори е по-малко от дължината на машинната дума.



Фиг. 2.10. Непосредствено адресиране

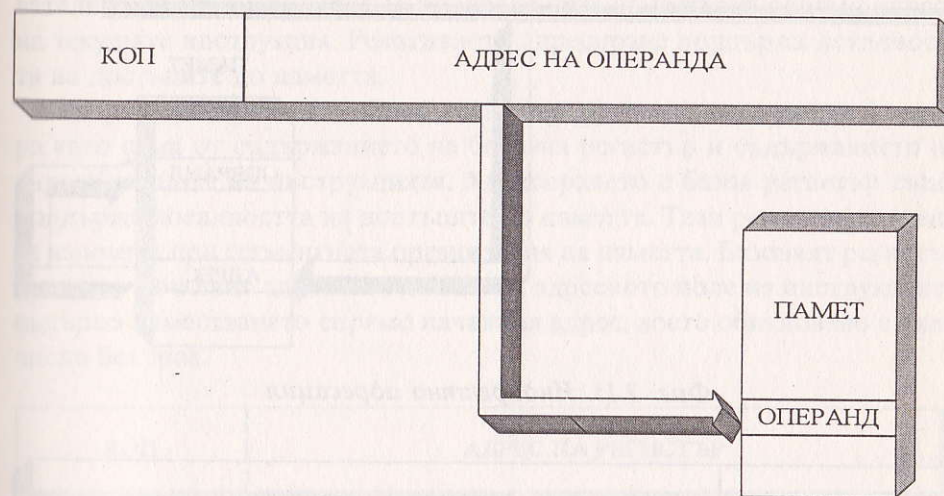
При имплицитното адресиране (фиг. 2.11) се използва стек. Стекът представлява резервиран блок от клетки в паметта. Той представлява опашка с дисциплина на обслужване „последен влязъл, пръв излязъл“ (LIFO – Last In First Out). Данните се записват на върха и се четат пак от върха на стека. Върхът на стека се адресира от регистър, който се нарича указател на стека (SP – Stack Pointer). При някои процесори двата елемента на върха на стека се съхраняват в регистрите на процесора, така че указателят на стека се обръща към третия елемент на стека.



Фиг. 2.11. Имплицитно адресиране

Директното адресиране (фиг. 2.12) е най-простият и бърз режим на адресиране на клетки в паметта. При него адресното поле на инструкцията съдържа ефективния адрес на операнда. Поради ограничения размер

на адресното поле директната адресация може да се използва за обръщане към ограничена начална област на паметта.

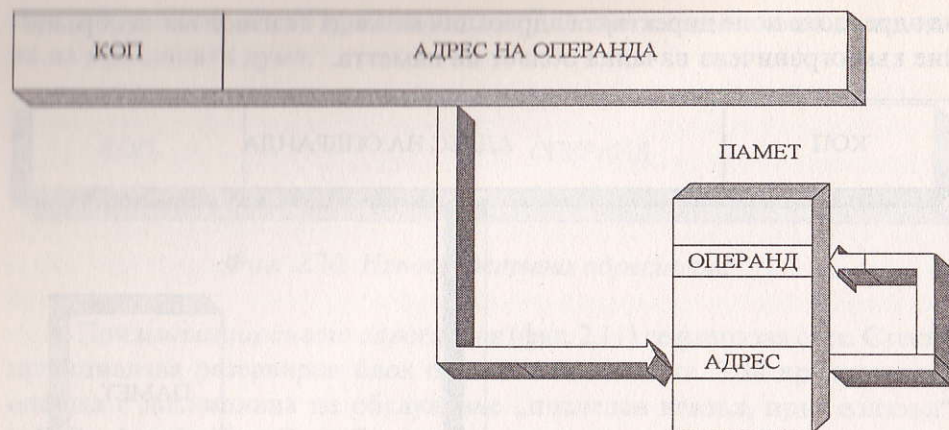


Фиг. 2.12. Директно адресиране

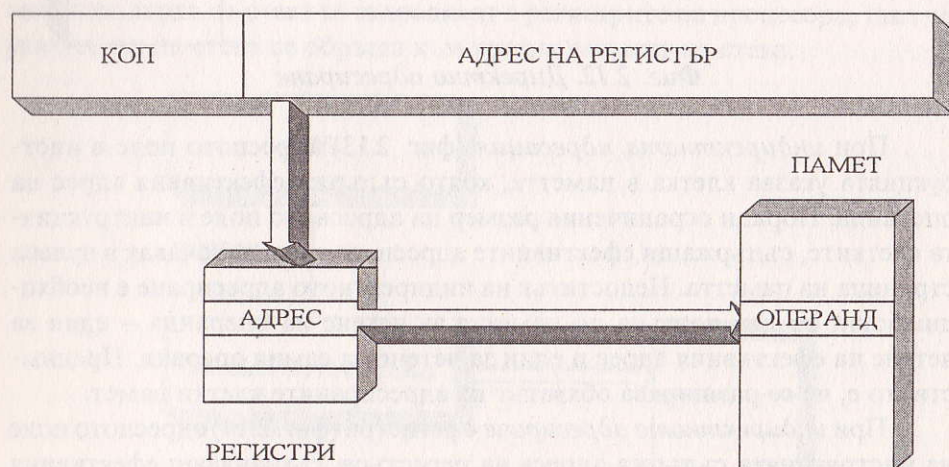
При индиректната адресация (фиг. 2.13) адресното поле в инструкцията указва клетка в паметта, която съдържа ефективния адрес на операнда. Поради ограничения размер на адресното поле в инструкцията клетките, съдържащи ефективните адреси, се съсредоточават в нулева страница на паметта. Недостатък на индиректното адресиране е необходимостта от два достъпа до паметта за четене на операнда – един за четене на ефективния адрес и един за четене на самия операнд. Предимството е, че се разширява обхватът на адресираните клетки памет.

При индиректното адресиране с регистри (фиг. 2.14) адресното поле на инструкцията съдържа адреса на регистъра, съхраняващ ефективния адрес на операнда. Този режим е по-бърз от индиректното адресиране чрез клетка от паметта, тъй като се спестява един достъп до паметта.

Адресирането на регистри (фиг. 2.15) е подобно на директното адресиране с тази разлика, че броят на регистрите в процесора не е голям и адресното поле в инструкцията може да съдържа няколко бита, обикновено 4. Използването на регистри за съхраняване на операндите е ефективно, когато тези операнди участват в множество операции или представляват междинни резултати. Съвременните процесори съдържат за тази цел универсални регистри, като определянето на операндите, които ще се съхраняват в регистри, се предоставя на компилатора.



Фиг. 2.13. Индиректна адресация



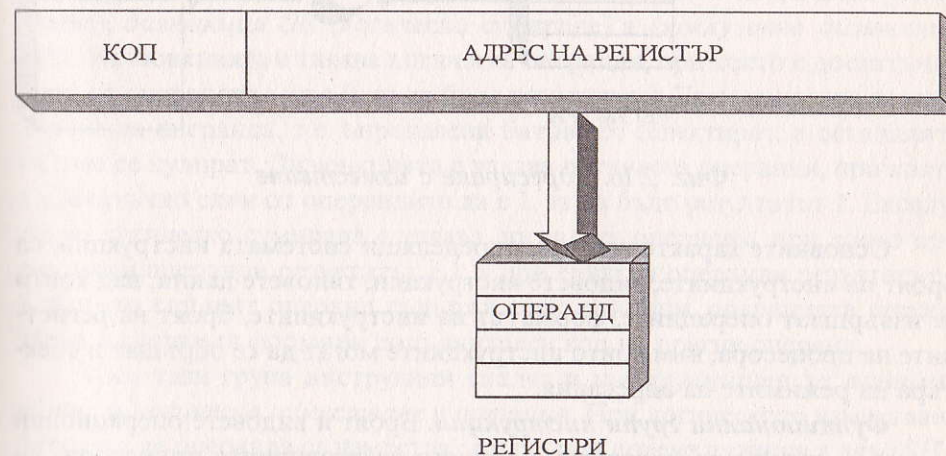
Фиг. 2.14. Индиректно адресиране с регистри

Адресирането с изместване (фиг. 2.16) съчетава възможностите на директното адресиране и индиректното адресиране с регистри. Този режим изисква две адресни полета, поне едното от които трябва да бъде зададено в явен вид. Ефективният адрес се формира като сума от адреса, зададен в адресното поле на инструкцията и съдържанието на определен регистър.

При релативното адресиране се използва съдържанието на програмния брояч (РС – Program Counter). Ефективният адрес се получава,

като адресът на текущата изпълнявана инструкция в програмния брояч се сумира със съдържанието на адресното поле на инструкцията. Следователно ефективният адрес се определя чрез изместване спрямо адреса на текущата инструкция. Релативното адресиране поддържа локалността на достъпите до паметта.

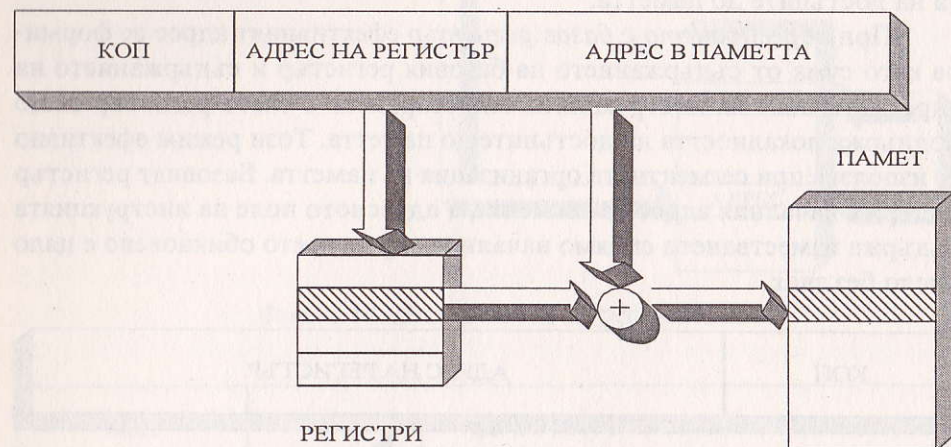
При адресирането с базов регистър ефективният адрес се формира като сума от съдържанието на базовия регистър и съдържанието на адресното поле на инструкцията. Адресирането с базов регистър също поддържа локалността на достъпите до паметта. Този режим ефективно се използва при сегментната организация на паметта. Базовият регистър съдържа началния адрес на сегмента, а адресното поле на инструкцията съдържа изместването спрямо началния адрес, което обикновено е цяло число без знак.



Фиг. 2.15. Адресиране на регистри

При адресирането с индексирание адресното поле в инструкцията съдържа адрес на клетка в паметта, а индексният регистър съдържа положително изместване спрямо този адрес. Ефективният адрес се формира като сума от съдържанието на адресното поле на инструкцията и съдържанието на индексния регистър. Този режим на адресиране осигурява ефективен механизъм за реализация на итеративни операции, поради което след всяко обръщане към индексния регистър е необходимо той да бъде инкрементиран (да се увеличи съдържанието му с 1) или декрементиран (да се намали съдържанието му с 1). При процесори, при които

това се реализира автоматично, режимът се нарича автоиндексиране. Когато модификацията на съдържанието на индексния регистър се осъществява след изпълнението на операцията, режимът се нарича *постиндексиране*, в противен случай – *преиндексиране*.



Фиг. 2.16. Адресиране с изместване

Основните характеристики, определящи системата инструкции, са броят на инструкциите, видовете инструкции, типовете данни, над които се извършват операциите, форматът на инструкциите, броят на регистрите на процесора, към които инструкциите могат да се обръщат и спектъра на режимите на адресация.

Функционални групи инструкции. Броят и видовете операционни кодове в системата инструкции варират за различните процесори, но съществуват общи функционални групи инструкции, които се поддържат от всеки процесор. Това са групите за трансфер на данни, за аритметични операции, за логически операции, за конвертиране на данните, вход/изход, групата инструкции за управление на системата и за предаване на управлението.

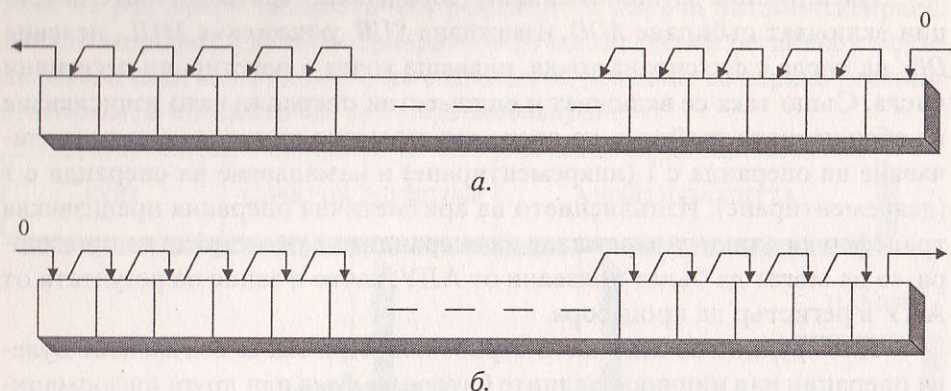
Инструкции за трансфер на данни. Източниците и дестинациите на трансфера на данни в компютърната система могат да бъдат регистрите на процесора, клетки в паметта, стека. Във всяка инструкция за трансфер на данни трябва да бъдат указани източникът и дестинацията при трансфера, както и дължината на обменяните данни. Така например *MOV* се използва за трансфер на данни между регистрите или регистрите и паметта, *PUSH* – за запис в стека, *POP* – за четене от стека и др.

Инструкции за аритметична обработка. Аритметичните операции включват събиране *ADD*, изваждане *SUB*, умножение *MUL*, деление *DIV* на числа с фиксирана точка, плаваща точка и пакетирани десетични числа. Също така се включват и едноместни операции като изчисляване на абсолютната стойност на операнда, промяна на неговия знак, увеличаване на операнда с 1 (инкрементиране) и намаляване на операнда с 1 (декрементиране). Изпълнението на аритметична операция предизвиква трансфер на данни за зареждане на операндите в регистрите на процесора, за да могат да бъдат ползвани от АЛУ, както и запис на резултата от АЛУ в регистър на процесора.

Инструкции за логическа обработка. При тях се изпълняват Булеви операции над индивидуалните битове на дума или друга информационна единица. Тази група включва *логическата инверсия NOT*, при която се инвертират битовете на операнда, *конюнкция AND* (логическо умножение), *дизюнкция OR* (логическо сумиране) и *ексклузивна дизюнкция XOR*. Конюнкцията е такава логическа операция, при която е достатъчно един от операндите да е 0, за да бъде резултатът 0. Тя се използва за маскиране на операнда, т.е. определени битове се селектират, а останалите битове се нулират. Дизюнкцията е такава логическа операция, при която е достатъчно един от операндите да е 1, за да бъде резултатът 1. Ексклузивно логическо сумиране е такава логическа операция, при която при различни операнди резултатът е 1, а при еднакви операнди резултатът е 0. Когато единият операнд съдържа само единици, операцията ексклузивна дизюнкция формира допълнителен код на другия операнд.

Към тази група инструкции спадат и *инструкциите за аритметично и логическо изместване и ротация*. При логическото изместване битовете на операнда се изместват с една или повече позиция в ляво *SHL* или в дясно *SHR*, при което в единия край битовете се изгубват, а в другия край се вкарват нули (фиг. 2.17). Логическите операции за изместване се използват за изолиране на полета от битове в думата. Излишните битове се изместват извън рамките на разрядната решетка.

При *аритметичното изместване* данните се третират като цяло число със знак и знаковият бит не участва в изместването (фиг. 2.18). При аритметичното изместване вдясно *SAR* освободените позиции надясно от знаковия бит приемат неговата стойност. Аритметичното изместване надясно с 1 разряд на числа в допълнителен код е еквивалентно на деление на 2, а аритметичното изместване вляво *SAL* с 1 разряд е еквивалентно на умножение с 2 при положение, че не възниква препълване или незапълване.

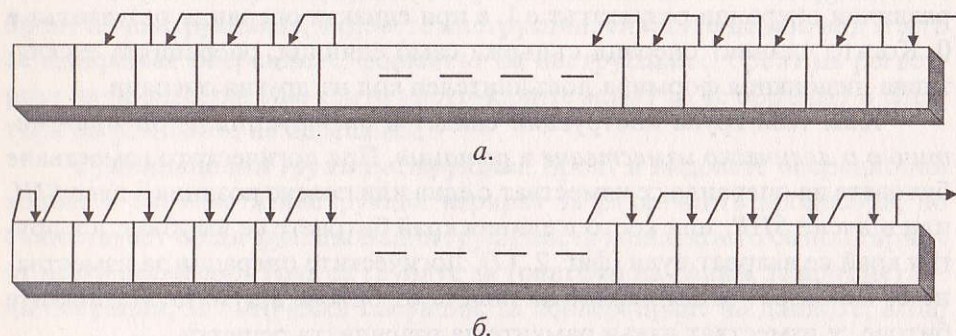


Фиг. 2.17. Логическо изместване

а. логическо изместване в ляво, б. логическо изместване в дясно

При операциите за циклично изместване (ротация) в ляво *ROL* и в дясно *ROR* се запазват всички битове на операндите (фиг. 2.19).

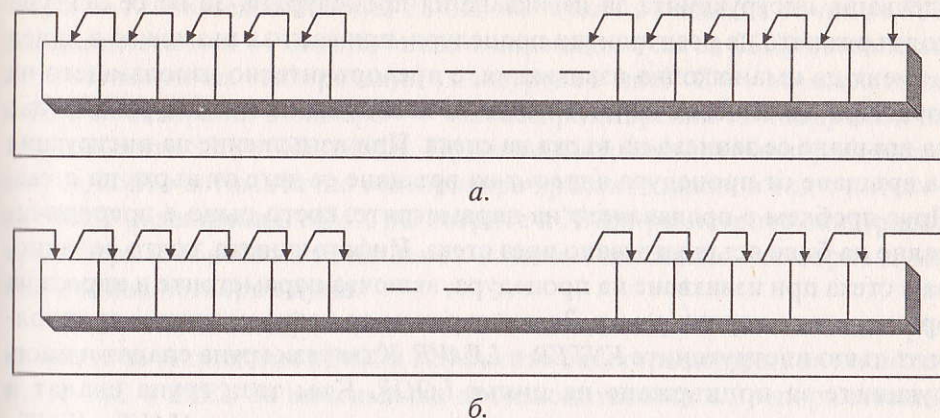
Инструкциите за конвертиране на данните служат за форматиране на данните или за промяна на техния формат – например конвертирането на ASCII код в EBCDIC или трансформирането на десетично пакетирани код в двоичен код.



Фиг. 2.18. Аритметично изместване

а. аритметично изместване в ляво, б. аритметично изместване в дясно

Инструкциите за управление на системата са привилегирани инструкции, които се изпълняват в режим супервайзор на централния процесор от операционната система. Такива са инструкциите за четене или зареждане на управляващите регистри, промяна на ключ за защита на паметта и др.



Фиг. 2.19. Ротация (циклично изместване)

а. ротация в ляво, б. ротация в дясно

Групата инструкции за предаване на управлението променят естествения ред на изпълнение на инструкциите в компютърната програма, като актуализират съдържанието на програмния брояч с адреса на инструкцията, подлежаща на изпълнение. Тази група включва инструкциите за преход, инструкциите за извикване на и връщане от процедура, примитиви за цикъл, инструкции за програмно прекъсване. Инструкциите за безусловен преход *JMP* съдържат в адресното си поле адреса на следващата инструкция, която трябва да бъде изпълнена. Те предизвикват зареждане на указания адрес на инструкцията в програмния брояч. При инструкциите за условен преход се тества кодът на условията *CC* (Condition Code), който включва флаг, отразяващ типа на резултата на определена операция. Аритметичните флагове, които се използват, са пренос *C* (carry), четност *P* (parity), междутетраден пренос *A* (auxiliary carry), нулев резултат *Z* (zero), знак на резултата *S* (sign), препълване *O* (overflow). При положение че условието е изпълнено, програмният брояч се зарежда с адреса на инструкцията, указана в адресното поле. В противен случай се запазва естественият ред на изпълнението на инструкциите, т.е. изпълнява се следващата инструкция. Инструкциите за извикване на процедури *CALL* преустановяват изпълнението на главната програма и стартират изпълнението на процедурата чрез обръщане към началния адрес на процедурата, зададен в адресното поле на инструкцията за извикване. Инструкциите за връщане от процедура *RETURN* предизвикват изпълнение на инструкцията от извикващата програма, непосредствено

следваща инструкцията за извикване на процедурата. За да се осигури поддържането на реентрантни процедури, при които е възможно в даден момент да има няколко извиквания, е препоръчително използването на стек. При изпълнение на инструкция за извикване на процедура, адресът на връщане се записва на върха на стека. При изпълнение на инструкция за връщане от процедура адресът на връщане се чете от върха на стека. Друг проблем е предаването на параметрите, което също е препоръчително да бъде осъществявано през стека. Информацията, която се записва в стека при извикване на процедура, включва параметрите и адреса на връщане и се нарича рамка. За поддържане на този механизъм се използват също инструкциите *ENTER* и *LEAVE*. Към тази група спадат и инструкциите за поддържане на цикъл *LOOP*. Към тази група спадат и инструкциите за спиране на изпълнението на програмата *HALT* и *WAIT*.

Инструкции за символна обработка. Стрингът се дефинира като наредена последователност от символи. Операциите, които се поддържат, са трансфер на стринг *MOVS*, сравнение на стрингове *CMPS*, търсене на шаблон, търсене на шаблон с канонично заместване, конкатенация (присъединяване на стрингове), въвеждане на стринг *INS*, извеждане на стринг *OUTS*, както и префикс *REP* (няколко разновидности) за повторение на указаната операция последователно за всички елементи на стринга.

Инструкции за вход/изход поддържат въвеждането и извеждането на информация. Тази група съдържа операции за вход (*IN*) и операции за изход (*OUT*).

Инструкции за MMX технология и технология с потокови SIMD (SIMD – Single Instruction flow Multiple Data flows) **разширения.** Поддържат се от процесорите Intel® Pentium™ 4, Intel® Xeon™ и Pentium™ М. Това са технологии, ориентирани към паралелна обработка при единичен поток инструкции над множество потоци данни, осигуряващи ефективна разработка на мултимедийни приложения, обработка на сигнали и моделиране. Тези приложения включват алгоритми за компресиране, филтриране и разпознаване на говор, програми за видео дисплей, 3D графика (геометрия), алгоритми за обработка на изображения и видео, пространствено (3D) аудио, моделиране, криптографски алгоритми. SIMD ориентацията на SSE, SSE2 и MMX технологиите изискват нови методи за кодиране на алгоритмите, единият от които е векторизацията. Векторизацията представлява трансформиране на последователния скаларен код в код, който може да се изпълнява паралелно, като се използва заложеният SIMD архитектурен паралелизъм. На векторизация подлежат цикли, обработаващи масиви от 8- или 16-битови цели

числа за MMX технологията, 32-битови числа с плаваща точка за SSE технологията или 64-битови числа с плаваща точка за SSE2 технологията. Автоматична векторизация е осигурена в Intel® C++ компилатора, който се базира на нови C/C++ езикови разширения за поддръжка на технологиите SSE2, SSE, MMX. При него е вложен оптимизиращ механизъм, който автоматично векторизира простите цикли в код с потокови SIMD разширения. MMX регистрите и XMM регистрите осигуряват трансфера и обработката на данни с голяма дължина – четворна дума или двойна четворна дума.

За тази цел са дефинирани нови типове данни: 64-битови пакетирани данни за технология MMX и 128-битови пакетирани данни за технологиите SSE и SSE2. За максимална производителност е препоръчително данните да бъдат разполагани на 16-байтови граници в паметта. За MMX технологията са осигурени инструкции за пакетизиране и разпакетиране на данните в MMX регистрите. Така например инструкцията за екстракция на дума *pextrw* премества дума от MMX регистър в младшата половина на 32-битов регистър за цели числа. Инструкцията за вмъкване на дума *pinsrw* зарежда дума от младшата половина на 32-битов MMX регистър за цели числа или от паметта в MMX регистър.

При тази група инструкции се осигурява реализацията на често използвани операции по разместване на данните: *Broadcast* (предаване на една дума към всички думи), *Swap* (разменяне на съдържанието на две думи) и *Reverse* (обръщане на реда на подреждането на думите).

Осигурени са две допълнителни инструкции *movq2dq* и *movdq2q* за обмен на данни между 64-битови MMX регистри към 128-битови SIMD регистри. Добавени са инструкции за конвертиране на формата на данните, инструкции за определяне на максималната (*pmaxsw*) и минималната (*pminsw*) дума от 4 думи със знак в SIMD регистрите, инструкции *pavgb* за изчисляване на средната стойност на байтове (думи), съдържащи се в два SIMD регистъра или в SIMD регистър и клетка от паметта.

Характерна особеност на тази група инструкции е използването на т.нар. аритметика с насищане. Групата инструкции включва инструкции за умножение на комплексни числа *pmaddwd*, при които се използват MMX регистрите и реалните и имагинерните числа са 16-битови, инструкции за умножение на пакетирани 32-битови числа без знак *pmuludq* (версии за 64 бита и за 128 бита), сумиране/изваждане на 64-битови числа с и без знак *paddq/psubq*, инструкции за изместване в ляво/дясно на 128 битови числа по зададен брой байтове *pslldq/psrldq*, при които изпразнените млади/старши байтове се запълват с нули.

При приложенията, свързани с 3D-графика, се препоръчва вертикалната организация на изчисленията, тъй като при нея се осигурява 100%-но използване на SIMD-регистрите.

Оптимизацията на използването на тракта процесор-памет се поддържа от инструкции за трансфер на цяла позиция в кеша (напр. *movdq*).

Разширенията в системата инструкции за SSE, SSE2 и MMX технологиите осигуряват оптимизация на използването на кеша. Те се отразяват особено благоприятно за редуциране на трафика процесор-памет при видео приложенията.

SIMD разширенията включват няколко инструкции за предварително извличане на инструкции: *prefetchnta* (прехвърляне в кеш L2) и *prefetch0, 1, 2* – прехвърляне в кешовете на всички нива на Pentium™ 4 до ниво 2. Поради критичното влияние на шината към паметта като критичен ресурс използването на подходящи комбинации от инструкциите *prefetchnta* (non-temporal) може да съкратят транзакциите до паметта чрез оптимизация на съдържанието на кеша от второ ниво L2 и намаляване на транзакциите за обратен запис.

2.4. КОМПЮТРИ СЪС СЪКРАТЕН НАБОР ИНСТРУКЦИИ (RISC)

Изключително бързите темпове на развитието на технологията доведоха до значително спадане на цените на компютърния хардуер. Едновременно с това цените на компютърния софтуер нарастнаха. Появиха се мощни и сложни езици за програмиране от високо ниво, поддържащи концепциите на структурното или обектно-ориентираното програмиране. Езиците за програмиране от високо ниво осигуряват езикови конструкции за представяне на изчислителния процес във форма близка до човека, докато процесорът изпълнява строго дефиниран набор от машинни команди. Предназначението на компилатора е да генерира последователност от машинни инструкции за изпълнението на всеки оператор от език на високо ниво.

Задачата на компилатора се опростява, ако съществуват машинни инструкции, които да са близки до операторите в езиците за програмиране от високо ниво. Следователно се стимулира развитието на архитектури с големи набори инструкции, широк диапазон на режими за адресиране, както и хардуерна реализация на оператори от езиците от високо ниво.

Тези тенденции в развитието на компютърните архитектури включват разработването на процесори с комплексен набор инструкции CISC (Complex Instruction Set Computers). Основните цели на това направление са увеличаването на системната производителност и опростяването на компилаторите, с което да се осигури по-ефективна поддръжка на езиците от високо ниво. За подобряване на ефективността на изчислителния процес сложните последователности от операции се реализират чрез микрокод.

Алтернативната тенденция са компютрите със съкратен набор инструкции RISC (Reduced Instruction Set Computers). Характерните особености на тези архитектури са ограничен набор инструкции с фиксиран формат, малко на брой прости режими на адресиране, използване на големи регистрови файлове или компилатор, оптимизиращ използването на регистрите, както и оптимално използване на инструкционни конвейери.

Съществените белези на системата инструкции на RISC процесорите са:

- Изпълнението на една инструкция заема един машинен цикъл.
- Фиксиран размер на инструкцията.
- Прости инструкционни формати с фиксирани дължини на полетата.
- Мнозинството операции използват операции регистър-регистър (R-R).
- Малък брой режими на адресиране.
- Не се използва индиректно адресиране, т.е. не се допуска допълнителен цикъл за достъп до паметта за получаване на адреса на операнда.
- Операндите и резултатите при аритметичните операции задължително се съхраняват в регистри.
- За дадена инструкция само един операнд може да се съдържа в паметта.
- Разположението на инструкциите и на данните в паметта е съобразено с границите на машинната дума и те се съдържат в рамките на една страница.

Фиксираните полета в рамките на формата на инструкцията дават възможност да се съвмести декодирането на инструкцията с четенето на операндите от регистрите, а освен това опростяват управляващото устройство. Почти всички RISC инструкции използват просто адресиране на регистри. Могат да се включат няколко допълнителни режима като

изместване и релативно адресиране. По-сложните режими на адресиране могат да бъдат синтезирани чрез софтуера от по-простите такива.

Броят на обръщенията към паметта е редуциран за сметка на обръщенията към регистрите, което води до значително повишаване на бързодействието. Предвид на локалността на тези обръщения е целесъобразно използването на разширен регистров файл.

Използване на голям регистров файл. Основната цел на RISC архитектурите е високото бързодействие, така че изборът на голям регистров файл се основава на факта, че регистрите са най-бързата памет в компютърната система. Регистровият файл има малък капацитет и е вграден в интегралната схема на микропроцесора. Адресите на регистрите са много по-къси от тези на кеша и основната памет. Следователно необходима е стратегия, осигуряваща наличието на най-често използваните операнди в регистрите на процесора. Възможни са два подхода – софтуерен и хардуерен. При софтуерния подход се разчита на компилатора да осигури оптимално използване на регистрите. Компилаторът разполага в регистрите тези променливи, които се използват най-интензивно в даден интервал от време. Този подход изисква прецизни алгоритми за анализ на програмата. Хардуерният подход изисква вграждането на повече регистри в чипа на процесора, така че повече променливи да се съхраняват в регистрите за по-продължително време.

Регистрови прозорци. Най-често операндите са локални скалари, така че е разумно те да се съхраняват в регистровия файл, като малък брой регистри се използват за глобалните променливи.

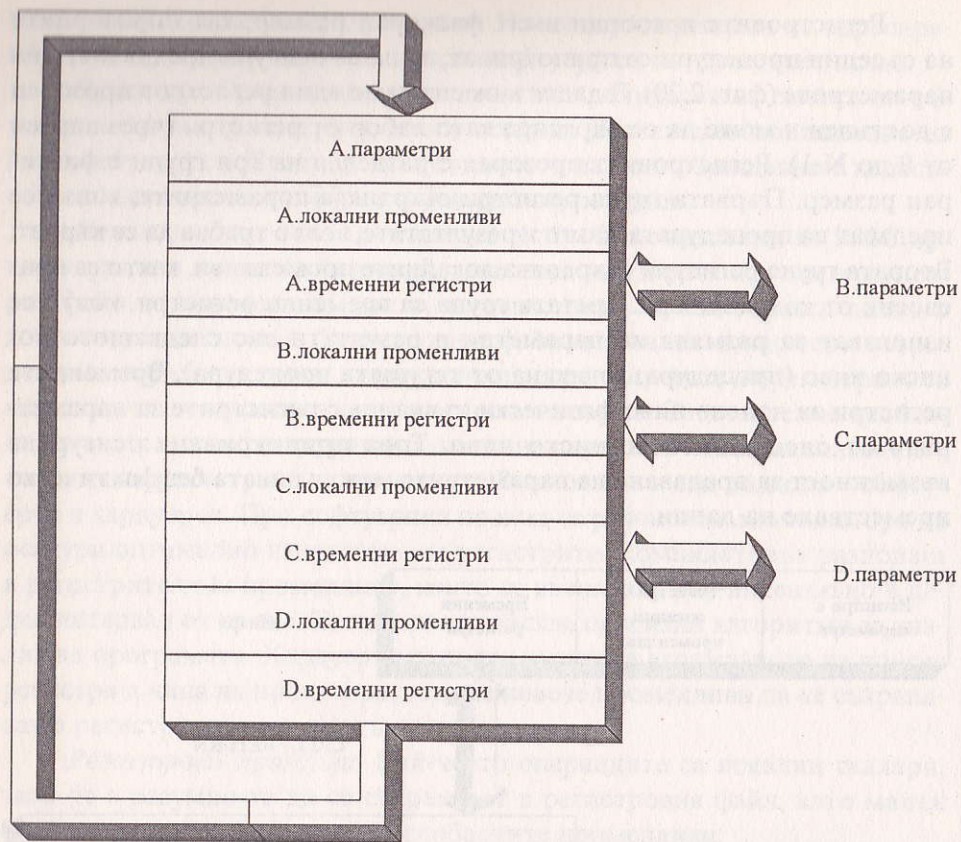
Особен интерес представляват операциите, които се изпълняват при извикване и връщане от процедура. При всяко извикване на процедура, локалните променливи трябва да се прехвърлят от регистрите в паметта, за да могат тези регистри да бъдат използвани от извиканата процедура. Освен това параметрите трябва да бъдат прехвърлени на извиканата процедура. При връщане от процедура променливите на извикващата програма трябва да бъдат възстановени обратно в регистрите и резултатите трябва да бъдат върнати на извикващата програма. При това трябва да се има предвид, че обикновено процедурите използват само няколко параметъра и локални променливи. От друга страна, дълбочината на извикване на процедурите варира в малък диапазон. Това мотивира използването на множество малки регистрови файлове, всеки един от които е предназначен да обслужва една процедура. Вместо да се съхранява съдържанието на регистрите в паметта, извикването на процедура автоматически прекъсва процесора да използва съответен регистров прозорец.

Регистровите прозорци имат фиксиран размер, като прозорците на съседни процедури се припокриват, за да се осигури предаването на параметрите (фиг. 2.20). В даден момент само един регистров прозорец е достъпен и може да се адресира като набор от регистри (чрез адреси от 0 до N-1). Регистровият прозорец е разделен на три групи с фиксиран размер. Първата група регистри съхранява параметрите, които се предават на процедурата, както и резултатите, които трябва да се върнат. Втората група регистри съхранява локалните променливи, както са присвоени от компилатора. Третата група са временни регистри, които се използват за размяна на параметри и резултати със следващото по-ниско ниво (процедура, извикана от текущата процедура). Временните регистри за дадено ниво физически съвпадат с регистрите за параметрите от следващото по-ниско ниво. Това припокриване осигурява възможност за предаване на параметрите между нивата без фактическо преместване на данни.



Фиг. 2.20. Припокриване на регистрови прозорци

Регистровите прозорци се използват за последните извикани процедури. По-старите извиквания трябва да бъдат съхранени в паметта и по-късно възстановени, когато нивото на влагане намалее. В този случай фактическата организация на регистровия файл е кръгов буфер с припокриващи се прозорци. На фиг. 2.21 е показан кръгов буфер с 4 прозореца. Буферът е запълнен до четвърто ниво. Указателят на активния прозорец указва прозореца на текущата активна процедура. Обръщенията към регистри от машинните инструкции използват този указател за определяне на физическия регистър.



Фиг. 2.21. Кръгов буфер с припокриващи се прозорци.

Регистров файл с N прозореца може да обслужва $N-1$ извиквания на процедури. Практически най-често се използват 16 прозореца с 32-битови регистри.

Схемата с регистровите прозорци не е ефективна за съхраняване на глобалните променливи. Възможни са два подхода.

При първия подход променливите, декларирани като глобални в програмата на език от високо ниво, се съхраняват в клетки от паметта, определени от компилатора. Всички машинни инструкции, които използват тези променливи като операнди, ще ги четат от паметта. Този подход не е ефективен при често използвани глобални променливи.

Вторият подход е да се осигури множество глобални регистри в рамките на процесора. Броят на глобалните регистри е фиксиран и те

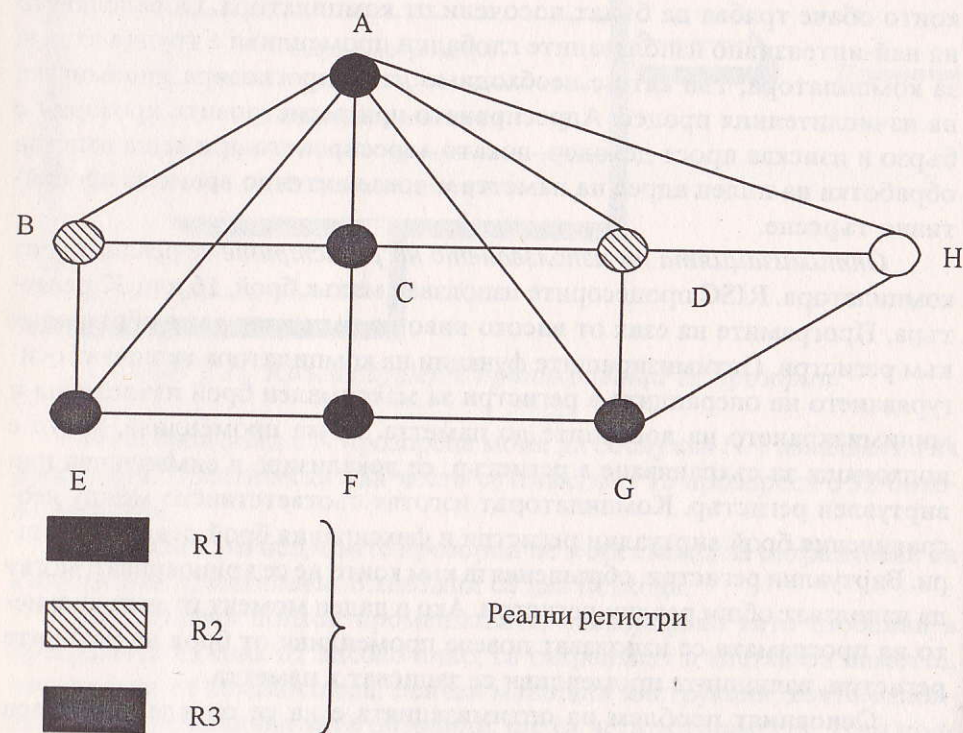
трябва да са достъпни за всички процедури. Компилаторът трябва да определи кои глобални променливи ще се съхраняват в тези регистри.

Регистровият файл, организиран като множество прозорци, функционира като бърз малък буфер за съхраняване на най-често използваните променливи. Алтернативата е за тази цел да се използва кеш. Регистровите прозорци съдържат всички локални променливи на последните $N-1$ извикани процедури, докато кешът съдържа множество от последните използвани локални променливи. Кешът може да съдържа както локалните, така и глобалните променливи. В общия случай броят на глобалните променливи е голям, но само малка част от тях се използват интензивно. Динамиката на кеша (стратегията за обновяване на информацията в кеша) му позволява да открие кои са тези променливи и да ги задържи. Ако регистровият файл има глобални регистри, те също могат да съхраняват най-често използваните глобални променливи, които обаче трябва да бъдат посочени от компилатора. Определянето на най-интензивно използваните глобални променливи е трудна задача за компилатора, тъй като е необходимо да се прогнозира динамиката на изчислителния процес. Адресирането при регистровите прозорци е бързо и изисква прост декодер, докато адресирането при кеша изисква обработка на пълен адрес на паметта и допълнително време за асоциативно търсене.

Оптимизацията на използването на регистрите се реализира от компилатора. RISC процесорите използват малък брой, 16 или 32 регистра. Програмите на език от високо ниво не съдържат явни обръщания към регистри. Оптимизиращите функции на компилатора включват осигуряването на операндите в регистри за максимален брой изчисления и минимизирането на достъпите до паметта. Всяка променлива, която е подходяща за съхраняване в регистър, се локализира в символичен или виртуален регистър. Компилаторът изготвя съответствието между неограничения брой виртуални регистри и фиксирания брой реални регистри. Виртуални регистри, обръщанията към които не се припокриват, могат да използват общи реални регистри. Ако в даден момент от изпълнението на програмата се използват повече променливи от броя на реалните регистри, излишните променливи се записват в паметта.

Основният проблем на оптимизацията е да се определи в даден момент от изпълнението на програмата кои променливи да се съхраняват в регистрите. Най-често използваната техника при RISC компилаторите е оценяване на графите. Програмата се анализира, като се конструира граф на зависимостите между регистрите. Възлите на графа

представят виртуалните регистри. Ако два виртуални регистъра са активни при изпълнението на даден програмен сегмент, представящите ги възли се съединяват с дъга. Графът се оцветява с n цвята, като n е броят на реалните регистри. Съседните възли задължително се оцветяват с различни цветове. Възли с еднакъв цвят могат да използват еднакви реални регистри. Възлите, които не могат да бъдат оцветени, използват клетки от паметта. За тяхното използване е необходимо да се използват операции LOAD и STORE. На фиг. 2.22 е показан прост пример за оцветяване на графи. В даден момент от изпълнението на програмата се използват виртуалните регистри A, B, C, D, E, F, G, H при налични три реални физически регистъра R1, R2, R3 означени съответно с черен, червен и син цвят. Съседните върхове са оцветени с различни цветове от наличните три цвята. За възел H няма свободен цвят, така че променливата, която той представя, се записва в паметта.



A, B, C, D, E, F, G, H – виртуални регистри

Фиг. 2.22. Оцветяване на графи

Оптимизация на конвейеризацията. При RISC процесорите уплътняването на инструкционния конвейер при инструкции за преход и инструкции, зависими по данни, се основава на пренареждане на инструкционния поток от компилатора, като се използват методът на отложен преход и методът на отложеното зареждане (при инструкции LOAD). Планирането на инструкциите за конвейера и динамичното преразпределяне на регистрите трябва да се разглеждат и прилагат в тясно взаимодействие за постигане на висока ефективност на системата.

Следователно RISC архитектурите могат да станат по-ефективни, като се включват елементи от CISC архитектурите и обратно, CISC архитектурите могат да станат по-ефективни, като се включват елементи от RISC архитектурите. Тези тенденции се наблюдават при съвременните широко разпространени комерсиални системи. Например системите, основани на Pentium™, нямат „чиста“ CISC архитектура и имат вложени RISC елементи.

2.5. РЕГИСТРИ НА ПРОЦЕСОРА

Централният процесор е отговорен за обработката на данните и за управлението на системата, което определя и основните му компоненти: АЛУ и УУ. Компонентите на процесора са свързани чрез вътрешно-системна шина, чрез която се осъществява трансфера на данни и инструкции към и от процесора и се управлява работата на АЛУ. АЛУ получава операндите си и записва резултата в регистрите на процесора. Освен това регистрите се използват за формиране на адреса и съхраняване на информацията, която се получава от или изпраща към паметта. За управление на системата процесорът използва т.нар. управляващи регистри и регистри за състоянието.

Централният процесор съдържа регистри, които са *програмно достъпни* (от програмите на машинен език) и които формират т.нар. *програмен модел на процесора*. Тези регистри при съвременните микропроцесори могат да бъдат организирани в следните функционални групи: универсални регистри (регистри с общо предназначение), регистри за данни, адресни регистри, регистри за настройка на програмите, регистри за числа с плаваща точка, регистри за мултимедия, регистри за SIMD изчисления, флагов регистър. На фиг. 2.23 е представена регистровата структура на съвременен процесор. Универсалните регистри са акумулатор (EAX – разширен акумулатор), базов регистър (EBX – разширен),

УНИВЕРСАЛНИ РЕГИСТРИ

АКУМУЛАТОР	EAX
БАЗОВ РЕГИСТЪР	EBX
БРОЯЧ	ECX
РЕГИСТЪР ЗА ДАННИ	EDX

УКАЗАТЕЛИ И ИНДЕКСНИ РЕГИСТРИ

УКАЗАТЕЛ НА СТЕКА	SP
УКАЗАТЕЛ НА БАЗАТА	BP
ИНДЕКС НА ИЗТОЧНИКА	SI
ИНДЕКС НА ДЕСТИНАЦИЯТА	DI

СЕГМЕНТНИ РЕГИСТРИ

ПРОГРАМЕН СЕГМЕНТ	CS
СЕГМЕНТ ЗА ДАННИ	DS
ДОПЪЛНИТЕЛЕН СЕГМЕНТ	ES
СТЕКОВ СЕГМЕНТ	SS

РЕГИСТРИ ЗА ЧИСЛА С ПЛАВАЩА ТОЧКА

FPR7
FPR6
FPR5
FPR4
FPR3
FPR2
FPR1
FPR0

РЕГИСТРИ ЗА МУЛТИМЕДИЯ

MMX7
MMX6
MMX5
MMX4
MMX3
MMX2
MMX1
MMX0

РЕГИСТРИ ЗА SIMD ОБРАБОТКА

ПРОГРАМЕН БРОЯЧ	PC		
РЕГИСТЪР НА ИНСТРУКЦИЯТА	IR		
ФЛАГОВ РЕГИСТЪР	CCR		
УПРАВЛЯВАЩИ РЕГИСТРИ			
CR0	CR1	CR2	CR3

XMM7
XMM6
XMM5
XMM4
XMM3
XMM2
XMM1
XMM0

Фиг. 2.23. Примерна регистрова структура на процесор

регистър-брояч (ECX – разширен), регистър за данни (EDX – разширен). В някои случаи универсалните регистри могат да се използват освен за съхраняване на данни и при изчисляването на адреса при по-сложните режими на адресация. Регистрите за данни служат за съхраняване на операнди и резултати на операции. Адресните регистри могат да бъдат универсални или специализирани, т.е. да се предназначени за определени режими на адресация. Сегментните регистри съдържат сегментния селектор (програмно достъпен) и сегментния дескриптор (прозрачен за потребителя) при поддържана сегментна организация на паметта. Зареждането на сегментния селектор води до автоматичното зареждане на сегментния дескриптор.

Според типа на сегмента те могат да бъдат: сегментен регистър на кода CS (на програмния сегмент), сегментен регистър за данни DS, сегментен регистър за стрингове, сегментен регистър на стека и др. Индексните регистри (SI – индекс на източника, DI – индекс на дестинацията) се използват за режима на адресиране с индексирание и обикновено е осигурена автоиндексация. Указателят на стека (SP) съдържа адреса на върха на стека, който е организиран в паметта. Указателят на базата (BP) се използва при адресиране с базиране. В съвременните процесори са вложени регистри за числа с плаваща точка, регистри за мултимедия MMX и регистри за SIMD изчисления XMM, регистри за настройка на програмите DR.

Флаговият регистър е частично достъпен за програмиста и съдържа кодовете на условията (CC), наричани още флагове. Това са еднобитови флагове, които се установяват в 1 или се нулират от процесора в зависимост от резултата на извършената операция. Флаговете могат да се четат от програмата, но не могат да бъдат променени от нея.

При прекъсване или при извикване на подпрограма е необходимо програмно достъпните регистри да се съхранят в стека или в паметта. Аналогично това се налага при изпълнението на инструкциите call и return. При някои процесори това се извършва автоматично, при други се налага програмистът да осигурява съхраняването на съдържанието на програмно достъпните регистри и впоследствие тяхното възстановяване от стека или паметта.

За изпълнението на инструкциите от особена важност са групата регистри, която осигурява информационния трансфер между процесора и паметта. Тази група включва програмния брояч (PC), съдържащ адреса на следващата инструкция, която трябва да бъде извлечена от паметта, регистъра на инструкцията (IR), съдържащ последната извлечена

инструкция, адресния регистър на паметта (MAR), който съдържа адреса, по който се извършва обръщането към паметта, буферния регистър на паметта (MBR), който съдържа данните за четене от или запис в паметта. Обикновено адресният регистър на паметта (MAR) директно е свързан към адресната шина, а буферният регистър на паметта (MBR) директно се свързва към шината за данни. Програмно достъпните регистри обменят информация с буферния регистър на паметта. При някои процесори АЛУ има директен достъп до буферния регистър на паметта и регистри за данни. При други процесори са осигурени допълнителни буферни регистри, които служат като входни и изходни регистри за АЛУ, а също така обменят данни с буферния регистър на паметта и програмно достъпните регистри.

Управляващите регистри и регистрите за състоянието се използват от УУ за определяне на режима на работа на процесора и на цялата система, а също така те се използват при изпълнението на привилегировани инструкции от операционната система за управление на изпълнението на програмата. Обикновено управляващите регистри и регистрите за състоянието или поне повечето от тях са прозрачни за потребителя. Някои от тях са достъпни в режим супервайзор. Регистърът PSW (дума за състоянието на програмата) съдържа информация за състоянието на процесора и обикновено включва и флаговете. Освен аритметичните флагове се съдържат и флаг за разрешаване/забраняване на прекъсванията, за режима на работа на процесора (режим супервайзор/потребител), текущото ниво на приоритета на обслужваните прекъсвания, ниво на привилегированост на входно-изходните операции, флаг за вложени задачи и др. Процесорите разполагат с множество управляващи регистри CR за задаване на режимите му на работа или индицират състоянието му при изпълнението на дадена задача. Съвременните процесори поддържат многозадачен режим. Управляващите регистри CR поддържат страничната организация на паметта, задават размера на страницата, управляват работата на външния кеш, активират обработката на изключения, активират броячите на появата и продължителността на специфични събития и др.

2.6. ОБРАБОТКА НА ПРЕКЪСВАНИЯТА И ИЗКЛЮЧЕНИЯТА

Изключенията и прекъсванията са събития, които показват, че съществува условие в системата, процесора или в текущата програма или за-

дача, които изискват вниманието на процесора. В резултат изпълнението се предава от текущата програма или задача към специална подпрограма или задача, наричана подпрограма, обслужваща прекъсването или изключението.

Прекъсванията могат да възникват в произволни моменти от изпълнението на дадена програма в резултат на получавани сигнали от хардуера. Системният хардуер използва прекъсвания, за да обслужва събития, външни за процесора, като например заявки за обслужване на периферни устройства. Софтуерът също може да генерира прекъсвания чрез използване на инструкцията за програмно прекъсване INT п.

Изключения възникват, когато процесорът открие грешка при изпълнението на дадена инструкция, като например деление на нула, липсващи страници в паметта, неразрешен достъп и др. Освен грешки в процесора, изключения възникват и при грешки по шината. Прекъсванията и изключенията се обработват прозрачно за приложните програми и операционната система. При получаване на сигнал за прекъсване или откриване на изключение изпълнението на текущата програма временно се преустановява, докато процесорът изпълни подпрограмата, обслужваща прекъсването или изключението. Възстановяването на прекъснатата процедура или задача се осъществява без нарушаване на непрекъснатостта на изпълнението на програмата, освен ако връщането от прекъсване е било възпрепятствано от изключение или прекъсване, предизвикало преустановяване на изпълнението на текущата програма.

На всяко архитектурно дефинирано изключение и всяко условие за прекъсване, изискващо специално обслужване, се присвоява уникален идентификационен номер, наречен вектор. Процесорът използва този вектор като индекс в таблицата на прекъсванията за определяне на входната точка на подпрограмата, обслужваща това изключение или прекъсване. Броят на векторите при архитектурата IA-32 е 256. Векторите от 0 до 31 се използват за архитектурно дефинирани изключения и прекъсвания. Някои от тези вектори са резервирани и не се разрешава тяхното използване. Векторите от 32 до 255 са предназначени за прекъсвания, дефинирани от потребителя. Тези прекъсвания обикновено се използват от външните В/И устройства.

Вектор 0 е за грешка при операция деление, вектор 2 – за немаскируемо прекъсване, вектор 3 – за точка на програмно прекъсване, вектор 4 – за препълване, вектор 6 за невалиден операционен код, вектор 7 – за липсващо устройство, вектор 8 за двойна грешка, вектор 11 – за липсващ сегмент, вектор 13 – за нарушение на общата защита, вектор 14 – за липс-

ваща страница, вектор 16 – за грешка в устройството за плаваща точка, вектор 19 – за грешка при изпълнението на SSE или SSE2 инструкции и т.н.

Процесорите Pentium IV™ и Xeon™ имат вграден в чипа им локален програмируем контролер на прекъсванията. Сигналите за *маскируеми външни прекъсвания* могат да се подават през извода INTR на процесора и могат да използват всичките вектори на прекъсвания от 0 до 255. Друга възможност е сигналите за маскируеми външни прекъсвания да бъдат подавани през локалния програмируем контролер на прекъсванията и в този случай те могат да използват векторите от 16 до 255. Тези прекъсвания могат да бъдат маскирани чрез флага за прекъсване IF във флаговия регистър EFLAGS. Флагът за прекъсване IF може да бъде установен в 1 (разрешаване на прекъсването) с инструкцията STI и нулиран (забрана на прекъсването) с инструкцията CLI. Тези инструкции са разрешени за изпълнение, само ако текущото ниво на привилегированост на процесора CPL е равно или по-малко от нивото на привилегированост IOPL на В/И.

Източниците на изключения могат да бъдат групирани, както следва:

- Изключения, генерирани от процесора при изпълнението на програмата;
- Изключения, генерирани от програмата при изпълнението на инструкциите INTO, INT3 и BOUND;
- Изключения, възникващи при проверка на машината (процесорът Pentium IV™ поддържа механизми за проверка на вътрешния хардуер в чипа и шинните транзакции).

Когато изпълнението на програмата или задачата се възобновява след обработката на грешка, се рестартира за изпълнение инструкцията, предизвикала грешката. При възникване на грешка за липсваща страница, когато програмата или задачата се опитват да прочетат данни от страница, която не е в паметта, подпрограмата, обслужваща това изключение, зарежда липсващата страница в паметта и възобновява изпълнението на програмата или задачата чрез рестартиране на инструкцията, предизвикала изключението. За да се осигури прозрачното рестартиране на тази инструкция за програмата или задачата, процесорът съхранява съдържанието на регистрите и указателите на стековете, за да може да възстанови състоянието си преди изпълнението ѝ.

Спекулативните предварителни изчисления не оказват влияние върху обслужването на прекъсванията от процесора.

Немаскируемо прекъсване може да се генерира или от външен апаратен сигнал през извода NMI на процесора, или когато процесорът

(Pentium™ IV, Xeon™) получи съобщение през системната шина или от външния програмируем контролер на прекъсванията. То се обслужва от подпрограмата, обслужваща немаскируемо прекъсване чрез вектор 2. Освен това процесорът взема мерки, докато не завърши това обслужване, да не се получават и обслужват други прекъсвания, включително мас-кируеми прекъсвания.

Когато възникнат повече от един сигнали за изключения или прекъсвания, процесорът ги обслужва в ред, определен от техните приоритети. Приоритетите при Pentium™ IV, Xeon™ са разпределени на 8 нива, като ниво 1 е най-високоприоритетното, а ниво 8 е с най-нисък приоритет. Приоритетно ниво 1 включва апаратен RESET и проверките на машината. Приоритетно ниво 2 включва изключенията при превключване на задача. Приоритетно ниво 3 включва интервенциите на външния хардуер. Приоритетно ниво 4 включва изключенията от предизвикването на изпълнението на предишна инструкция (програмни точки и стъпков режим). Приоритетно ниво 5 включва външните прекъсвания – немаскируемото прекъсване NMI и маскируемите апаратни прекъсвания. Приоритетно ниво 6 включва грешки от предварително извличане на инструкции, приоритетно ниво 7 – грешки при декодирането на следваща инструкция, приоритетно ниво 8 – грешки при изпълнението на инструкция (препълване, липсващ сегмент, грешка в стека, обща защита, липсваща страница, грешка в устройството за плаваща точка, грешка при SIMD изчисленията).

Pentium IV™ поддържа дескрипторна таблица (таблица на описателите) на прекъсванията IDT, която свързва всяко изключение или вектор на прекъсване с гейт – дескриптор на процедура или задача, обслужваща прекъсването или изключението. Регистърът IDTR съдържа 32-битов базов адрес и 16-битово изместване за дескрипторната таблица на прекъсванията IDT.

Процесорът изпълнява подпрограмата, обслужваща прекъсването или изключението, аналогично на извикванията на процедура с инструкция CALL. Когато отговаря на прекъсване или изключение, процесорът използва вектора на прекъсване или изключение като индекс към дескриптор в дескрипторната таблица на прекъсванията IDT. При извикването на процедурата, обслужваща прекъсването или изключението, процесорът съхранява текущото състояние на флаговия регистър EFLAGS, сегментния регистър на кода CS и указателя на инструкцията EIP. Съдържанието на регистрите CS и EIP осигуряват указател към инструкцията за връщане от прекъсване. За да осъществи връщане от прекъсва-

нето, процесорът изпълнява инструкция IRET, която е аналогична на инструкцията RET с тази разлика, че се възстановяват флаговете EFLAGS.

Задачите при IA архитектурата не са реентрантни, така че подпрограмата, обслужваща прекъсването, трябва да забрани прекъсванията в интервала след завършване на обслужването на прекъсването и изпълнението на инструкцията IRET. В противен случай ще възникне грешка от обща защита.

Когато при обработката на двойна грешка възникне ново изключение, процесорът влиза в режим „изключено“ (*shutdown*). Този режим е подобен на състоянието на процесора след изпълнението на инструкция HLT. Процесорът генерира специален цикъл по шината, за да индицира, че е в състояние „shutdown“. В този режим процесорът спира да изпълнява инструкциите до получаване на сигнал за прекъсване от типа NMI или апаратен RESET.

В чиповете на процесорите Intel® Pentium IV™ и Xeon™ е вграден локален програмируем контролер на прекъсванията, който изпълнява следните функции:

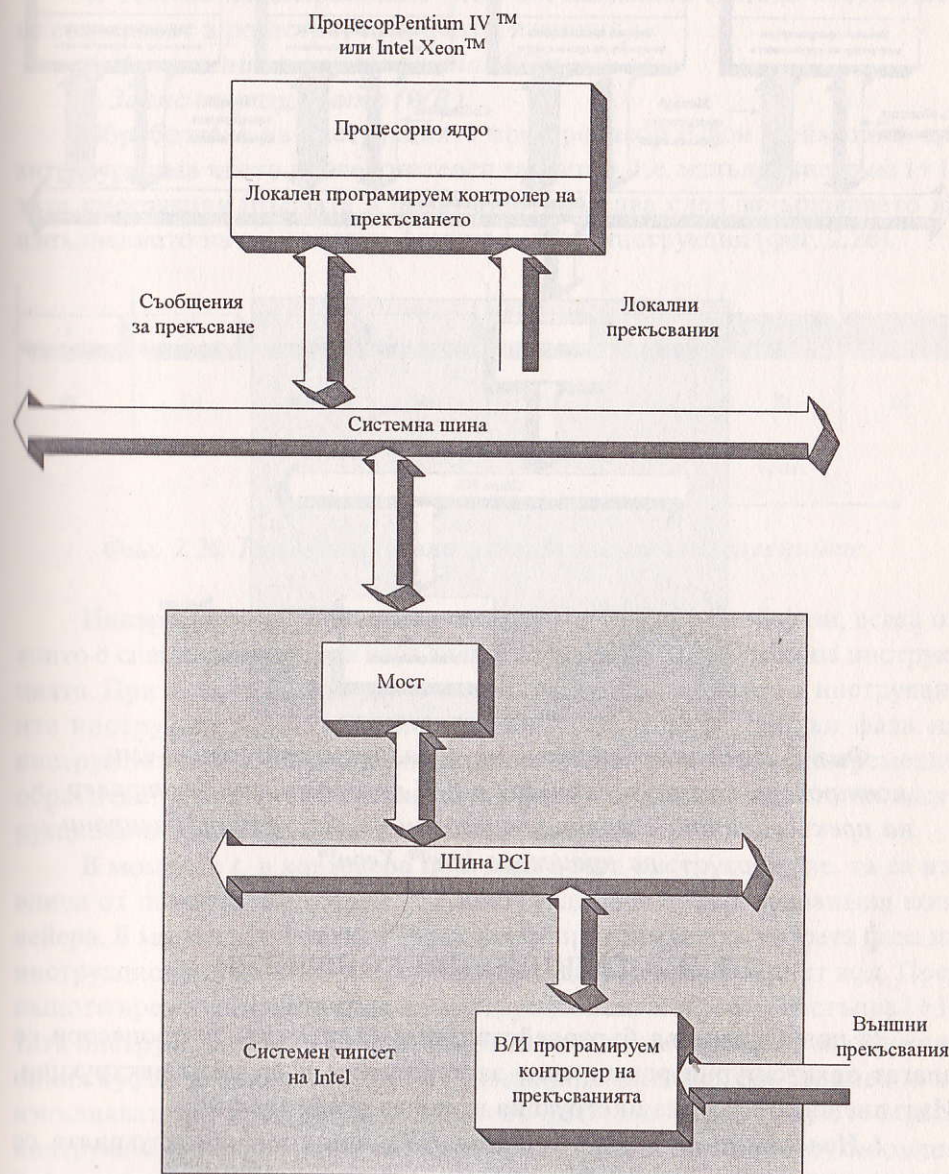
- Приема сигналите за прекъсване от външните изводи за прекъсване на процесора, от вътрешните източници, от външния В/И програмируем контролер на прекъсванията и/или от друг външен програмируем контролер на прекъсванията и ги изпраща към ядрото на процесора за обслужване (фиг. 2.24);

- В мултипроцесорни системи изпраща и получава съобщения за междупроцесорни прекъсвания по системната шина (фиг. 2.25).

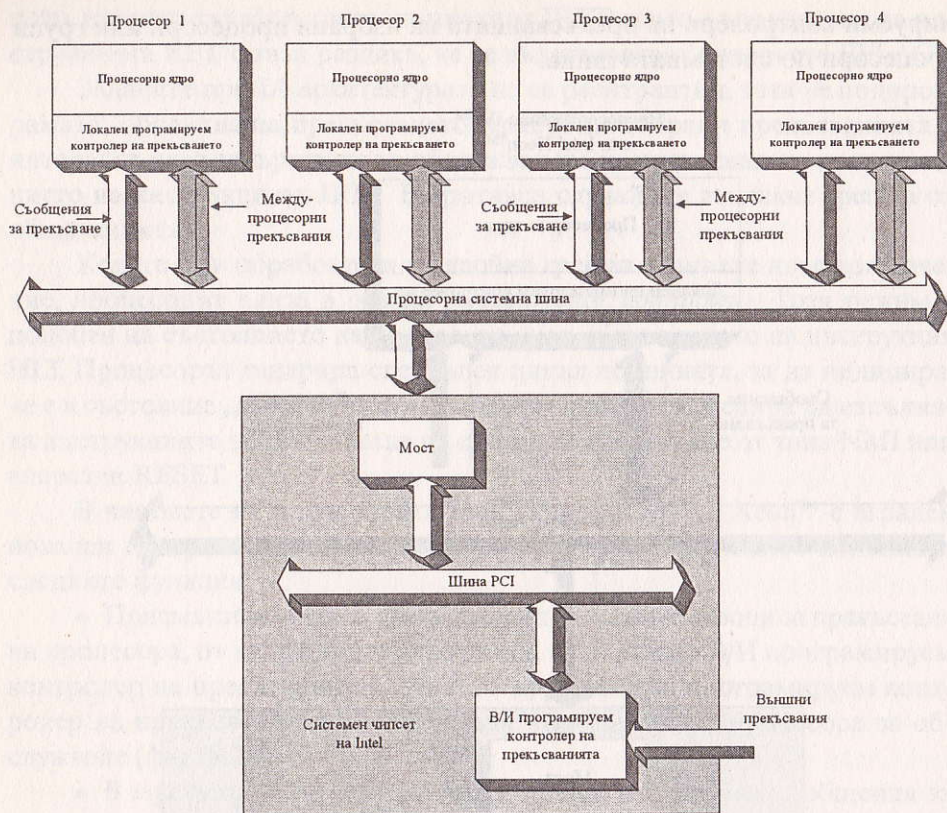
Локалният програмируем контролер на прекъсванията приема сигнали за прекъсване от следните източници: локално свързани В/И устройства, външно свързани В/И устройства (при мултипроцесорни системи), междупроцесорни прекъсвания (при мултипроцесорни системи), прекъсвания, генерирани от вътрешния му таймер, прекъсвания от броячите за мониторинг на производителността, от вътрешния термичен сензор, вътрешни за локалният програмируем контролер на прекъсванията изключения.

Външният програмируем контролер на прекъсванията е част от т. нар. чипсет на Intel®. Неговото основно предназначение е да приема сигналите за външни прекъсвания от системата и В/И устройства и да ги предаде на локалния програмируем контролер на прекъсванията под формата на съобщения за прекъсване. В мултипроцесорни системи В/И програмируем контролер на прекъсванията също така осигурява механизъм за разпределяне на външните прекъсвания към локалните програ-

мируеми контролери на прекъсванията на избрани процесори или групи процесори по системната шина.



Фиг. 2.24. Взаимодействие на локалния и В/И програмируеми контролери на прекъсванията в еднопроцесорна система



Фиг. 2.25. Взаимодействие на локалните програмируеми контролери на прекъсванията с В/И програмируем контролер на прекъсванията в мултипроцесорни конфигурации, базирани на процесора Intel® Xeon™

2.7. ИНСТРУКЦИОННИ КОНВЕЙЕРИ

За повишаване на бързодействието в съвременните процесори се влагат архитектурни разширения за паралелизми на ниво инструкции. Изпълнението на всяка инструкция включва следните фази:

1. **Извличане на инструкцията (FI)**, при което инструкцията се зарежда в регистъра на инструкцията;

2. **Декодирание на инструкцията (DI)**, при което се определя операционният код и спецификаторите на операндите;

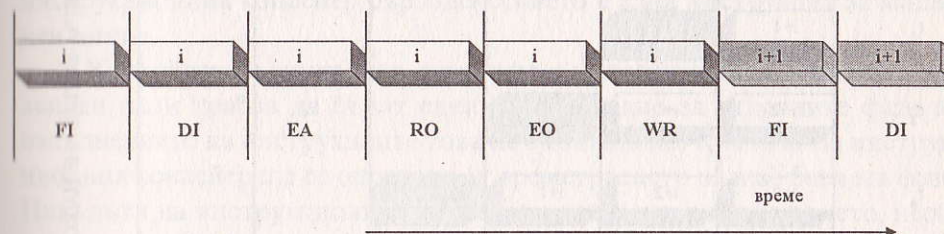
3. **Изчисляване на ефективните адреси на операндите (EA)** в зависимост от режима на адресиране;

4. **Четене на операндите (RO)** от паметта (когато операндите се съхраняват в регистри, тази фаза е излишна);

5. **Изпълнение на операцията (EO)**;

6. **Запис на резултата (WR)**.

Обработката на инструкциите при процесор с фон Нойманова архитектура има чисто последователен характер, т.е. изпълнението на $i+1$ -тата инструкция (първата ѝ фаза) стартира едва след завършването на изпълнението на последната фаза на i -тата инструкция (фиг. 2.26).

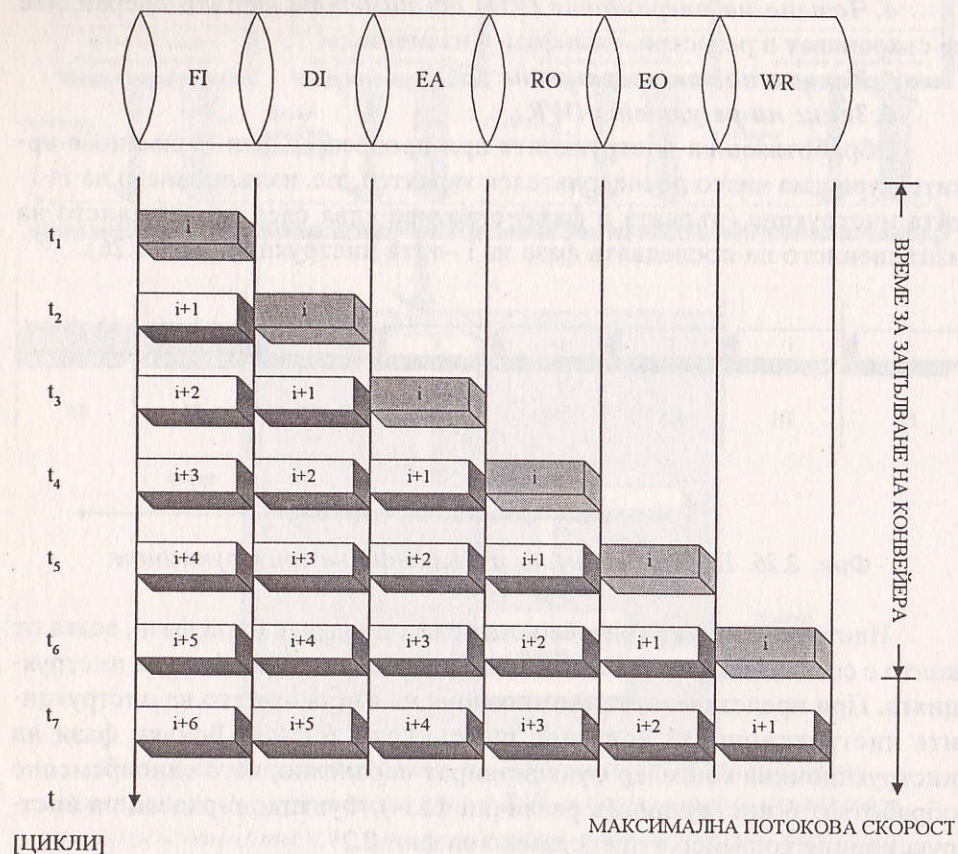


Фиг. 2.26. Последователно изпълнение на инструкциите

Инструкционният конвейер съдържа определен брой фази, всяка от които е специализирана за изпълнението на определена фаза на инструкцията. При представената декомпозиция на изпълнението на инструкциите инструкционният конвейер ще съдържа 6 фази. Всички фази на инструкционния конвейер функционират паралелно, като едновременно обработват 6 инструкции (в различни фази). Функционирането на инструкционния конвейер е представено на фиг. 2.27.

В момента t_1 в конвейера постъпва i -тата инструкция, т.е. тя се извлича от паметта и се зарежда в инструкционен буфер, захранващ конвейера. В момента t_2 i -тата инструкция се придвижва във втората фаза на инструкционния конвейер, където се декодира операционният код. През същото време в първата фаза на инструкционния конвейер постъпва $i+1$ -тата инструкция, т.е. тя се извлича от паметта и се зарежда в инструкционния буфер. В момента t_3 в инструкционния конвейер едновременно се изпълняват три инструкции в различни фази: в първата фаза е $i+2$ -тата инструкция, която се извлича от паметта и се зарежда в инструкционния буфер, във втората фаза е $i+1$ -тата инструкция, на която се декодира операционният код и в третата фаза е i -тата инструкция, за която се изчисляват ефективните адреси на операндите. На всеки цикъл инструкционният

ИНСТРУКЦИОНЕН КОНВЕЙЕР



Фиг. 2.27. Паралелна обработка на инструкции в шестфазен инструкционен конвейер

конвейер се захранва от инструкционния буфер, т.е. на входа му (в първа фаза) постъпва за изпълнение по една инструкция. След запълването на инструкционния конвейер, т.е. след като i -тата инструкция постъпи в последната фаза, в инструкционния конвейер едновременно се изпълняват толкова инструкции, колкото фази има. За примера, който разглеждаме, инструкционният конвейер обработва едновременно 6 инструкции в различни фази от тяхното изпълнение: първата фаза извлича $i+5$ -тата инструкция от паметта и се зарежда в инструкционния буфер, втората фаза на

инструкционния конвейер декодира $i+4$ -тата инструкция, третата фаза изчислява ефективните адреси на операндите на $i+3$ -тата инструкция, четвъртата фаза чете операндите от паметта на $i+2$ -тата инструкция, петата фаза изпълнява операцията на $i+1$ -тата инструкция, а последната шеста фаза записва резултата на i -тата инструкция в паметта.

Времето за запълване на конвейера се нарича време за инициализация (начално установяване) и изисква толкова цикъла, колкото са фазите на инструкционния конвейер. След запълването си инструкционният конвейер започва да работи с максимална потокова скорост, т.е. на всеки цикъл завършва изпълнението на една инструкция. Следователно при инструкционния конвейер бързодействието е една инструкция за машинен цикъл.

Конвейерът е синхронна структура и времената на обработката във всички фази трябва да бъдат еднакви. Очевидно за отделните фази на изпълнението на инструкциите това не е изпълнено. Цикълът на инструкционния конвейер ще се определя от времетраенето на най-бавната фаза. Цикълът t на инструкционния конвейер се дефинира като времето, необходимо за обработката на множеството инструкции в една фаза и придвижването им в следващата фаза, т.е.

$$\tau = \tau_{\max}(\tau_i) + d = \tau_{\max} + d, \text{ за } i \in [1, k], \quad (2.10)$$

където:

- k – брой на фазите на инструкционния конвейер;
- τ_i – времетраене на обработката в i -тата фаза на инструкционния конвейер;
- $\tau_{\max}$ – времетраене на обработката в най-бавната фаза на инструкционния конвейер;
- d – време за предаване на сигналите и данните от една фаза към следващата фаза на инструкционния конвейер.

Времето за запълване на конвейера е

$$T_{\text{init, pipeline}} = k * \tau \quad (2.11)$$

Следователно, времето за обработка на n инструкции в k -фазен инструкционен конвейер е:

$$T_{\text{pipeline, n}} = [k + (n-1)] * \tau \quad (2.12)$$

Ускорението при изчисленията се дефинира като отношение на времето за последователно изпълнение на инструкциите към времето за изпълнение на същите инструкции от инструкционен конвейер.

$$S = T_{seq, n} / T_{pipeline, n} = (n * k * \tau) / [k + (n-1)] * \tau \quad (2.13)$$

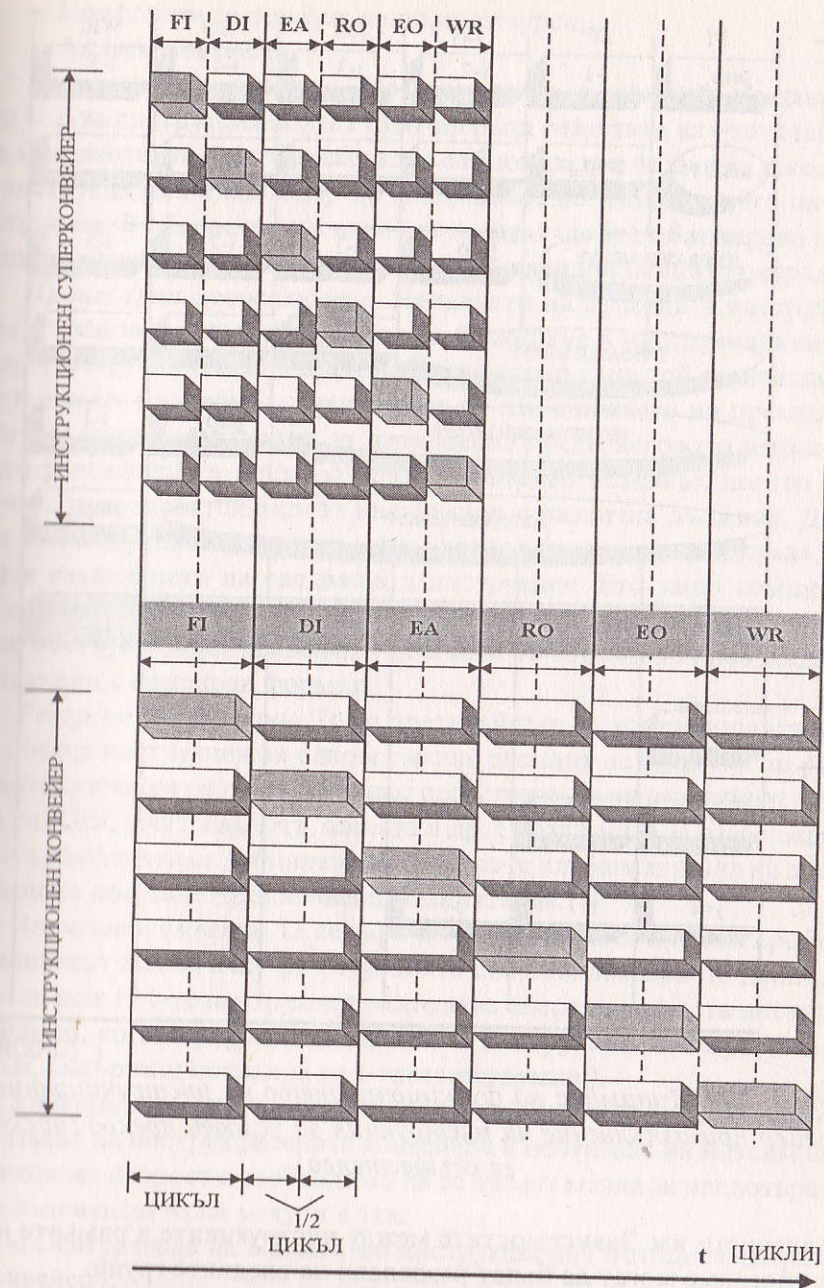
$$S = (n * k) / [k + (n-1)] \quad (2.14)$$

За много големи стойности на броя на изпълнението на инструкциите $n \gg k$ ускорението се определя от броя на фазите на инструкционния конвейер $S \rightarrow k$.

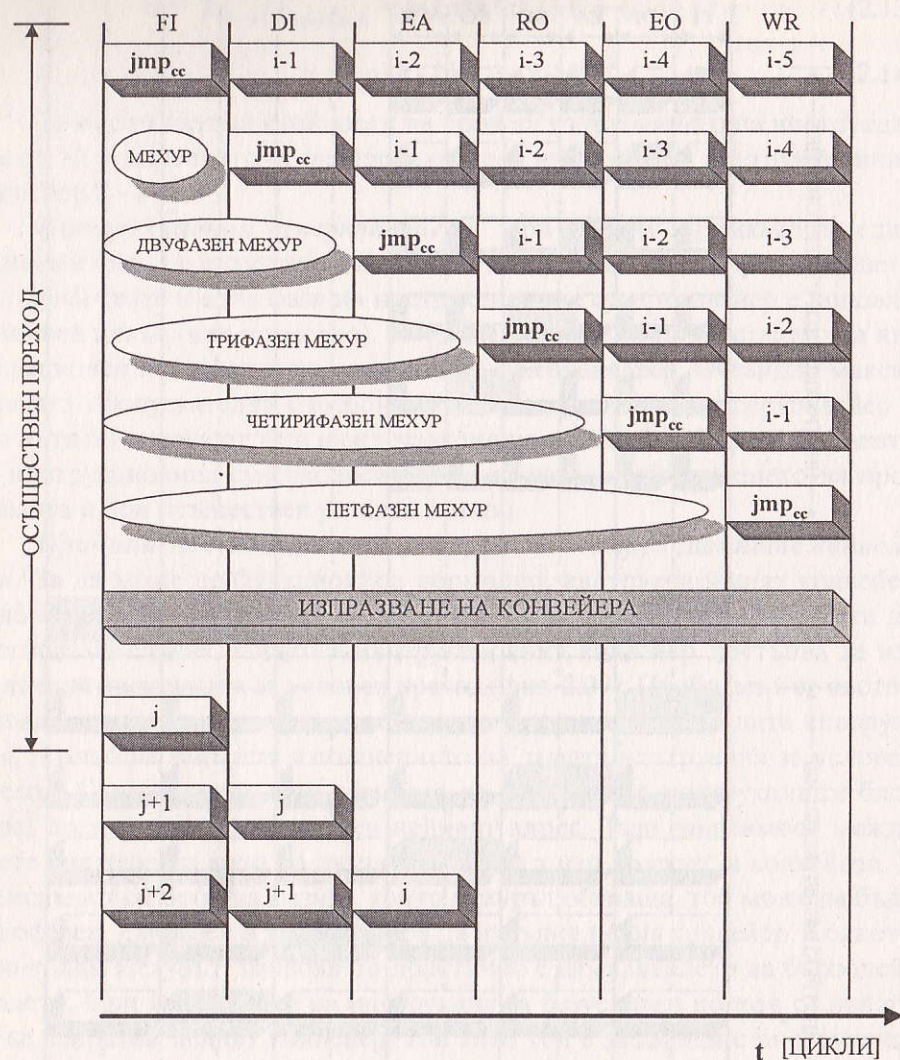
Инструкционни суперконвейери. При тях в рамките на един машинен цикъл е възможно изпълнението на две фази или времетраенето на обработката в една фаза на инструкционния суперконвейер е половин машинен цикъл (или по-малко). На фиг. 2.28 е сравнено действието на инструкционен конвейер и инструкционен суперконвейер. Очевидно максималната пропускателна способност на инструкционния суперконвейер е два пъти по-висока от тази на инструкционния конвейер. Бързодействието на инструкционния суперконвейер е по-ниско при стартирането на програмата и при осъществен условен преход.

Причини за възникване на „мехури“ в инструкционните конвейери. За да може да функционира нормално, инструкционният конвейер трябва непрекъснато да бъде захранван с поток от инструкции. Нека да разгледаме случая, когато в инструкционния конвейер постъпва за изпълнение инструкция за условен преход (фиг. 2.29). Проблемът се състои в това, че е невъзможно да се определи адресът на следващата инструкция, докато не завърши изпълнението на самата инструкция за условен преход. Следователно изпълнението на следващата инструкция се блокира, докато не стане известен нейният адрес. Тази зависимост между двете инструкции води до възникването на т.нар. „мехур“ в конвейера. В зависимост от броя на фазите, които мехурът обхваща, той може да бъде еднофазен, двуфазен и т.н. или дори да изпълни целия конвейер. Колкото е по-голям мехурът, толкова по-драстично е намаляването на бързодействието. При изпълнение на инструкция за безусловен преход се налага да се изпразни целият конвейер, тъй като той е запълнен с инструкции, следващи непосредствено инструкцията за безусловен преход.

Паралелизмът на ниво инструкции отразява степента на възможното паралелно изпълнение на инструкциите. За да бъде възможно 100%-ното изпълнение на инструкциите е необходимо те да бъдат независими една от друга. При реалните програми, обаче, съществуват множество и различни по вид зависимости между инструкциите. Паралелното изпълнение на зависими инструкции води до възникването на мехури в инструкционните конвейери, което значително намалява бързодействието при



Фиг. 2.28. Сравнение на функционирането на инструкционен суперконвейер и инструкционен конвейер



Фиг. 2.29. Динамика на функционирането на инструкционния конвейер при изпълнение на инструкция за условен преход (преходът се осъществява)

изпълнението им. Зависимостите между инструкциите в рамките на дадена програма могат да бъдат разделени на следните групи:

- Зависимости по данни;
- Процедурни зависимости;

- Конфликти за заемане на общи ресурси;
- Антязависимости.

Зависимости по данни. Две инструкции са зависими по данни, когато $i+1$ -та инструкция ползва като операнд резултата на i -тата инструкция. Следователно $i+1$ -та инструкция се блокира във фазата на декодиране на инструкционния конвейер до завършване на изпълнението на i -тата инструкция. В общия случай една инструкция ще бъде блокирана в инструкционния конвейер дотогава, докато са налични нейните операнди.

Процедурни зависимости. Наличието на преходи в инструкционния поток повишава вероятността за блокиране в инструкционния конвейер. Инструкциите след прехода (независимо дали той се осъществява или не) имат процедурна зависимост от изпълнението на прехода и не могат да бъдат стартирани за изпълнение преди неговото изпълнение. Друго усложнение в инструкционния конвейер възниква, когато се изпълнява последователност от инструкции с различна дължина. Дължината на инструкцията става известна едва във фазата на декодиране, което забавя извличането на следващата инструкция. Ето защо компютрите със съкратен набор инструкции (RISC) са по-ефективни при използването на инструкционни конвейери, тъй като те използват преобладаващо инструкции с фиксиран формат.

Ресурсни конфликти. Те се предизвикват от конкуренцията на две или повече инструкции за едновременно заемане на един и същ ресурс. Общите системни ресурси включват регистрите, функционалните устройства на АЛУ, кеша, паметта, шината и др. Стратегиите за разрешаването на тези конфликти са дублиране на ресурсите или намаляване на времето за тяхното ползване чрез конвейеризацията им.

Антязависимости. Те се наричат така, тъй като при тях проблемите възникват аналогично на ситуацията със зависимости по данни, но в обратен ред: $i+1$ -та инструкция разрушава операнд на i -тата инструкция (например, когато резултатът на $i+1$ -та инструкция се съхранява в регистър, съдържащ операнда на i -тата инструкция).

Уплътняване на инструкционните конвейери. За максимално използване на инструкционните конвейери и постигане на максималната им потокова скорост е необходимо да се вземат мерки за предотвратяване на възникването на мехури в тях.

- Осигуряване на множество инструкционни потоци за захранване на конвейера;
- Предварително извличане на следващата инструкция при осъществен преход;

- Осигуряване на инструкционен буфер за тялото на цикли;
- Отлагане на прехода;
- Предсказване на прехода;
- Промяна на реда на изпълнението на инструкциите, заложен в програмата;
- Оптимизация на използването на регистрите.

За преодоляване на отрицателните ефекти на процедурните зависимости върху работата на инструкционните конвейери е препоръчително предварително да се извлича и инструкцията-цел на прехода.

RISC процесорите масово използват метода на отложения преход. При него се подлагат на анализ зависимостите на инструкциите в рамките на програмата. Възникналите мехури при инструкции за безусловен преход или при осъществен условен преход се запълват с изпълнението на независими инструкции, т.е. осъществява се промяна на зададения ред на изпълнението на инструкциите в програмата.

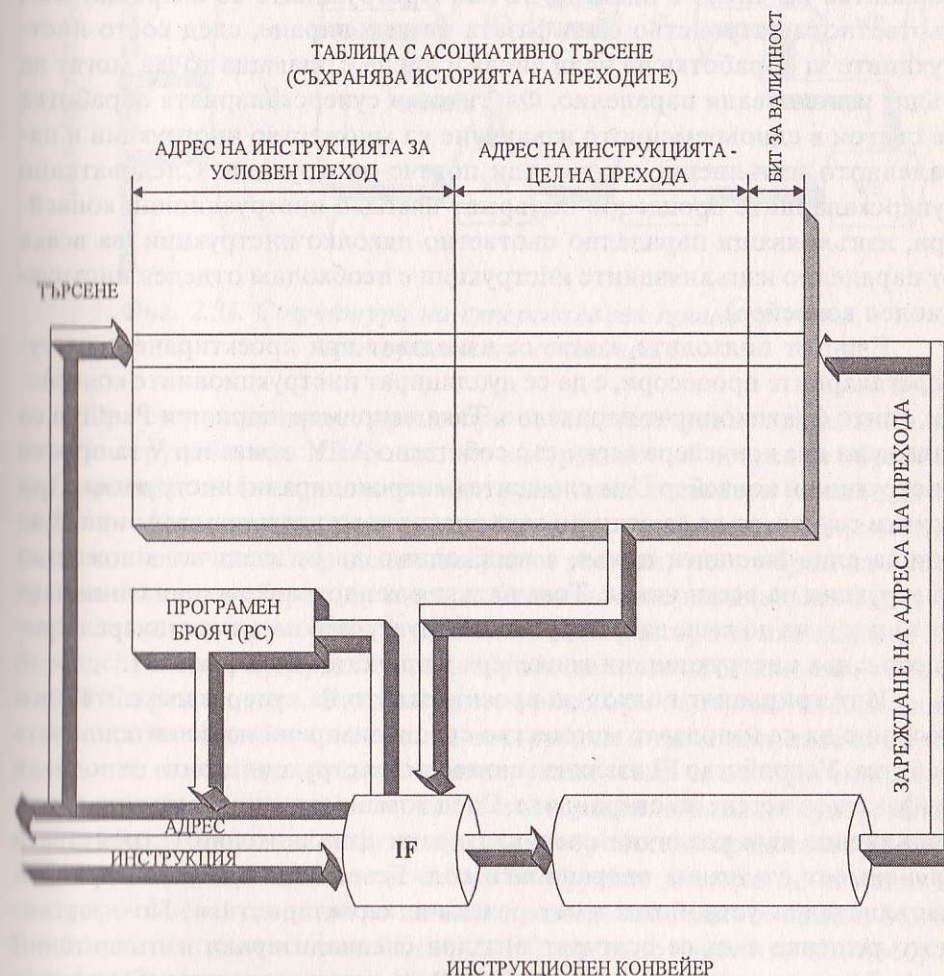
Съвременните процесори използват различни стратегии за предсказване на преходите, които могат да бъдат статични и динамични. Основната разлика е, че при динамичните стратегии за предсказване на прехода се следи и се взема предвид поведението на програмата до стартирането на конкретната инструкция за условен преход.

Най-често използваните статични стратегии за предсказване на прехода се основават на следното:

- Приема се, че преходът никога не се осъществява и се извличат следващите инструкции;
- Приема се, че преходът винаги се осъществява и се извличат инструкцията-цел на прехода и последователността от инструкциите след нея;
- Решението зависи от операционния код на инструкцията за условен преход.

Основната цел на динамичните стратегии за предсказване на прехода е да се повиши точността на прогнозата, като се използва историята на изпълнението на програмата до този момент. За съхраняването на историята на всяка инструкция за условен преход (или поне на последните изпълнени такива) се използват допълнителни битове, съдържащи стойностите на ключовете за осъществени/неосъществени преходи. Таблицата, съхраняваща историята на преходите, обикновено се съдържа в малък кеш, свързан с фазата на конвейера за предварително извличане на инструкции. Всяка позиция в таблицата съдържа три компоненти: адрес на инструкцията за условен преход, битове за историята на прехода, информация за целта на прехода (адрес на инструкцията-цел на прехода или

самата инструкция-цел на прехода). Таблицата, съхраняваща историята на преходите, се нарича буфер за целта на прехода **ВТВ** (Branch Target Buffer). На фиг. 2.30 е показана структурата на буфер за целта на прехода. При използването на ВТВ е възможно да се достигне до 85-90% прогнозиране на преходите. Предимствата на този подход са, че не се изисква рекомпиляция на кода или промяна в машинните инструкции. Недостатъците са, че апаратната реализация е скъпа и ефективността на подхода силно се влияе от контекстните превключвания.



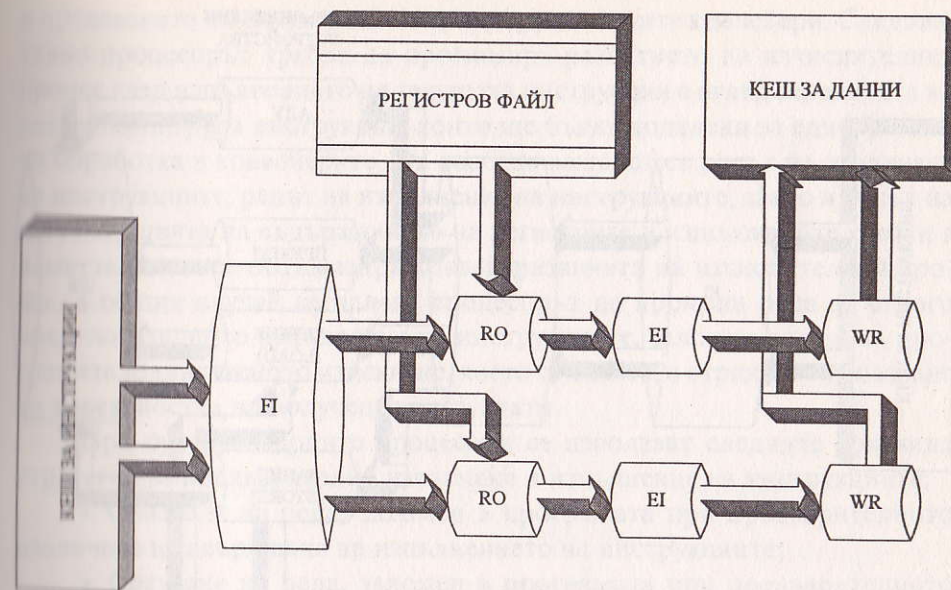
Фиг. 2.30. Инструкционен конвейер с предсказване на прехода в ВТВ

2.8. СУПЕРСКАЛАРНИ ПРОЦЕСОРИ

Конвенционалният „скаларен“ процесор изпълнява скаларни инструкции, т.е. инструкции, обработващи цели числа. Суперскаларният процесор изпълнява едновременно две или повече скаларни инструкции. Ограничена форма на суперскаларна обработка съществува при процесори, които обработват цели числа и числа с плаваща точка. При тях се използват отделни устройства за обработка на цели числа и за обработка на числа с плаваща точка. Инструкциите се изпращат към съответното устройство след фазата на декодиране, след което инструкциите за обработка на цели числа и числа с плаваща точка могат да бъдат изпълнявани паралелно. Фактически суперскаларната обработка се състои в едновременното извличане на множество инструкции и паралелното изпълнение на две или повече инструкции. Следователно суперскаларните процесори съдържат няколко инструкционни конвейери, изпълняващи паралелно съответно няколко инструкции (за всяка от паралелно изпълняваните инструкции е необходим отделен инструкционен конвейер).

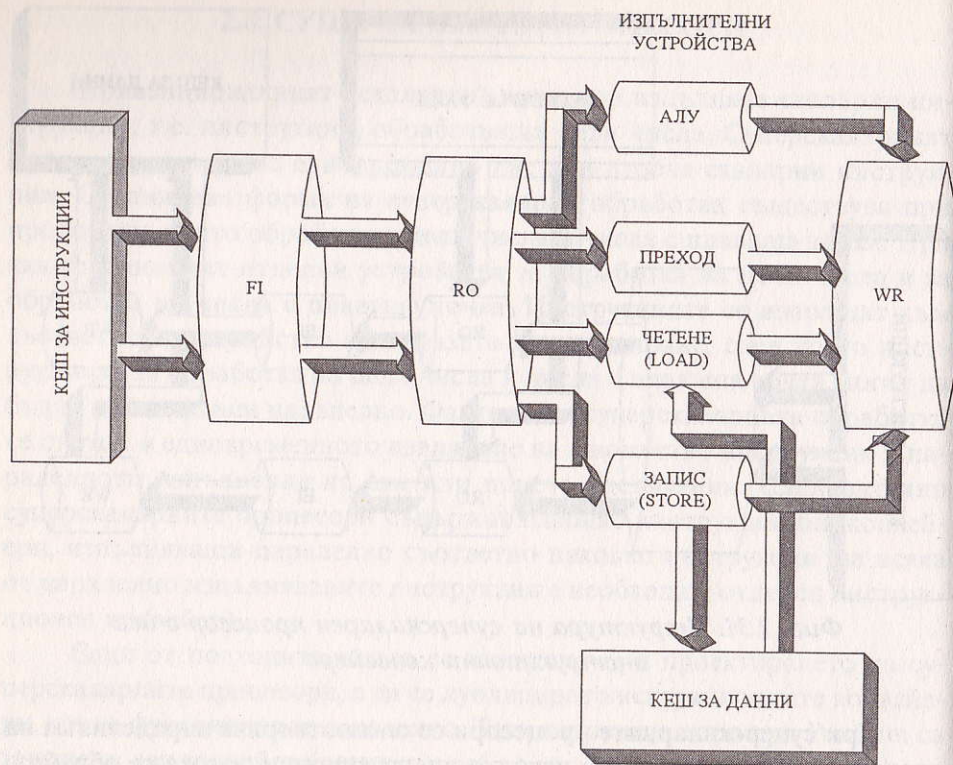
Един от подходите, които се използват при проектирането на суперскаларните процесори, е да се дублицират инструкционните конвейери, които функционират паралелно. Така например, дори при Pentium са заложили два конвейера всеки със собствено АЛУ: конвейер V за прости инструкции и конвейер U за сложните (микрокодирани) инструкции. При всички случаи, за да се осигури изпълнението на две или повече инструкции за един машинен цикъл, е необходимо да се извличат множество инструкции на всеки цикъл. Това налага разширяване на шината за данни за достъпа до кеша за инструкции. Структурата на суперскаларен процесор с два инструкционни конвейера е представен на фиг. 2.31.

Интегрираният подход за проектирането на суперскаларните процесори е да се използват множество специализирани изпълнителни устройства. Устройство FI извлича множество инструкции, които се подават на фазата за четене на операндите. След това инструкциите се подават за изпълнение към различни специализирани изпълнителни устройства в зависимост от техния операционен код. Естествено специализираните изпълнителни устройства имат различни характеристики. По-ефективното решение е да се осигурят отделни специализирани изпълнителни устройства за четене от паметта (Load), за запис в паметта (Store) и за преходи (Branch), отколкото те да се интегрират в един многофункционален конвейер (фиг. 2.32).



Фиг. 2.31. Структура на суперскаларен процесор с два инструкционни конвейера

При суперскаларните процесори се оползотворява паралелизъм на ниво инструкции, при което няколко инструкционни потока се обработват паралелно в два или повече инструкционни конвейери (които могат да бъдат оформени и като суперконвейери). Бързодействието зависи също така и от наличните паралелни апаратни ресурси, т.е. от броя на инструкционните конвейери в процесора. Следователно паралелният хардуер и паралелизмът на ниво инструкции са важни фактори за повишаване на бързодействието. Суперскаларната обработка може да се използва в RISC и в CISC процесорите. Съвременните процесори съдържат няколко инструкционни конвейера за прости инструкции и един или повече инструкционни конвейера за сложни инструкции, така че за един машинен цикъл се изпълняват две или повече инструкции. Модел на суперскаларен процесор с множество функционални устройства е представен на фиг. 2.32. Устройствата за запис и четене осигуряват достъпа до паметта и съдържат всички средства за изпълнението на операциите Load и Store. В модела е показано само едно АЛУ, но съвременните процесори съдържат повече такива устройства в зависимост от очаквания интензитет на аритметичните операции. Допълнително АЛУ и регистри могат да бъдат добавени за обработката на числа с плаваща точка.



Фиг. 2.32. Суперскаларен процесор със специализирани изпълнителни устройства

Паралелизмът на ниво инструкции се ограничава от честотата на зависимостите по данни и процедурните зависимости между инструкциите и зависи както от системата инструкции, така и от самата приложна програма.

За да се увеличи степента на паралелизъм, т.е. за да се увеличи броят на паралелно обработваните инструкции в инструкционните конвейери, се използват както апаратни техники, така и оптимизиращи функции на компилатора. При наличие на зависимости между инструкциите в програмата процесорът може да завърши изпълнението на инструкциите в ред, различен от този, заложен в програмата. За оптимизиране на използваната степен на паралелизъм на ниво инструкции процесорът трябва да определи групите инструкции, които могат да бъдат изпълнени паралелно и да организира тяхното извличане, декодиране и изпълнение. Това рефлектира в стратегията за предварително извличане на инструкциите

и предаването им за изпълнение в инструкционните конвейери. Следователно процесорът трябва да прогнозира развитието на изчислителния процес след изпълнението на текущата инструкция с оглед селекцията на следващата група инструкции, които ще бъдат подадени за едновременна обработка в конвейерите. От особена важност са редът на извличане на инструкциите, редът на изпълнение на инструкциите, както и редът на актуализацията на съдържанието на регистрите и използваните клетки в паметта. За да се оптимизира конвейеризацията на изчислителния процес, в общия случай се налага процесорът да промени реда за строго последователното изпълнение на инструкциите, заложен в кода на програмата. Единственото изискване, което се спазва, е стриктното спазване на коректността на получените резултати.

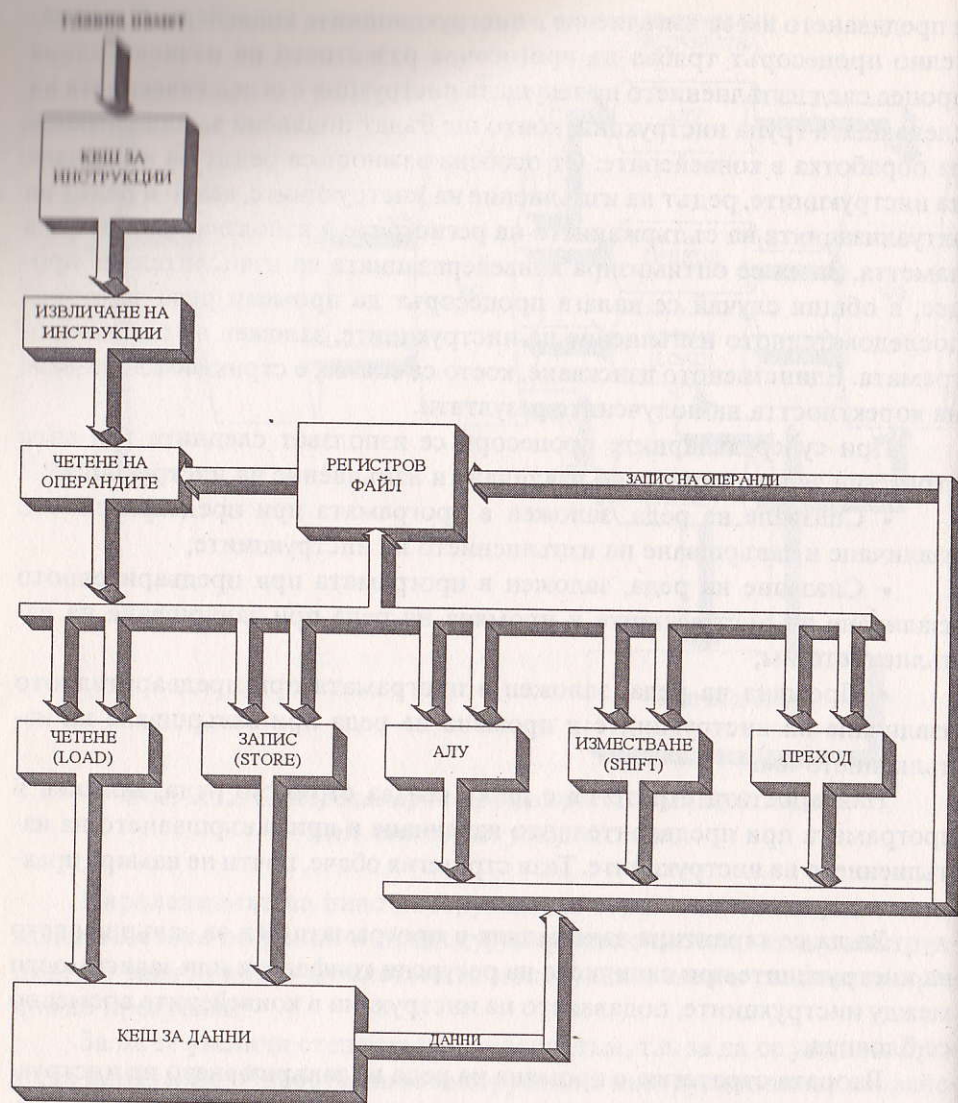
При суперскаларните процесори се използват следните три вида стратегии за предварително извличане и изпълнение на инструкциите:

- Спазване на реда, заложен в програмата при предварителното извличане и завършване на изпълнението на инструкциите;
- Спазване на реда, заложен в програмата при предварителното извличане на инструкциите и промяна на реда при завършване на изпълнението им;
- Промяна на реда, заложен в програмата при предварителното извличане на инструкциите и промяна на реда при завършване на изпълнението им.

Най-простата стратегия е да се спазва стриктно реда, заложен в програмата при предварителното извличане и при завършването на изпълнението на инструкциите. Тази стратегия обаче, почти не намира практическо приложение.

За да се гарантира заложеният в програмата ред за завършването на инструкциите при наличието на ресурсни конфликти или зависимости между инструкциите, подаването на инструкции в конвейерите временно се блокира.

Втората стратегия, с промяна на реда на завършването на инструкциите, дава възможност в произволен момент от времето множество инструкции, чийто брой се ограничава само от степента на хардуерния паралелизъм, да бъдат обработвани паралелно в инструкционните конвейери. Мехури в конвейерите могат да бъдат предизвикани само от зависимости по данни, процедурни зависимости или ресурсни конфликти. Тук са актуални проблемите, предизвикани от т.нар. зависимости „запис-запис“, при които един и същ регистър съдържа резултата на i -тата инструкция и операнда на $i+1$ -тата инструкция. Стратегията с промяна



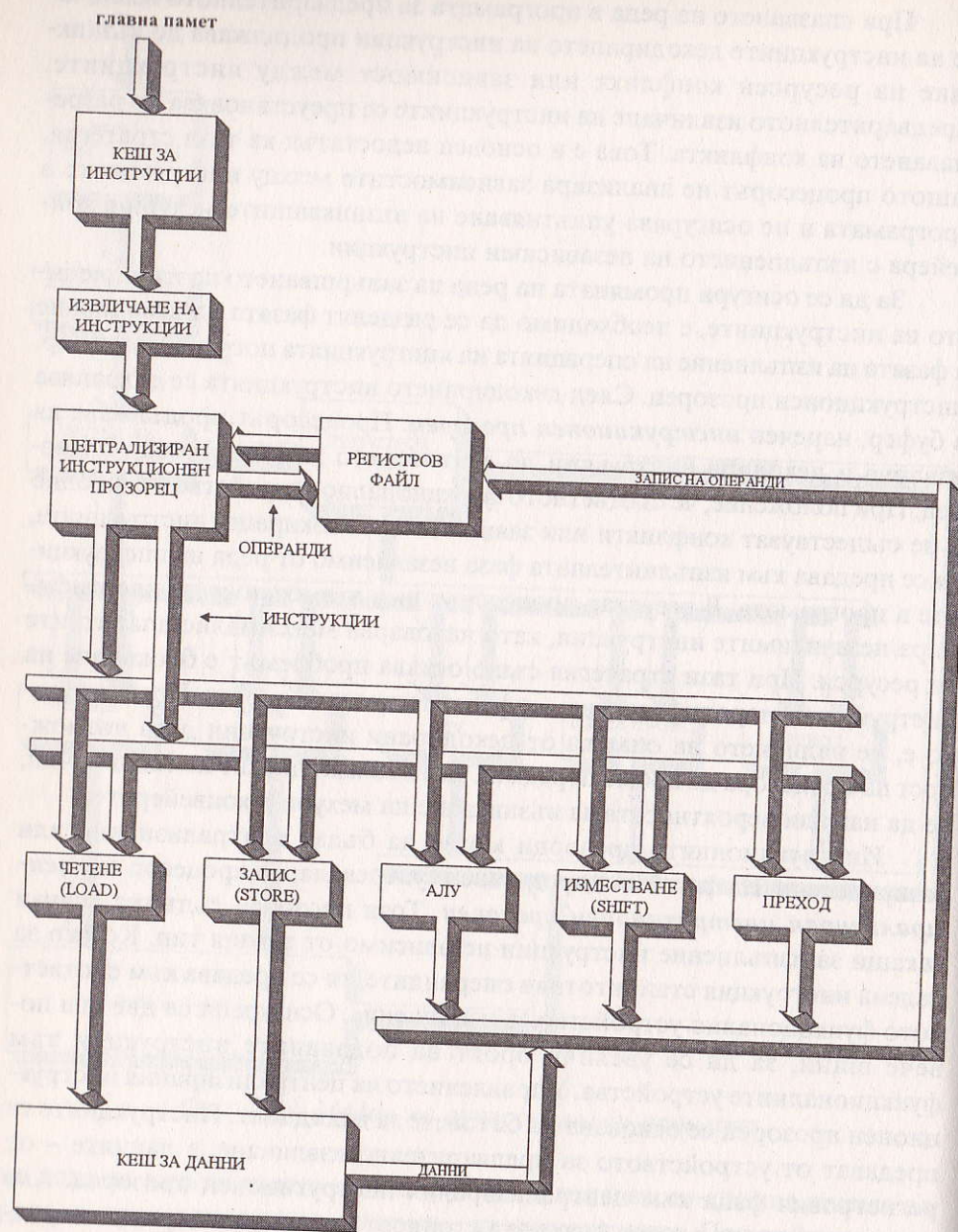
Фиг. 2.33. Модел на суперскаларен процесор

на реда на завършването на инструкциите изисква по-сложна логика на реда на предварителното извличане на инструкциите. При нейното прилагане възникват сериозни проблеми с обработката на прекъсванията и изключенията и по-точно с възобновяването на работата на процесора след обработката на прекъсването.

При спазването на реда в програмата за предварителното извличане на инструкциите декодирането на инструкции продължава до възникване на ресурсен конфликт или зависимост между инструкциите. Предварителното извличане на инструкциите се преустановява до разрешаването на конфликта. Това е и основен недостатък на тази стратегия, защото процесорът не анализира зависимостите между инструкциите в програмата и не осигурява уплътняване на възникващите мехури в конвейера с изпълнението на независими инструкции.

За да се осигури промяната на реда на завършването на изпълнението на инструкциите, е необходимо да се разделят фазата на декодиране и фазата на изпълнение на операцията на инструкцията посредством т.нар. инструкционен прозорец. След декодирането инструкцията се съхранява в буфер, наречен *инструкционен прозорец*. Процесорът продължава да извлича и декодира инструкции до запълването на регистровия прозорец. При положение, че съответното функционално устройство е свободно и не съществуват конфликти или зависимости, блокиращи инструкцията, тя се предава към изпълнителната фаза независимо от реда на инструкциите в програмата. В резултат процесорът има възможност да идентифицира независимите инструкции, като натоварва максимално апаратните си ресурси. При тази стратегия също остава проблемът с блокиране на инструкциите поради ресурсни конфликти или зависимости. Предимството е, че наличието на опашка от декодирани инструкции дава възможност на процесора да промени реда на тяхното изпълнение по такъв начин, че да намали вероятността за възникване на мехури в конвейерите.

Инструкционните прозорци могат да бъдат централизирани или разпределени. На фиг. 2.34 е представен суперскаларен процесор със *централизиран инструкционен прозорец*. Този прозорец съдържа всички чакащи за изпълнение инструкции независимо от техния тип. Когато за дадена инструкция станат готови операндите, тя се предава към съответното функционално устройство за изпълнение. Осигурени са две или повече шини, за да се увеличи броят на подаваните инструкции към функционалните устройства. Управлението на централизирания инструкционен прозорец се основава на битовите за валидност. Инструкциите се предават от устройството за предварително извличане, а данните – от регистровия файл към централизирания инструкционен прозорец. Ако даден операнд не е готов (неговата стойност не е актуализирана от предходна инструкция), битът за валидност е нулиран. В този случай идентификаторът на съответния регистър се записва в инструкционния прозорец на мястото на операнда. Следователно инструкционният прозорец



Фиг. 2.34. Суперскаларен процесор с централизиран инструкционен прозорец

съдържа инструкции с операнди или идентификатори на регистри, както е показано на фиг. 2.35. Когато всичките операнди станат готови, инструкцията се подава за изпълнение към съответното функционално устройство. В случаите, когато инструкцията актуализира стойността на регистър, съответните идентификатори на този регистър в инструкционния прозорец се заменят със стойността на операнда. По този начин актуализациите на регистровия файл и на инструкционния прозорец се извършват едновременно. Инструкционният прозорец се имплементира с асоциативна памет. За примера, показан на фиг. 2.35, инструкциите 1, 2, 4 и 5 имат по един операнд, но другият операнд не е готов. Инструкция 3 има два готови операнда и може да бъде изпратена за изпълнение. При актуализиране на инструкционния прозорец се извършва обръщение към всички инструкции, които имат операнд в референтния регистър. След актуализацията всички инструкции с налични операнди се предават за изпълнение.

Централизираният инструкционен прозорец трябва да има съответната логика за селекция на адекватното функционално устройство за изпълнение на инструкцията. Когато дадена инструкция бъде предадена за изпълнение, съответната ѝ позиция в инструкционния прозорец се освобождава и тя може да бъде използвана от нова инструкция. За да се запази редът на изпълнението на инструкциите в програмата, инструкционният прозорец може да се реализира като диспечеризиращ стек (dispatch stack). Новите инструкции се добавят към върха на прозореца, а съществуващите позиции с инструкции се придвижват надолу към дъното на стека. Друг вариант е инструкционният прозорец да се имплементира с регистри, организирани в опашка с дисциплина на обслужване „пръв влязъл – пръв излязъл“ като позициите с инструкции, включително празните, се придвижват към началото на опашката.

При разпределената версия на инструкционния прозорец инструкционните буфери, наречени резервационни станции, се осигуряват за всички функционални устройства (фиг. 2.36). Декодираните инструкции се изпращат към резервационните станции на съответните функционални устройства. Инструкцията се предава от резервационната станция към функционалното устройство за изпълнение, когато то е свободно и операндите на инструкцията са готови. При актуализация на съдържанието на регистровия файл, в резервационните станции идентификаторите на актуализираните регистри се заменят със стойностите на операндите. Размерът на резервационната станция зависи от типа на функционалното устройство, което тя захранва. Всяко функционално устройство може

регистри. Използването на техники за оптимизиране на използването на регистрите от компилаторите още повече влошава положението, тъй като те целят максимално използване на регистрите.

За решаването на проблемите с антизависимостите и зависимости-те от типа „запис-запис“ се използва подход с дублиране на ресурсите, наречен *преименуване на регистрите*. Регистрите съхраняват стойности, необходими за изпълнението на инструкциите и се разпределят динамично от процесора. При изпълнението на инструкция, чийто операнд се съдържа в регистър, процесорът разпределя нов регистър за този операнд. Нека да разгледаме примерната последователност от инструкции:

ADD R1, R2, R4; $R1 = R2 + R4$

ADD R2, R1, 1; $R2 = R1 + 1$

ADD R1, R4, R5; $R1 = R4 + R5$

Тази инструкционна последователност съдържа няколко зависимости. Въвеждаме нови регистри $R1^*$ и $R2^*$ и преобразуваме последователността по следния начин, като елиминираме антизависимостта и зависимостта „запис-запис“:

ADD R1, R2, R4; $R1 = R2 + R4$

ADD R2*, R1, 1; $R2^* = R1 + 1$

ADD R1*, R4, R5; $R1^* = R4 + R5$

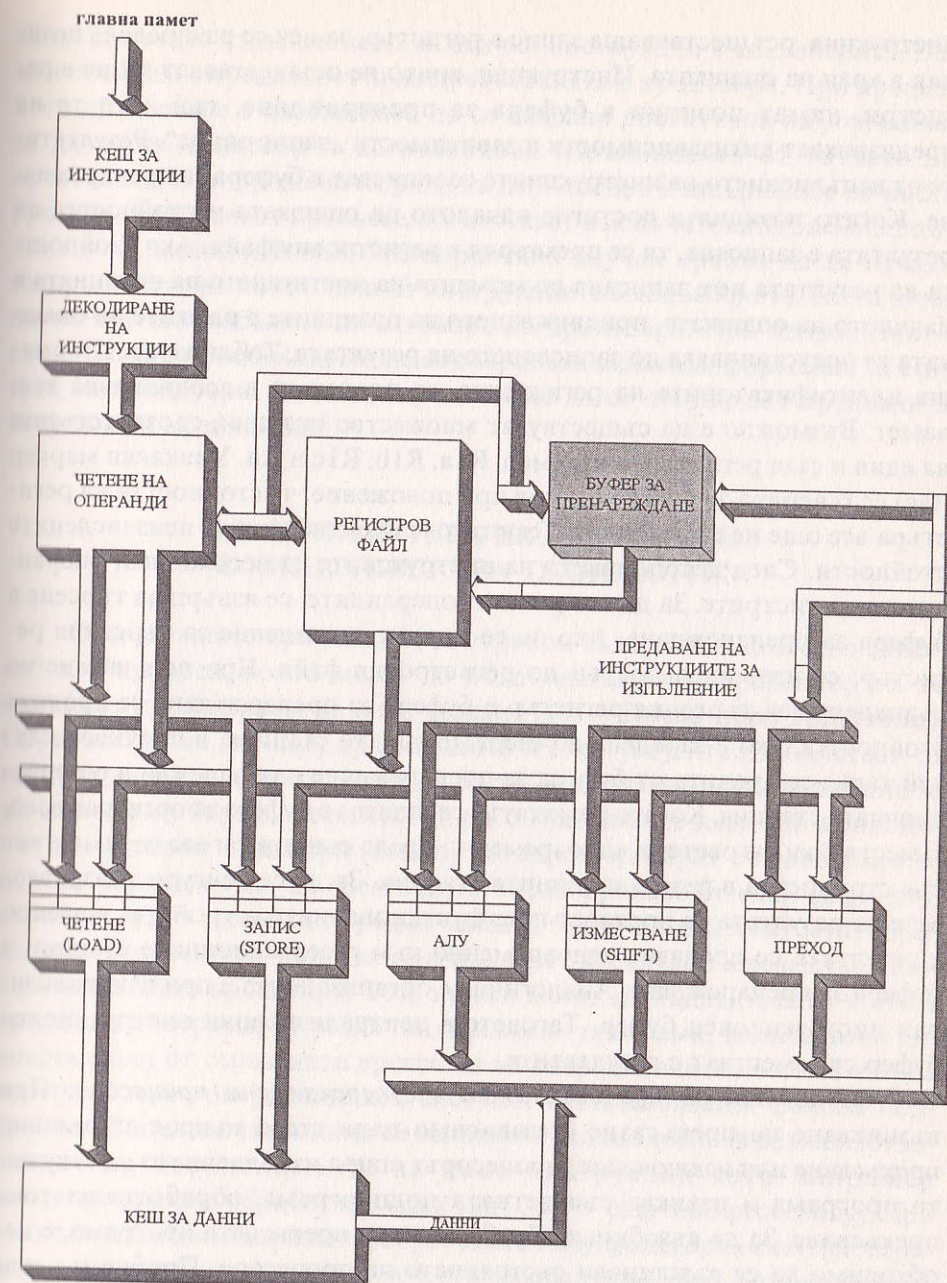
Очевидно е невъзможно да се използват все нови и нови регистри при изпълнението на програмата. От практическа гледна точка не е възможно да се имплементира много голям регистров файл, тъй като това увеличава сложността на хардуера, усложнява обработката на прекъсванията и т.н. Решението е да се използва временно преименуване на регистрите. $R1$ може временно да бъде преименуван на $R1a$, $R1b$, $R1c$ и т.н., когато се използва многократно в програмата.

ADD R1a, R2a, R4a

ADD R2b, R1a, 1

ADD R1b, R4, R5

Преименуването на регистрите може да елиминира антизависимостите и зависимости „запис-запис“. То се реализира апаратно, като създаването и унищожаването на инстанции на регистри се прави непрекъснато в хода на изпълнението на програмата. Преименуването на регистрите може да се осъществи чрез *буфер за пренареждане* (Reorder buffer). Той представлява опашка с дисциплина на обслужване „пръв влязъл – пръв излязъл“ като позициите са свързани динамично с регистрите за резултата на инструкциите. На фиг. 2.37 е представен суперскаларен процесор с буфер за пренареждане и резервационни станции. При декодирането на



Фиг. 2.37. Суперскаларен процесор с буфер за пренареждане и резервационни станции

инструкция, осъществяваща запис в регистър, за нея се разпределя позиция в края на опашката. Инструкции, които не осъществяват запис в регистри, нямат позиции в буфера за пренареждане, тъй като те не предизвикват антизависимости и зависимости „запис-запис“. Резултатите от изпълнението на инструкциите се записват в буфера за пренареждане. Когато позицията достигне началото на опашката и стойността на резултата е записана, тя се прехвърля в регистровия файл. Ако стойността на резултата не е записана до момента на достигането на позицията в началото на опашката, придвижването на позициите в рамките на опашката се преустановява до записването на резултата. Таблицата, съдържаща идентификаторите на регистрите, се поддържа в асоциативна кеш памет. Възможно е да съществуват множество позиции, съответстващи на един и същ регистър, например, R1a, R1b, R1c и т.н. Уникален маркер (tag) се генерира за всяка позиция при положение, че стойността на регистъра все още не е изчислена. Тези тагове идентифицират неизчислените стойности. След декодирането на инструкциите са необходими операндите от регистрите. За да се прочетат операндите, се извършва търсене в буфера за пренареждане. Ако не се получи съвпадение за търсения регистър, се извършва достъп до регистровия файл. При получаване на съвпадение за търсения регистър в буфера за пренареждане се прочита стойността, ако е заредена в резервационната станция, в противен случай тагът се прочита от буфера за пренареждане и се зарежда в резервационната станция. Когато резултатът се записва в буфера за пренареждане, замествайки таговете, едновременно с това същите тагове се заместват със стойността в резервационните станции. За да се осигури това, таговете на резултата се предават през функционалните устройства и заедно с резултата се предават едновременно към резервационните станции и буфера за пренареждане. Аналогична е организацията и при централизиран инструкционен буфер. Таговете в централизирания инструкционен буфер се заместват с резултатите.

Обработка на прекъсванията в суперскаларни процесори. При възникване на прекъсване (независимо дали става въпрос за външно прекъсване или изключение) процесорът спира изпълнението на текущата програма и извиква съответната подпрограма, обработваща това прекъсване. За да възобнови обработката на прекъснатата програма, е необходимо да се възстанови състоянието на процесора. Проблемът при суперскаларните процесори е, че в момента на прекъсването повече от една инструкция ще бъдат в процес на изпълнение. Времето за отговор на процесора при прекъсване ще стане недопустимо голямо, ако трябва

да се изчака изпълнението на всички инструкции в конвейерите. Важен проблем е прецизността на прекъсването в процесора. При прецизното прекъсване е необходимо да се съхрани достатъчна информация, за да може процесорът да възобнови изпълнението от точката на прекъсване. При промяна на реда на изпълнение и завършване на инструкциите прецизните прекъсвания изискват изключително големи допълнителни разходи. В най-благоприятния случай трябва да се изчака завършването на поне някои от инструкциите в конвейерите, което води до значително забавяне на отговора на процесора. При непрецизното прекъсване не е необходимо да се съхранява цялата информация за статуса на процесора. В някои случаи може да се поддържат прецизни и непрецизни прекъсвания.

2.9. АРХИТЕКТУРА НА ПРОЦЕСОРИ С ХИПЕРНИШКОВА ТЕХНОЛОГИЯ

Съвременните приложения за сървъри съдържат множество „нишки“ или процеси, които могат да бъдат изпълнени паралелно, т.е. те съдържат паралелизъм на ниво нишки (TPL – Thread Level Parallelism). Обработката на (on-line) транзакциите и Web услугите изобилстват от програмни нишки, които могат да бъдат изпълнявани едновременно, за да се повиши бързодействието. Дори съвременните десктоп приложения за настолни компютри (desktop) се характеризират с повишена степен на паралелизъм. Съвременните високопроизводителни (high-end) сървъри, както и тези със средна мощност (mid-range), имат мултипроцесорна архитектура. При тях се изпълняват паралелно множество нишки в множество процесори. Тези нишки могат да бъдат от една и съща приложна програма, от различни приложни програми, изпълнявани едновременно, от сервизните програми на операционната система или от програмите на операционната система, осъществяващи фоновата поддръжка. В последните години се използват редица техники за оползотворяване на паралелизмите на ниво инструкции като например мултипроцесорна обработка в един чип (CMP – chip multiprocessing). При нея се осъществява паралелна обработка от двупроцесорна система, вградена в рамките на един чип. Двата процесора разполагат с пълно множество архитектурни ресурси и могат да споделят вграден кеш в чипа. Осигурени са възможности за изграждане на мултипроцесорни конфигурации на базата на двупроцесорните чипове. Естествено размерът на дву-

процесорния чип е значително по-голям от този на конвенционалния еднопроцесорен чип, което значително оскъпява производството.

Друг модерен подход е *многонишковата обработка*, при която процесорът обработва едновременно множество нишки чрез превключване на изпълнението им. При *многонишковата обработка с времоделене* процесорът превключва между софтуерните нишки на фиксирани интервали от време. При нея могат да възникнат празни слотове за изпълнение, но се минимизират отрицателните ефекти от голямото време за достъп до паметта. *Многонишковата обработка с превключване при събитие* се използва при сървърни приложения, при които две нишки изпълняват подобни задачи и възникват голям брой липси в кеша. Събитията, предизвикващи превключването на нишките, имат голяма латентност и най-често това са липсите в кеша. Недостатък е неефективното използване на ресурсите на процесора.

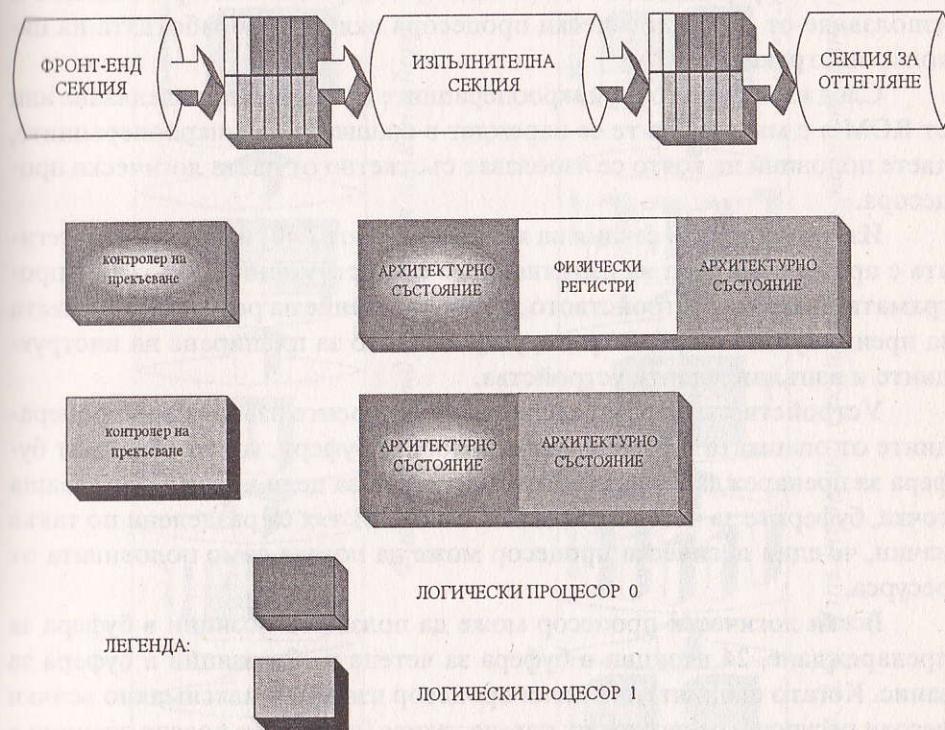
При едновременната многонишкова обработка множество нишки се изпълняват от един процесор без превключване. При нея се постига най-ефективно използване на ресурсите на процесора и най-висока производителност.

При архитектурите с хипернишкова технология един физически процесор изпълнява функциите на два логически процесора, всеки от които има собствено архитектурно състояние и споделя ресурсите на физическия процесор. Следователно процесите и нишките на приложните програми и програмите на операционната система могат да се изпълняват от двата логически процесора както в реална двупроцесорна система, за да се осигури това, инструкциите на двата логически процесора се изпълняват едновременно в общите ресурси на физическия процесор.

Първата имплементация на хипернишковата технология е в микропроцесорната фамилия Xeon™ на Intel® за двупроцесорни и мултипроцесорни сървъри, при които един физически процесор обслужва два логически процесора. Увеличаването на размера на кристала в случая е не повече от 5% [11]. Архитектурният статус на всеки логически процесор включва съдържанието на програмно достъпните регистри, управляващите регистри и регистрите на програмируемия контролер за прекъсване. Всеки логически процесор разполага със свой собствен контролер за прекъсване, като прекъсванията на даден логически процесор се обработват ексклузивно от собствения контролер за прекъсване. Логическите процесори ползват общо всички ресурси на физическия процесор като изпълнителни устройства, кешове, управляваща логика и шини. Повечето от общите ресурси се ползват алтернативно от двата логически

процесора при висок коефициент на справедливост. Друга част от ресурсите са разделени на две половини, като всяка половина се ползва ексклузивно от един от логическите процесори. За оптимално използване на ресурсите, когато единият логически процесор е блокиран поради липси в кеша, погрешно предсказан преход или зависимости между инструкциите, всички ресурси се използват изцяло само от другия логически процесор.

Конвейерът на процесор с хипернишкова технология е изграден от три основни секции: фронт-енд секция (подготовка на инструкциите за изпълнение), изпълнителна секция и финализираща секция за оттегляне на инструкциите (фиг. 2.38).



Фиг. 2.38. Конвейер на процесор с хипернишкова технология

Фронт-енд секцията е ангажирана с предварителното извличане на инструкциите, декодирането им и подготовката им за предаване за изпълнение (фиг. 2.39). Тя съдържа кеша за проследяване, който е кеш за инструкции от първо ниво L1. При липса в кеша за проследяване

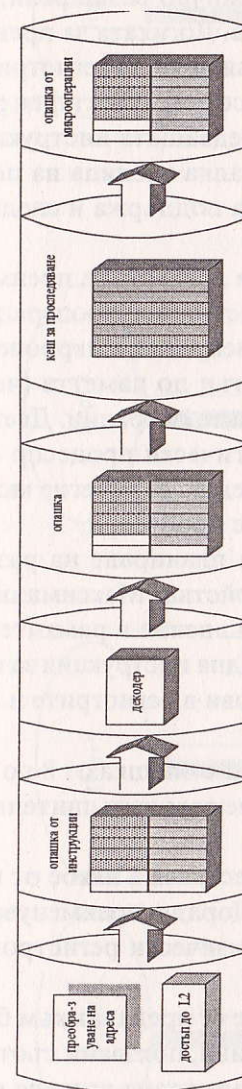
непосредствена близост до кеша за проследяване е разположен ROM с микрокода, който съхранява микрооперациите на комплексните инструкции. За комплексните инструкции кешът за проследяване изпраща указател към ROM, по който се извлича съответната последователност от микрооперации, имплементиращи тази инструкция. Всеки логически процесор разполага с два инструкционни буфера, захранващи фазата на декодиране на конвейера. Структурите за предсказване на преходите са дублирани. Глобалната таблица с историята на преходите е обща, като позициите в нея са маркирани с тагове на идентификатора на логическия процесор. Декодиращата логика приема инструкциите от захранващите буфери и ги декодира в микрооперации. Алтернативното ѝ използване от двата логически процесора включва обработката на няколко инструкции.

След извличането на микрооперациите от кеша за проследяване или от ROM-а с микрокода те се нареждат в опашката на микрооперациите, двете половини на която се използват съответно от двата логически процесора.

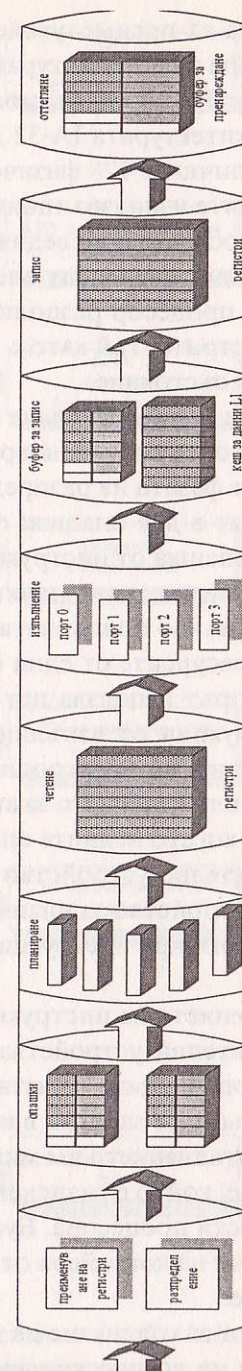
Изпълнителната секция на конвейера (фиг. 2.40) използва стратегията с промяна на реда на изпълнението на инструкциите, заложен в програмата и включва устройството за разпределение на ресурсите, логиката за преименуване на регистрите, устройството за планиране на инструкциите и изпълнителните устройства.

Устройството за разпределение на ресурсите извлича микрооперациите от опашката и разпределя ключовите буфери, които обхващат буфера за пренареждане, физическите регистри за цели числа и за плаваща точка, буферите за четене и за запис. Някои от тях са разделени по такъв начин, че един логически процесор може да ползва само половината от ресурса.

Всеки логически процесор може да ползва 63 позиции в буфера за пренареждане, 24 позиции в буфера за четене и 12 позиции в буфера за запис. Когато единият логически процесор използва максимално всички негови ресурси и се нуждае от повече такива (например повече позиции в буфера за запис), разпределителното устройство временно блокира работата на този логически процесор и започва да обслужва другия логически процесор. Като се ограничава максимално допусимото използване на такива ресурси като ключовите буфери, се осигурява висок коефициент на справедливост на обслужване на двата процесора и се предотвратява възникването на мъртво блокиране.



Фиг. 2.39. Фронт-енд секция на конвейера



Фиг. 2.40. Изпълнителна секция и секция за оттегляне на конвейера

Логиката за преименуване на регистрите преобразува имената на 32-та регистра на архитектурата IA-32 в имена на физическите регистри на процесора. Това дава възможност 8-те универсални целочислени регистри на архитектурата IA-32 да бъдат динамично разширени, за да се използват наличните 128 физически регистра. Логиката за преименуване на регистрите използва таблица на псевдонимите на регистрите, за да може да се проследява последната версия на всеки архитектурен регистър за указване къде се намират операндите за следващата инструкция. Всеки логически процесор разполага с индивидуална таблица на псевдонимите на регистрите, тъй като е необходимо да поддържа и следи своето архитектурно състояние.

Логиката за разпределяне на ресурсите и логиката за преименуване на регистрите обслужват едновременно една и съща микрооперация. След като напуснат фазата на разпределение/преименуване, микрооперациите се разпределят в две опашки: опашка за достъп до паметта (четене-запис) и обща опашка от инструкции за останалите операции. Двете опашки са разделени по такъв начин, че всеки логически процесор може да ползва максимално половината опашка с оглед да се избегне монополизирането на ресурсите от един от логическите процесори.

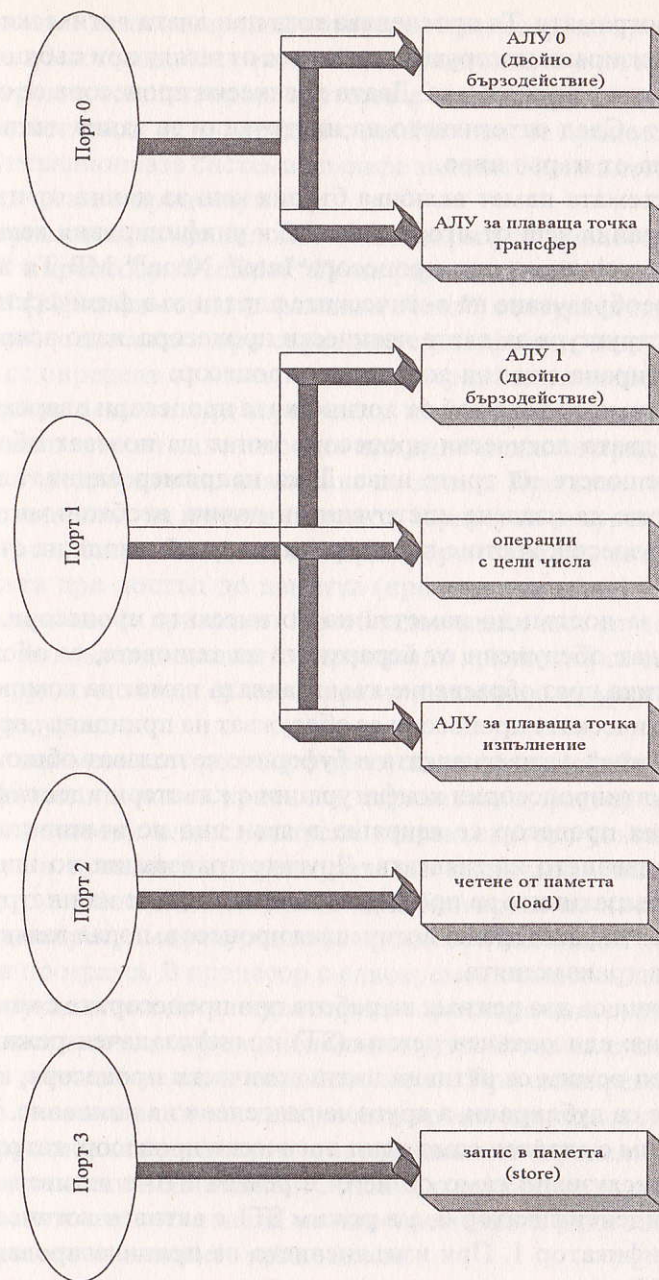
Процесорът използва пет устройства за планиране на различните типове инструкции от изпълнителните устройства. Максимално могат да бъдат планирани 6 микрооперации за изпълнение в рамките на един машинен цикъл. Решението за активиране на една инструкция за изпълнение се взема, когато нейните операнди са готови в регистрите и съответното изпълнително устройство е свободно.

Всяко устройство за планиране разполага с опашка от 8 до 12 позиции, от която избира инструкция за изпращане към изпълнителните устройства.

Изпълнението на инструкцията се осъществява в някое от паралелните изпълнителни устройства (фиг. 2.41). Поради преименуването на регистрите, операндите се четат от общия физически регистров файл и резултатът също се записва в него.

След изпълнението им микрооперациите се предават към буфера за пренареждане, който е разделен на две половини, ползвани съответно от двата логически процесора. Буферът за пренареждане разделя изпълнителната секция на конвейера от неговата финална секция за оттегляне на инструкциите.

Логиката за оттегляне на инструкциите актуализира архитектурното състояние на логическия процесор съобразно реда на инструкциите,



Фиг. 2.41. Изпълнителни устройства в Intel® NetBurst™ микроархитектура

заложен в програмата. Тя проследява кога при двата логически процесора има финализирани инструкции, които се оттеглят при съблюдаване на нареждането им в програмата. Двата логически процесора се обслужват алтернативно. След оттеглянето на инструкции за запис данните се записват в кеша от първо ниво.

Подсистемата памет включва бързия кеш за данни от първо ниво L1, унифицирания кеш от второ ниво L2 и унифицирания кеш от трето ниво L3, наличен само при процесора Intel® Xeon™ MP. Тя включва и буфера за преобразуване на логическите адреси във физически. Този буфер е обща структура за двата логически процесора, като всяка позиция в него е маркирана с таг на логическия процесор.

Независимо от това кой от логическите процесори е заредил данните в кеша и двата логически процесора могат да ползват общо всички позиции в кешовете на трите нива. Така например единият логически процесор може да извлече инструкции и данни, необходими на другия логически процесор, което е често срещан случай в кода на сървърните приложения.

Заявки за достъп до паметта на логическите процесори, които не могат да бъдат обслужени от йерархията на кешовете, се обслужват от шинната логика чрез обръщение към главната памет на компютъра. Заявките на логическите процесори се обслужват на принципа „първ влязъл – първ обслужен“, като опашката и буферите се ползват общо.

При мултипроцесорни конфигурации с клъстери идентификаторът на логическия процесор се изпраща в явен вид по външната шина по време на подаването на заявката. Другите транзакции по шината като например транзакциите за предварително извличане на инструкции, наследяват идентификатора на логическия процесор, подал заявката, която е генерирала транзакцията.

Възможни са два режима на работа при процесорите с хипернишкова технология: еднозадачен режим (ST) и многозадачен режим (MT). В многозадачен режим са активни двата логически процесора, като някои от ресурсите са дублирани, а други са разделени на половина. При еднозадачен режим е активен само един логически процесор, като ресурсите се ползват ексклузивно само от него: в режим ST0 е активен логически процесор с идентификатор 0, а в режим ST1 е активен логически процесор с идентификатор 1. При изпълнението на привилегированата инструкция HALT процесорът преминава от многозадачен режим MT в еднозадачен режим ST0 или ST1 в зависимост от това, кой логически процесор е изпълнил HALT. В еднозадачен режим ST0 или ST1 сигнал за

прекъсване към дезактивирания логически процесор ще предизвика прекъсване в многозадачен режим. За операционната система и приложенияте програми компютърна система с процесори с хипернишкова технология изглежда като система с двойно повече процесори от физическите. Операционната система планира задачите и нишките за изпълнение от логическите процесори.

Хипернишковата технология се поддържа от процесорната фамилия Xeon™ и процесора Pentium™ IV на фирмата Intel®. По данни на фирмата Intel® [11] процесорната фамилия Xeon™ осигурява изключително висока производителност при сървърните платформи, която в значителна степен се определя от използваната хипернишкова технология.

Спекулативните предварителни изчисления представляват техника за намаляване на латентността при еднонишкови приложения чрез интензивно предварително извличане на данни. При тази техника се използва потенциалът на многонишковите хардуерни ресурси на процесора, за да реализира неявния нишков паралелизъм, като се скрива латентността при достъп до паметта (при липси в кеша) от критичния път в изпълняваната програма. Преобладаващата част от деск-топ приложенията за конвенционалните персонални компютри са еднонишкови и при тях трудно може да се оптимизира използването на многонишковите апаратни ресурси.

При съвременните процесори пропускателната способност на тракта процесор-памет се оказва критичен, ограничаващ фактор за постигане на висока производителност въпреки непрекъснатото усъвършенстване на кешовете и техниките за предварително извличане. Този проблем се решава, като в многозадачен режим при блокиране на инструкционния поток на една програма ресурсите на процесора изпълняват инструкциите на друга програма. В процесор с едновременна многонишкова технология инструкциите на логическите процесори (наричани също хардуерни нишки) се изпращат към функционалните устройства на суперскаларния процесор в рамките на един машинен цикъл. При едновременната многонишкова технология се постига висока производителност, т.к. степента на инструкционния паралелизъм се повишава от паралелизмите между независимите нишки в един цикъл. Основният проблем е да стане възможно използването на потенциала на едновременната многонишкова технология при еднонишковите приложения.

Спекулативните предварителни изчисления представляват техника за предварително извличане на данни от кеша чрез използване на нишки. Основната идея е да се натоварят престоиващите многонишкови харду-

ерни ресурси с изпълнението на спекулативни нишки в полза на основната (неспекулативна) нишка. Тези спекулативни нишки се опитват да предизвикат липси в кеша много преди да се осъществи достъп до паметта от неспекулативната нишка с оглед да се маскира латентността на този достъп. Следователно техниката на спекулативните предварителни изчисления осигурява специални механизми за предварително извличане на данни при инструкции за четене (load), предизвикващи непредсказуеми нерегулярни обръщания към паметта или такива, при които съществуват зависимости по данни.

При повечето програми само малък брой инструкции за четене предизвикват големия брой липси в кеша. Те се наричат „нарушители“. При спекулативните предварителни изчисления се конструират слайси (дялове) за предварителни изчисления, т.нар. *р-слайси*, които включват зависимите инструкции, изчисляващи адресите на достъп до паметта на инструкциите – „нарушители“. Те имат малък размер. Обикновено в един слайс се съдържат от 5 до 15 инструкции. Когато определено събитие активира такъв *р-слайс*, се размножава спекулативна нишка за неговото изпълнение. Спекулативно изпълненият *р-слайс* осъществява предварителното извличане за инструкциите-нарушители, които ще бъдат изпълнени по-късно от основната нишка.

Размножаването на спекулативната нишка води до разпределяне на хардуерна нишка, която копира стойностите в регистровия си файл и осигурява за контекста на нишката адреса на първата инструкция в *р-слайса*. Това предотвратява възможността за хазарт между нишките, при който съдържанието на даден регистър се променя от една нишка, преди нишката-дете да го е прочела. Спекулативната нишка използва хардуерната нишка, докато изпълни всички инструкции в *р-слайса*. Спекулативните нишки не могат да актуализират архитектурното състояние на логическия процесор.

Спекулативните предварителни изчисления може да се поддържат изцяло хардуерно, изцяло софтуерно или да се използват комбинирани форми. Динамичните спекулативни изчисления използват хардуерни механизми за селекция на инструкциите-нарушители и генериране на *р-слайси* за предварително извличане на данните. Спекулативните нишки се конструират, разпростират и премахват хардуерно.

Спекулативните предварителни изчисления трябва да осигурят на временно и коректно предварително извличане на данните. Тази техника може да се прилага от всеки процесор, използващ едновременната хипернишкова технология независимо от спецификата на системата инст-

рукции и организацията на конвейерите. Въпреки че е предназначена за еднонишкова обработка, техниката със спекулативните предварителни изчисления има положителен ефект върху производителността и при многонишкова обработка, дори ако се прилага при изпълнението само на една нишка.

2.10. ТЕНДЕНЦИИ В РАЗВИТИЕТО НА СЪВРЕМЕННИТЕ ПРОЦЕСОРИ (ОБОБЩЕНИЕ)

Тенденциите за увеличаване на бързодействието на съвременните процесори включват повишаване на тактовата честота, използване на паралелизми на ниво инструкции и интензивно използване на кешове. Техниките за повишаване на тактовата честота включват суперконвейеризация на изчислителния процес, при която конвейерите имат голям брой фази с малко времетраене на обработката в рамките на една фаза. Високите тактови честоти на процесорите осигуряват висока производителност чрез увеличаване на броя на паралелно обработваните инструкции. Основните проблеми на суперконвейеризираната микроархитектура на съвременните процесори са свързани с блокирането на конвейерите поради грешно предсказани условни преходи, липси в кеша, прекъсвания и др. Целта на техниките и използване на паралелизми на ниво инструкции е да се увеличи броя на инструкциите, изпълнени за един машинен цикъл. Суперскаларните процесори разполагат с множество паралелни изпълнителни устройства, осигуряващи едновременното изпълнение на множество инструкции в рамките на един машинен цикъл. Основното предизвикателство е да се осигури достатъчно голям брой инструкции, които да бъдат изпълнявани едновременно. За тази цел се използват инструкционни прозорци, осигуряващи паралелното изпълнение на инструкциите в ред, определен от зависимостите между тях и различен от този, фиксиран в програмата. В микроархитектурата на съвременните процесори се влагат средства за използване на паралелизми на ниво нишки. За тази цел се прилага хипернишкова технология за едновременна многонишкова обработка и техниката със спекулативни предварителни изчисления, повишаваща степента на инструкционния паралелизъм за еднонишкови приложения.

Критичен проблем при съвременните процесори е, че достъпът до главната памет на компютъра е значително по-бавен от цикъла на процесора. За да се намали негативното влияние на пропускателната спо-

собност на тракта процесор-памет, се осигуряват бързи кешове в непосредствена близост до процесора, които осигуряват бърз достъп до най-често използваните данни и инструкции. С цел оптимално хранване на инструкционните конвейери с инструкции и операнди за тях се осигуряват два отделни кеша, един за инструкциите и един за данните. За съжаление, кешовете са бързи, само когато имат малък капацитет. Ето защо съвременните процесори взаимодействат с йерархия от кешове, организирана по такъв начин, че на най-високото ѝ ниво процесорът взаимодейства непосредствено с малки и бързи кешове с време за достъп много близко до цикъла на ядрото на процесора. Надолу по нивата на йерархията кешовете са с прогресивно увеличаващ се капацитет и естествено с удължено време за достъп, като ниските нива съхраняват по-рядко използваните данни и инструкции. Въпреки това при липси в кешовете се налага процесорът да се обръща към главната памет на компютъра. Липсите в кеша повишават риска за изчерпване на предварително извлечените инструкции, хранващи инструкционните конвейери, както и блокиране на конвейерите поради липсващи операнди. Поради тези причини съвременните процесори използват множество техники за оптимизация на предварителното извличане на инструкции, както и предварителното извличане на данни.

Преобладаващата част от съвременните техники за повишаване на производителността на процесорите предопределят значително увеличаване на размера на силициевия кристал и значително нарастване на консумираната мощност. Тези техники повишават бързодействието на процесора, но не с 100%-ова ефективност, т.е. n -кратното увеличаване на броя на изпълнителните устройства в процесора не води до n -кратно ускорение, тъй като основният ограничаващ фактор е ниската степен на потенциалния паралелизъм, заложен в инструкционните потоци на програмите. Аналогично двукратното ускоряване на тактовата честота не удвоява производителността на процесора поради допълнителните цикли при грешно предсказани преходи, липси в кеша и т.н. Трябва да се има предвид, че повечето целочислени приложения имат ограничено ниво на инструкционен паралелизъм и съществуват значителни трудности при предсказването на хода на изпълнение на програмата.

По данни на Intel® [6] за последното десетилетие производителността на техните процесори се е увеличила от 5 до 6 пъти благодарение на усъвършенствания в тяхната микроархитектура. За същия период размерът на силициевия кристал на процесора се е увеличил 15 пъти, което е три пъти повече от повишаването на бързодействието. Поради значи-

телния прогрес в технологиите са осигурени възможности за интегриране на повече транзистори в чипа (над 55 милиона транзистора при Pentium IV™), така че действителният размер на силициевия кристал не се е увеличил значително. Относителното нарастване на мощността е 18 пъти. Разсейваната мощност при съвременните платформи е на границата на допустимата, така че трябва да се положат усилия за решаването на този проблем.

Глава 3

АРХИТЕКТУРА НА КОМПЮТЪРНИТЕ СИСТЕМИ

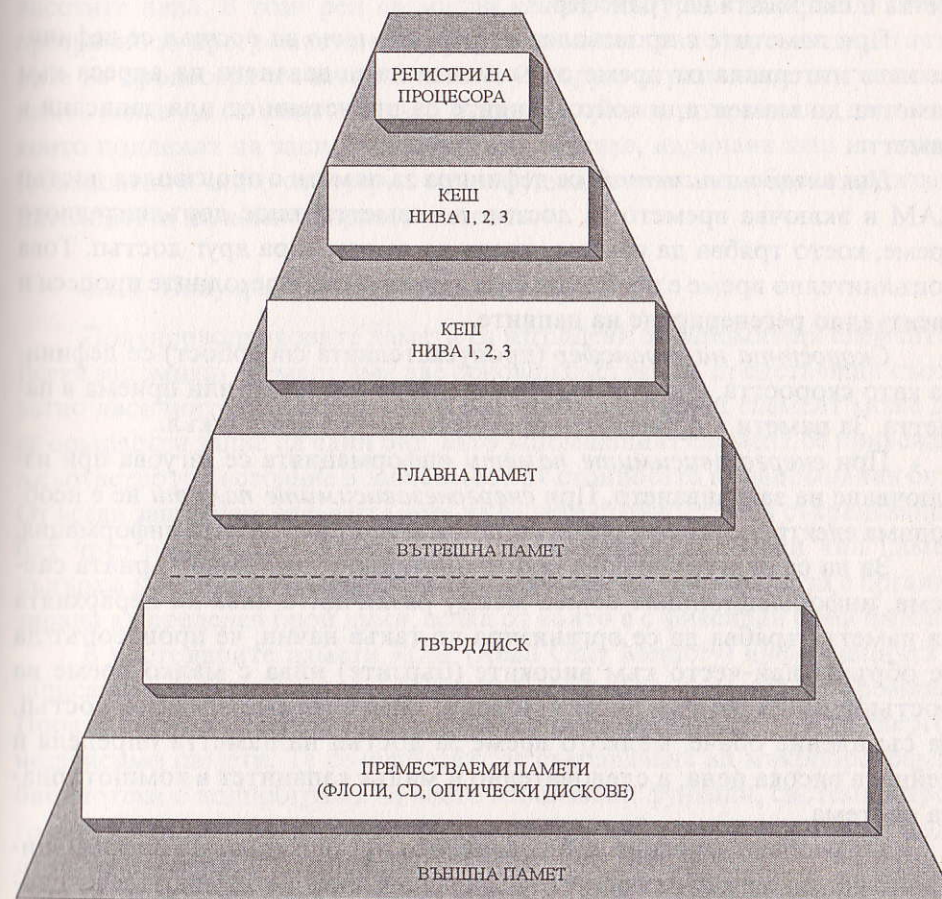
3.1. ЙЕРАРХИЯ НА ПАМЕТА В КОМПЮТЪРНИТЕ СИСТЕМИ

Йерархията на паметта в компютърната система съдържа три нива, които се различават по типа, бързодействието и капацитета на паметта (фиг. 3.1). На най-високото ниво на йерархията на компютърната памет са регистрите на процесора, следвани от кеш на няколко нива (първо ниво L1, второ ниво L2 и трето ниво L3). Следва главната памет на компютъра, която представлява динамична полупроводникова памет с произволен достъп DRAM. Изброените нива включват вътрешната (първичната) памет на компютърната система. Следват нивата на външната (вторичната) памет на компютъра, включваща твърдия диск и едно или повече нива с преместваеми памет като гъвкав магнитен диск (флопи диск), компактен диск CD и др. Информацията във външната памет е структурирана във файлове и записи за разлика от думите и индивидуалните байтове в главната памет. Процесорът има директен достъп до вътрешната памет на компютъра, докато достъпът до външната памет се осъществява посредством В/И модули. Външната памет осигурява разширение на главната памет, известно като виртуална памет.

Времето за достъп както и капацитетът на най-високите нива в йерархията на паметта, са най-малки, като по посока на ниските нива стойностите на тези характеристики се повишават, а цената за един бит спада. Оптималният случай е да се използват само най-бързите памет в компютъра, но поради високата им цена се налага да се използват нивата с по-бавните памет, при което съотношението цена/бит става приемливо. Основната идея е да се организира информационният трансфер по такъв начин, че изпълняваната програма и съответните данни да се поддържат в бързите памет на високите нива.

Основна характеристика на компютърната памет е нейният *капацитет*, отразяващ максималното количество на съхраняваната инфор-

мация. Капацитетът на вътрешните памет се измерват в байтове или думи, докато капацитетът на външната памет се измерва обикновено в байтове. Основната адресируема информационна единица в паметта е думата, но много системи допускат и адресиране на индивидуалните байтове в рамките на думата. Информационната единица за трансфер за вътрешната памет се определя от броя на думите, които едновременно се четат или записват, докато за външната памет информационният обмен използва много по-големи информационни единици, наречени блокове.



Фиг. 3.1. Йерархия на паметта в компютърните системи

Според метода за достъп паметите се разделят на памети с последователен, директен, произволен и асоциативен достъп. Последователният достъп следва модела на магнитната лента, директният достъп – модела на диска. При паметите с произволен достъп RAM (Random Access Memory) индивидуалните думи в паметта директно се четат или записват на съответния адрес. При асоциативния достъп селекцията на информационната единица се осъществява по признак (съдържанието), а не по адреса.

Друга основна характеристика е *бързодействието* на паметта. Тя се определя от три основни параметъра: време за достъп, цикъл на паметта и скоростта на трансфер.

При паметите с произволен достъп *времето за достъп* се дефинира като интервала от време от момента на подаването на адреса към паметта до момента, в който данните са прочетени от или записани в паметта.

Цикълът на паметта се дефинира за памети с произволен достъп RAM и включва времето за достъп до паметта плюс допълнителното време, което трябва да измине, преди да се стартира друг достъп. Това допълнително време е необходимо за затихване на преходните процеси и евентуално регенериране на данните.

Скоростта на трансфер (пропускателната способност) се дефинира като скоростта, с която информацията се подава от или приема в паметта. За памети с произволен достъп RAM тя е един цикъл.

При *енергозависимите памети* информацията се загубва при изключване на захранването. При *енергонезависимите памети* не е необходима електрическа енергия за поддържане на съхранената информация.

За да се осигури висока производителност на компютърната система, информационният обмен между различните нива на йерархията на паметта трябва да се организира по такъв начин, че процесорът да се обръща най-често към високите (бързите) нива с малко време на достъп и по-рядко към ниските (бавни) нива с голямо време за достъп. За съжаление обаче, малкото време за достъп на паметта определя и нейната висока цена, а следователно и малък капацитет в компютърната система.

От особено значение за оптималното организиране на информационния обмен между различните йерархични нива на компютърната памет е *принципът на локалността на достъпите* [13]. По време на изпълнението на програмата достъпите на процесора до паметта имат тенденцията да се съсредоточават (клъстерират) в определена област на

паметта. В общия случай програмите съдържат цикли и подпрограми. След като веднъж цикълът или подпрограмата са заредени в главната памет, следват повтарящи се достъпи до ограничено множество инструкции. Аналогично, при обработката на масиви от данни или таблици се осъществява достъп до клъстерирано множество от данни. В рамките на малък временен интервал процесорът използва фиксирани клъстери от достъпи до паметта, докато за голям временен интервал тези клъстери се променят.

Организацията на данните в йерархията на паметта поддържа нисък процент на достъпите до ниските нива и висок процент на достъпите до високите нива. В този ред на мисли кешът осигурява междинно ниво, прозрачно за програмиста, за информационния трансфер между регистрите на процесора и главната памет. От друга страна част от главната памет може да се използва като буфер, съхраняващ временно данните, които подлежат на запис в диска. Тази техника, наричана кеш на диска (обикновено е чисто софтуерна техника), значително подобрява производителността на компютърната система.

3.1.1. Полупроводникови памети

Полупроводниковите памети са изградени от запомнящи елементи. Всеки запомнящ елемент има две основни състояния, представящи съответно двоичните стойности 0 и 1. Във всеки запомнящ елемент може да се осъществи запис на един бит, като запомнящият елемент се привежда в съответното състояние в зависимост от стойността на запомняния бит. От всеки запомнящ елемент може да се чете стойността на съхранявания бит чрез индициране на състоянието на елемента. Всеки чип памет съдържа матрица от запомнящи елементи, като информацията е организирана в определен брой думи, всяка от които е с фиксиран брой битове.

В постоянните памети ROM (Read Only Memory) информацията е записана още в процеса на производството и не подлежи на промяна. Поради серийното производство имат ниска цена и освен това са енергонезависими памети. Те се използват за съхраняване на микропрограми, библиотеки с подпрограми за често използвани функции, системни програми, таблици с функции и в общия случай информация, която не подлежи на промяна в компютъра.

В случаите, когато са необходими по-малко на брой чипове, се използват *програмируеми постоянни памети* PROM (Programmable ROM). Те също са енергонезависими и записът в тях също се осъществява ед-

нократно, но от потребителя след процеса на производството. Записът на информацията в PROM се осъществява от специални устройства, наречени програматори.

Съществува група памет, които се използват преобладаващо за четене и също са енергонезависими:

- *Изтриваема програмируема постоянна памет EPROM* (Erasable Programmable ROM), при която информацията може да се изтрива и електрически да се записва нова информация. Преди записва обаче се налага наличната информация да бъде изтрита чрез експозиция на ултравиолетова светлина. EPROM е по-скъпа памет от PROM, но осигурява възможност за многократна (ограничен брой пъти) промяна на съдържаната информация.

- *Електрически изтриваема програмируема постоянна памет EEPROM* (Electrically Erasable Programmable ROM), при която информацията може да се променя произволен брой пъти, без да се налага предварително изтриване на старата информация. Времетраенето на записва е значително по-голямо от времетраенето на четенето. EEPROM е по-скъпа памет от EPROM и се отличава с по-ниска степен на интеграция.

Светкавичната (flash) памет е полупроводникова памет с много висока скорост на препрограмиране. Използва се електрическа технология за изтриване, като цялата флаш памет може да бъде изтрита за много кратко време (няколко секунди). Тя представлява преместваема памет и може да се използва многократно.

При паметите с произволен достъп RAM (Random Access Memory) индивидуалните думи в паметта директно се четат или записват на съответния адрес. Тя е енергозависима, като при прекъсване на захранването информацията се изгубва. Следователно паметта RAM може да съхранява временно информацията.

Технологията на RAM може да бъде статична или динамична. При статичните RAM памет битовете се съхраняват в тригери, т.е. те имат регистрова структура. В общия случай статичните RAM памет са по-бързи от динамичните, но за сметка на това са по-скъпи и имат по-ниска плътност на компонентите (чиповете са с по-ниска степен на интеграция). Динамичните DRAM памет допускат висока плътност на компонентите, което означава, че те осигуряват по-голям капацитет на паметта в рамките на един чип. Освен това имат и по-ниска цена и следователно са предпочитани за имплементиране на главната памет на компютрите.

Динамичните DRAM памет са изградени от запомнящи елементи, съхраняващи информацията под формата на капацитивни заряди. Нали-

чието на капацитивен заряд или отсъствието на такъв се интерпретира като съдържание на бита – съответно двоична 1 или 0. Поради утечка тези капацитивни заряди постепенно се разреждат, така че се налага периодически да се опреснява информацията в паметта. Това опресняване (refresh) се изразява в четене и последващ запис на всяка от позициите в паметта.

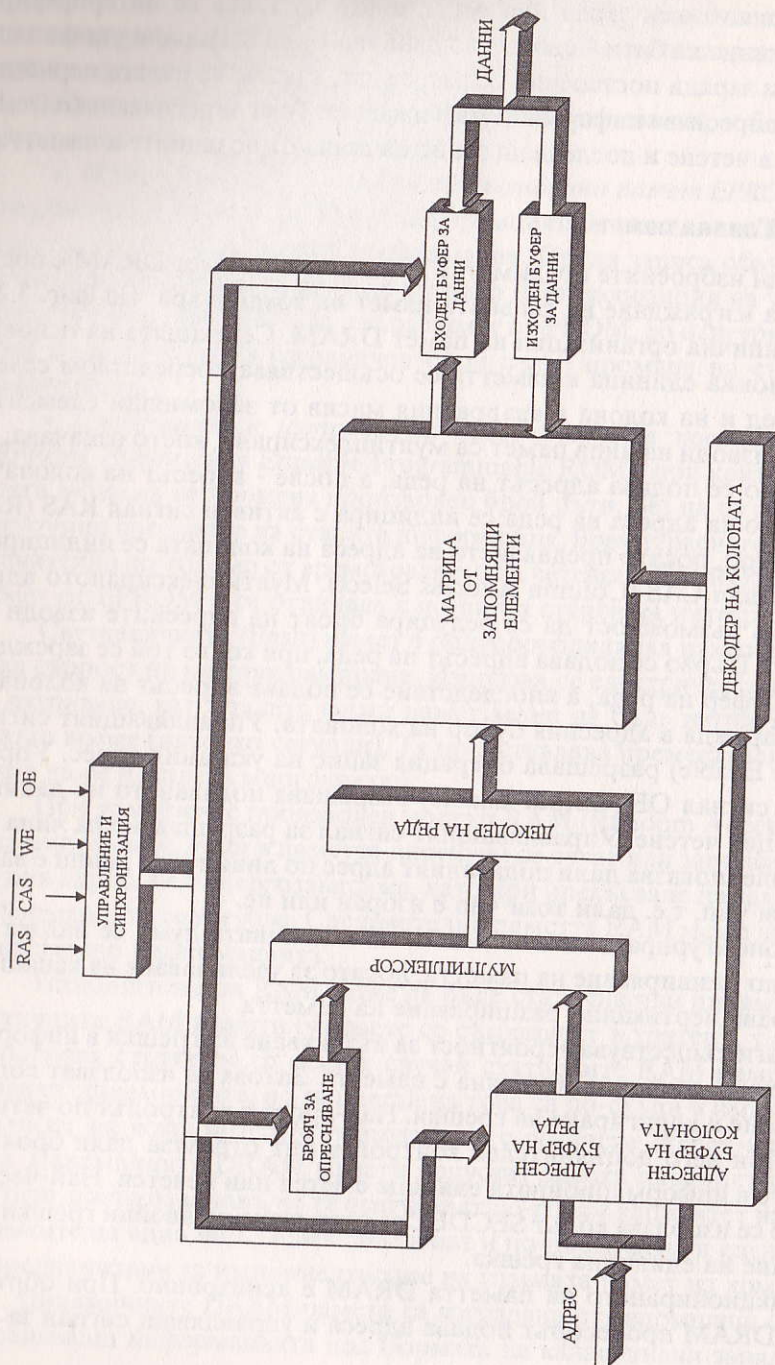
3.1.2. Главна памет

Поради изброените предимства динамичната памет DRAM е предпочитана за изграждане на главната памет на компютъра. На фиг. 3.2 е показана типична организация на памет DRAM. Селекцията на основна информационна единица в паметта се осъществява посредством селекцията на ред и на колона в квадратния масив от запомнящи елементи. Адресните изводи на чипа памет са мултиплексирани, което означава, че по тях първо се подава адресът на реда, а после - адресът на колоната. Предаването на адреса на реда се индицира с активен сигнал RAS (Row Address Select), докато предаването на адреса на колоната се индицира с активен сигнал CAS (Column Address Select). Мултиплексираното адресиране дава възможност да се редуцира броят на адресните изводи на чипа памет. Първо се подава адресът на реда, при което той се зарежда в адресния буфер на реда, а впоследствие се подава адресът на колоната, който се зарежда в адресния буфер на колоната. Управляващият сигнал WE (Write Enable) разрешава операция запис на указания адрес. Управляващият сигнал OE (Output Enable) разрешава подаването на данните при операция четене. Управляващият сигнал за разрешаване на чипа CE (Chip Enable) показва дали подаваният адрес по линиите за данни е валиден за този чип, т.е. дали този чип е избран или не.

За конфигуриране на дължината на машинната дума се налага хоризонтално разширяване на паметта, докато за увеличаване на капацитета ѝ се прави вертикално разширяване на паметта.

Винаги съществува вероятност за възникване на грешки в информацията, съхранявана или обменяна с паметта. Затова се използват кодове за откриване и коригиране на грешки. Най-прост е контролът по четност (parity). При него допълнителен контролен бит отразява дали броят на единиците в информационната единица е четен или нечетен. Най-често в паметите се използва кодът SECDEC за откриване на двойни грешки и за коригиране на единична грешка.

Функционирането на паметта DRAM е асинхронно. При обръщение към DRAM процесорът подава адреса и управляващ сигнал за опе-



Фиг. 3.2. Организация на памет DRAM

рацията, която ще се извърши (четене/запис). След закъснение, равно на времето за достъп, данните са прочетени или записани съответно в паметта. През това време процесорът е принуден да чака, което води до значителна деградация на системната производителност.

При синхронните памети SDRAM информационният обмен с процесора е синхронизиран със системния тактов сигнал, като паметта функционира със скоростта на шината процесор-памет, без да се налагат празни цикли на процесора за изчакване. Отговорът на SDRAM се получава след определен брой цикли, като времето за чакане на отговор се уплътнява от процесора, като превключва за изпълнение друга задача. Паметите SDRAM осигуряват максимално бързодействие в т.нар. експлозивен режим (burst mode), при който с голяма скорост се обменят големи блокове от данни.

RDRAM (Rambus DRAM) осигурява изключително висока скорост на трансфер, като използва асинхронен блок-ориентиран протокол. Паметта RDRAM получава заявките за достъп по високоскоростна шина, съдържаща желан адрес, типа на операцията и броя на байтовете, подлежащи на обмен.

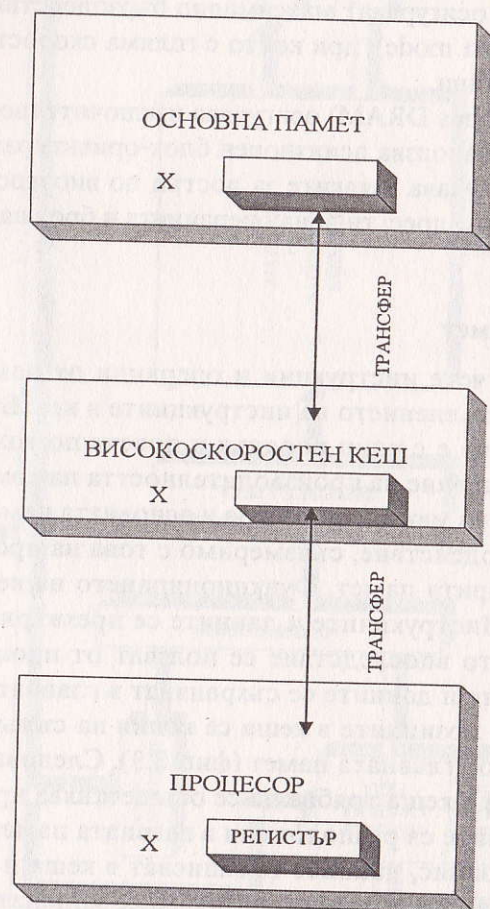
3.1.3. Кеш памет

Процесорът чете инструкции и операнди от паметта и записва резултатите от изпълнението на инструкциите в нея. Бързодействието на процесора обаче е с един порядък и повече по-голямо от това на паметта. За повишаване на производителността на компютъра се прибавя междинно ниво между процесора и основната памет – кеш с малък капацитет и бързодействие, съизмеримо с това на процесора. Думата *cache* означава скрита памет. Функционирането на кеша е прозрачно за потребителя. Инструкциите и данните се прехвърлят от основната памет в кеша, като впоследствие се ползват от процесора. В общия случай програмите и данните се съхраняват в главната памет, така че съдържанието на позициите в кеша са копия на съдържанието на определени клетки от главната памет (фиг. 3.3). Следователно достъпът до информацията в кеша трябва да се осъществява чрез същите адреси, на които данните са разположени в главната памет. Когато процесорът извършва запис, данните се записват в кеша и едновременно с това в главната памет. Алтернативата е да се запишат данните в кеша, а в главната памет по-късно, когато използваната позиция в кеша трябва да бъде използвана за запис на нови данни, т. к. кешът има ограни-

чен капацитет и може да съдържа копия само на ограничен брой клетки от паметта.

Успехът на използването на кеша зависи от характеристиките на програмата и по-точно от нейната степен на локалност. *Принципът на локалността* се отнася както до обръщението към инструкциите, така и до обръщението към данните и може да се разглежда в два аспекта:

- *Локалност във времето.* При нея има голяма вероятност обръщението към определени позиции в паметта да се повторят в близките бъдещи моменти;



Фиг. 3.3. Използване на кеш

- *Локалност в пространството.* Следващата позиция в паметта, към която ще се извърши обръщението е най-вероятно да бъде близко до позицията, към която е осъществено последното обръщението.

Локалността по време е актуална при наличието на цикли, които съдържат предимно малко на брой инструкции. Следователно след като инструкциите в цикъла са заредени веднъж в кеша, те ще бъдат четени многократно, докато се наложи извличането на нови инструкции от паметта (след излизането от цикъла). Аналогична е и ситуацията с данните.

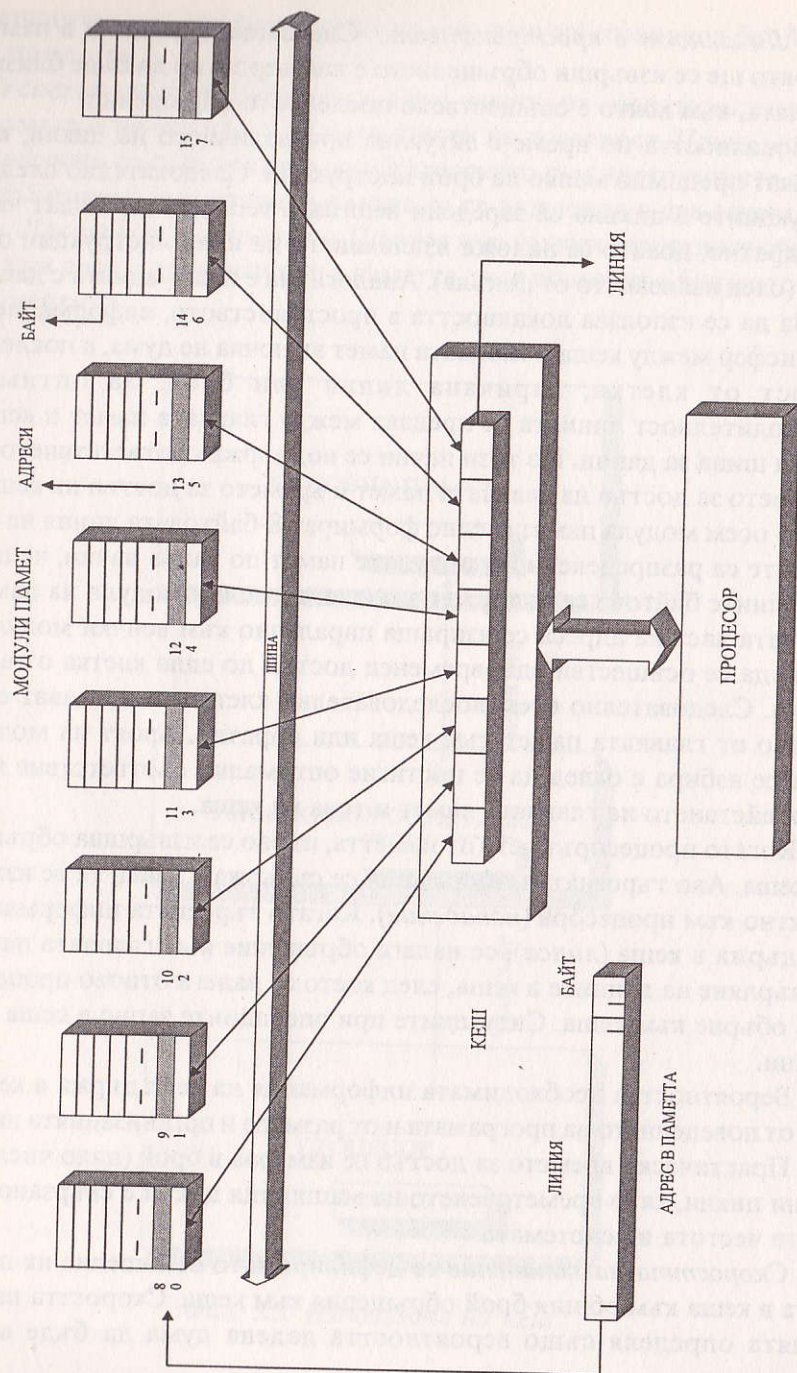
За да се използва локалността в пространството, информационният трансфер между кеша и главната памет включва не дума, а последователност от клетки, наричана линия или блок. За оптимална производителност линията се предава между главната памет и кеша по широка шина за данни. По този начин се поддържа съответствието между времето за достъп на главната памет и времето за достъп на кеша. На фиг. 3.4 осем модула памет заедно формират 8-байтовата линия на кеша. Адресите са разпределени по модулите памет по такъв начин, че последователните байтове се съдържат в последователни модули на паметта. Старшата част на адреса се изпраща паралелно към всички модули памет, за да се осъществи едновременен достъп до една клетка от всички модули. Следователно осем последователни клетки се предават едновременно от главната памет към кеша или обратно. Броят на модулите памет се избира с оглед да се постигне оптимално съответствие между бързодействието на главната памет и това на кеша.

Когато процесорът чете от паметта, първо се извършва обръщението към кеша. Ако търсената информация се съдържа в кеша, тя се изпраща директно към процесора (*попадение*). Когато търсената информация не се съдържа в кеша (*липса*), се налага обръщението към главната памет за прехвърляне на данните в кеша, след което се налага отново процесорът да се обърне към кеша. Ситуациите при операциите запис в кеша са по-сложни.

Вероятността необходимата информация да се съдържа в кеша зависи от поведението на програмата и от размера и организацията на кеша.

Практически времето за достъп се измерва в брой (цяло число) машинни цикли, като времетраенето на машинния цикъл е свързано с тактовата честота на системата.

Скоростта на попадение се дефинира като отношение на попаденията в кеша към общия брой обръщания към кеша. Скоростта на попаденията определя също вероятността дадена дума да бъде в кеша.



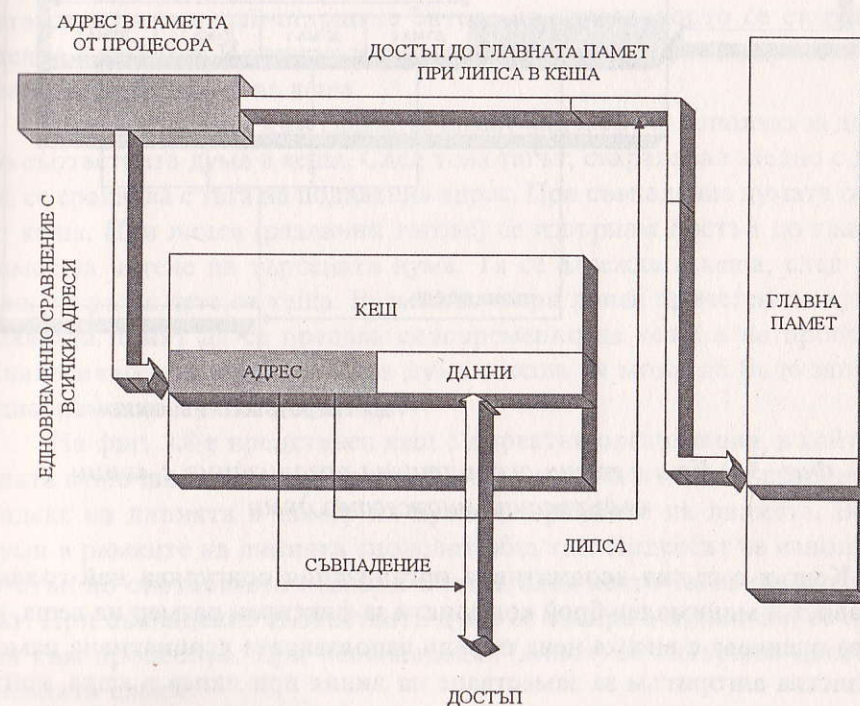
Фиг. 3.4. Съвързване на кеша с главна памет, изградена от множество модули

Скоростта на възникване на липси в кеша се дефинира като единица минус скоростта на попаденията.

Организация на кеша. Процесорът се обръща към кеша чрез адреса, по който се обръща и към главната памет. Следователно организацията на кеша трябва да използва този адрес, за да открие данните при попадение или да сигнализира на процесора, че е възникнала липса в кеша.

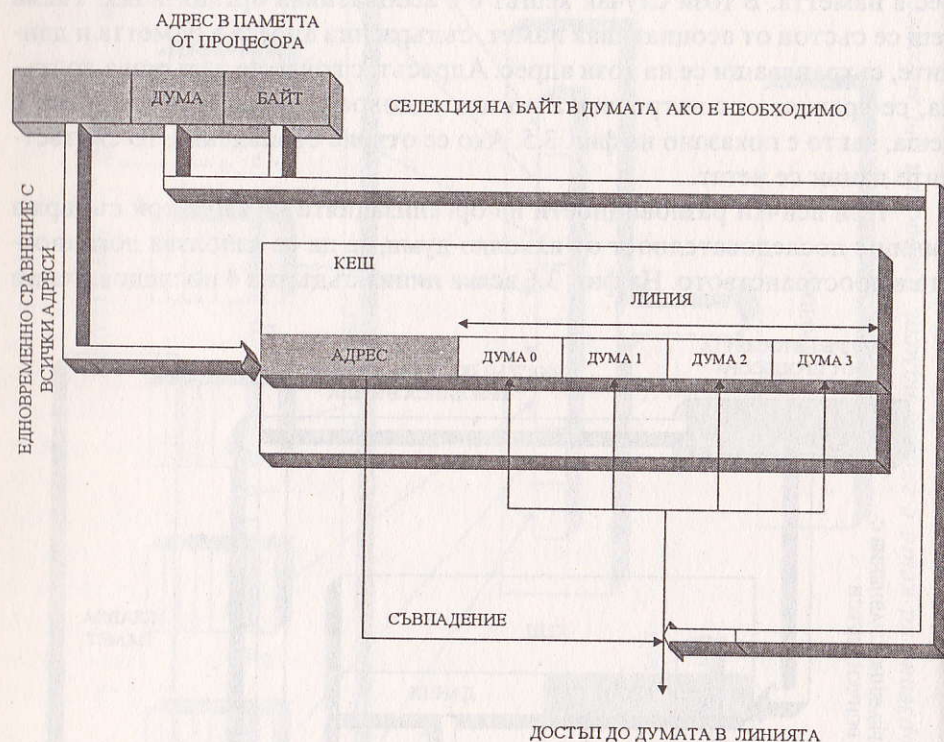
Кеш с пълна асоциативна организация. Най-логичният начин за организирането на кеша е в него да се съхраняват данните и техният адрес в паметта. В този случай кешът е с асоциативна организация. Такъв кеш се състои от асоциативна памет, съдържаща адреса в паметта и данните, съхраняващи се на този адрес. Адресът, с който се извършва достъп, се сравнява от вътрешната логика едновременно с всички адреси в кеша, както е показано на фиг. 3.5. Ако се открие съвпадение, то съответните данни се четат.

При всички разновидности на организацията на кеша той съдържа линии с последователност от няколко думи, за да се използва локалността в пространството. На фиг. 3.6 всяка линия съдържа 4 последователни



Фиг. 3.5. Кеш с асоциативна организация

думи, всяка от думите с по 4 байта. Младшата част на адреса избира байта, следващата част избира думата и останалите битове от адреса служат за сравнение с адреса в кеша. Цялата линия се обменя между кеша и главната памет в рамките на една транзакция, разбира се ако ширината на шината за данни между кеша и главната памет позволяват това. Необходимо е да бъде осигурена възможност да се извършва селекция на индивидуалните байтове в рамките на думата. Най-младшите битове на адреса се използват за идентифициране на байта.



Фиг. 3.6. Кеш с пълна асоциативна организация с линия, съдържащи множество думи

Кешът с пълна асоциативна организация осигурява най-голяма гъвкавост и минимален брой конфликти за фиксиран размер на кеша, но той се отличава с висока цена поради използваната асоциативна памет. Тя изисква алгоритъм за заместване на линия при липса в кеша, който трябва да бъде заложен хардуерно, за да се осигури високо бързодей-

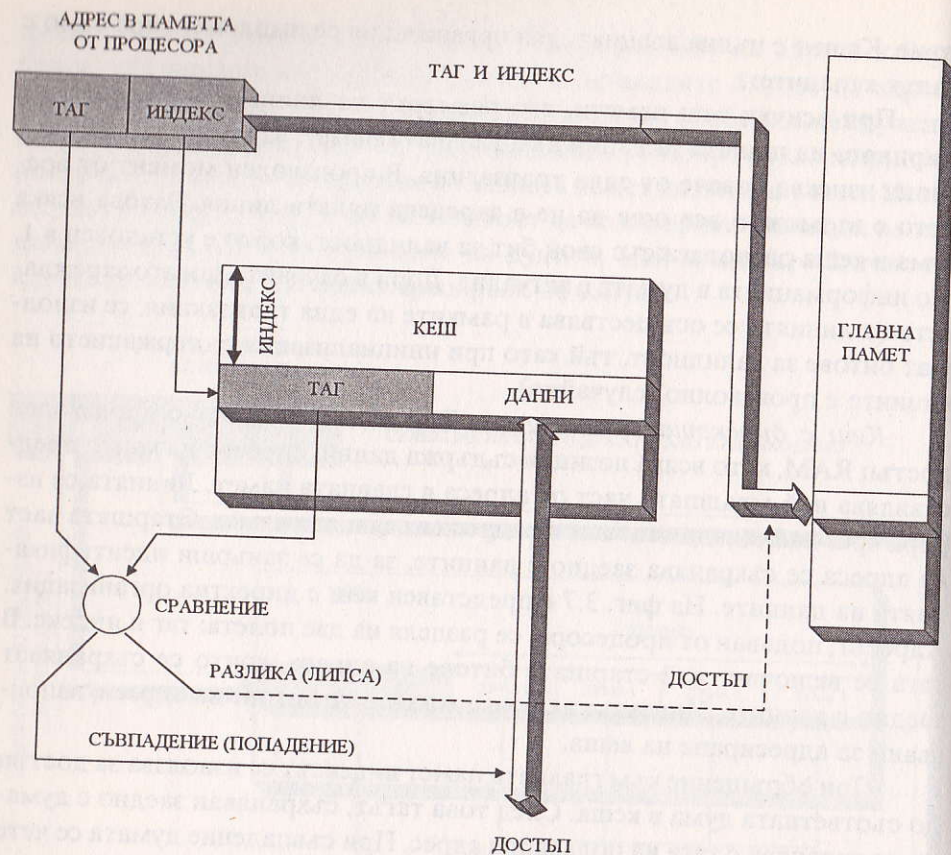
ствие. Кешът с пълна асоциативна организация се имплементира само с малък капацитет.

При всички кеш памети, ако размерът на линията е по-голям от ширината на шината за данни към главната памет, зареждането на една линия изисква повече от една транзакция. В произволен момент от времето е възможно все още да не е заредена цялата линия. Затова всяка дума в кеша разполага със свой бит за валидност, който е установен в 1, ако информацията в думата е актуална. Дори в случаите, когато зареждането на линията се осъществява в рамките на една транзакция, се използват битове за валидност, тъй като при инициализация съдържанието на линиите е произволно (случайно).

Кеш с директна организация. Той съдържа памет с произволен достъп RAM, като всяка позиция съдържа данни, адресът на които представлява най-младшата част от адреса в главната памет. Линията се избира чрез най-младшата част от адреса в главната памет. Старшата част на адреса се съхранява заедно с данните, за да се завърши идентификацията на данните. На фиг. 3.7 е представен кеш с директна организация. Адресът, подаван от процесора, се разделя на две полета: таг и индекс. В тага се включват най-старшите битове на адреса, които се съхраняват заедно с данните. Индексът включва младшите битове на адреса, използвани за адресиране на кеша.

При обръщение към главната памет индексът се използва за достъп до съответната дума в кеша. След това тагът, съхраняван заедно с думата, се сравнява с тага на подавания адрес. При съвпадение думата се чете от кеша. При липса (различни тагове) се извършва достъп до главната памет за четене на търсената дума. Тя се зарежда в кеша, след което процесорът я чете от кеша. Възможно е при липса прочетената дума от главната памет да се предава едновременно на кеша и на процесора. Аналогично при запис на нова дума в кеша тя може да бъде записана едновременно и в главната памет.

На фиг. 3.8 е представен кеш с директна организация, в който линията включва множество думи. Адресът в главната памет включва таг, индекс на линията и номер на думата в рамките на линията. Всички думи в рамките на линията споделят общ таг. Индексът се използва за достъп до съответната позиция в кеша, след което таговете се сравняват. При съвпадение съответната дума се избира в линията и се изпраща към процесора. При несъвпадение (липса) се извършва достъп до главната памет.

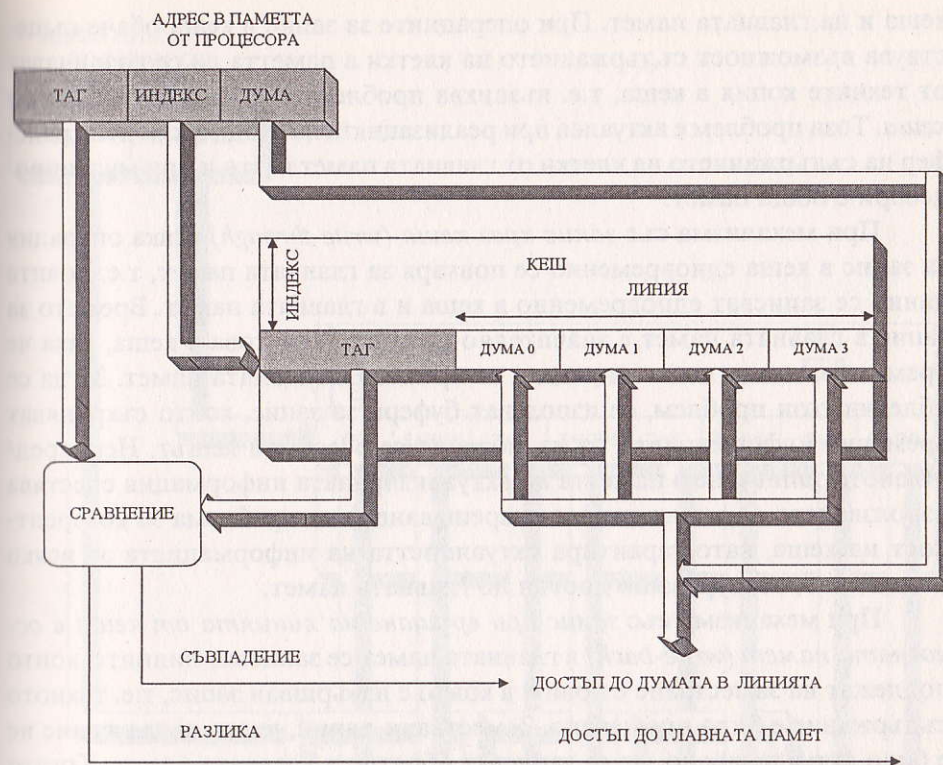


Фиг. 3.7. Кеш с директна организация

При кеша с директна организация, линии с един и същ индекс в главната памет ще бъдат съхранявани в една и съща линия в кеша. Следователно само линии с различни индекси могат да бъдат съхранявани в кеша.

Кешът с директна организация има проста апаратна реализация и високо бързодействие. Основен недостатък е, че производителността пада значително при серия от последователни обръщания на процесора към клетки с едни и същи индекси.

Кеш с асоциативна организация с множества. Той съдържа ограничен брой линии с еднакви индекси и различни тагове. Затова се счита като компромисно решение между кеша с напълно асоциативна организация и кеша с директна организация. На фиг. 3.9 е представен кеш с асоциативна организация с множества. Кешът е разделен на множества от линии. Кеш, който съдържа 4 линии във всяко множество, се нарича 4-



Фиг. 3.8. Кеш с директна организация, при който линията съдържа множество думи

пътно асоциативен. Броят на линиите във всяко множество определя асоциативността на кеша или размера на множеството. Всяка линия във всяко множество съдържа таг, който заедно с индекса (номера на множеството) идентифицира линията. Първо, индексът от адреса, подаван от процесора, се използва за достъп до множеството. После всички тагове на избраното множество се сравняват в компаратори с подавания таг. При съвпадение се осъществява достъп до избраната линия, в противен случай (при липса) се извършва достъп до главната памет. В екстремалния случай, когато размерът на множеството е една линия, кешът става с директна организация. В другия екстремален случай, когато кешът съдържа само едно множество, кешът става напълно асоциативен.

Запис в кеша. При операциите четене съдържанието на кеша не се променя, така че не съществува опасност от разлики в съдържанието на

кеша и на главната памет. При операциите за запис в кеша обаче съществува възможност съдържанието на клетки в паметта да се различават от техните копия в кеша, т.е. възниква проблемът за *кохерентност на кеша*. Този проблем е актуален при реализацията на входно/изходен трансфер на съдържанието на клетки от главната памет както и при мултипроцесори с обща памет.

При механизма със *запис през кеша (write-through)* всяка операция за запис в кеша едновременно се повтаря за главната памет, т.е. новите данни се записват едновременно в кеша и в главната памет. Времето за запис в главната памет е значително по-голямо от това в кеша, така че времето за запис ще се определя от времето в главната памет. За да се облекчи този проблем, се използват буфери за запис, които съхраняват временно информацията, за да може да се освободи кешът. Непосредственото записване в паметта на актуализираната информация спестява използването на механизми за разрешаването на проблема за кохерентност на кеша, като гарантира актуалността на информацията за всяко устройство, осигуряващо достъп до главната памет.

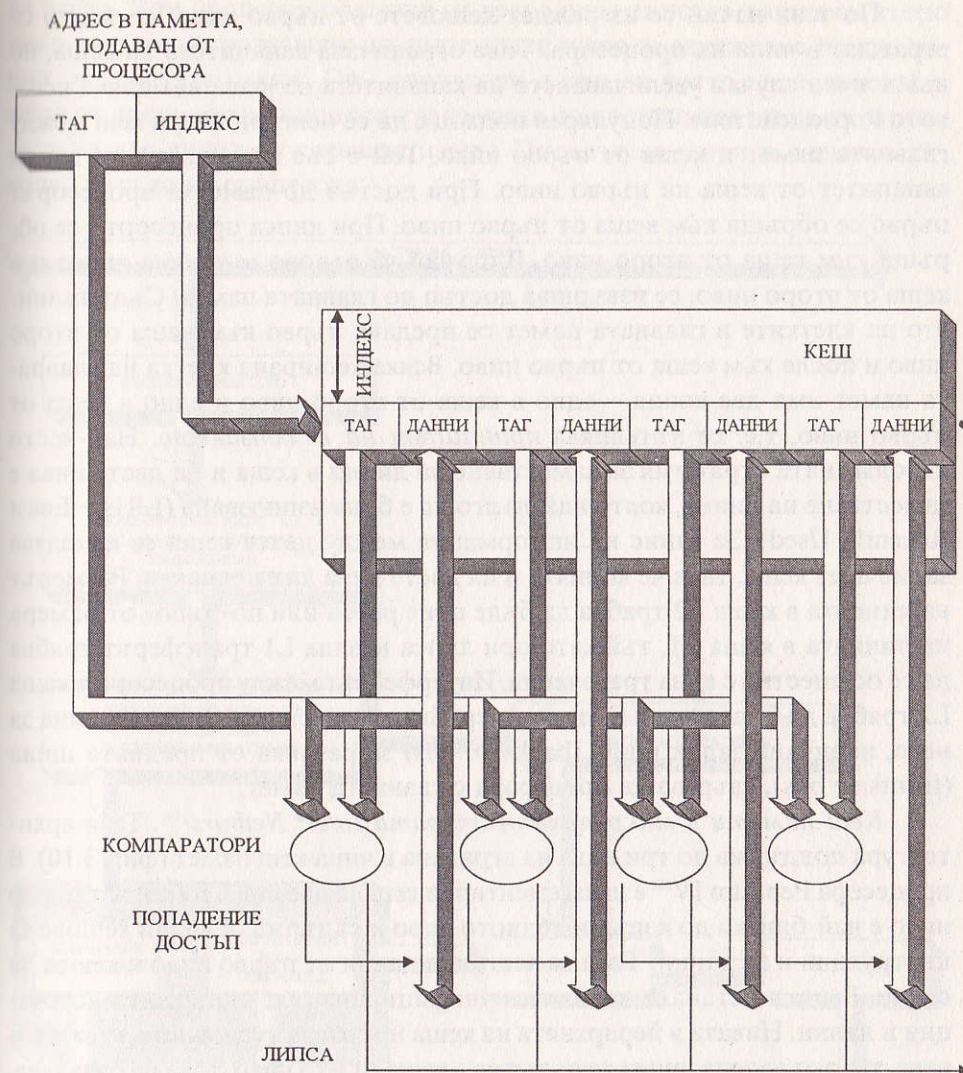
При механизма със *запис при връщане на линията от кеша в основната памет (write-back)* в главната памет се записват линиите, които подлежат на заместване с нови и в които е извършван запис, т.е. тяхното съдържание е било променено. Замествани линии, чието съдържание не е било актуализирано, не се записват обратно в главната памет. Когато се налага трансфер на нова информация в кеша при липса и кешът е пълен, се налага някои от линиите да бъдат заместени с нови.

Алгоритъмът за заместване се реализира изцяло хардуерно и е желателно определянето на линията, която ще бъде заместена, да се вмести в рамките на цикъла на паметта за зареждане на новата линия. Най-често използваните алгоритми за заместване са:

- избор на произволна линия за заместване;
- замества се линия, която е престояла най-дълго в кеша;
- замества се линия, която най-дълго време не е била използвана.

Нива на кеш паметта. Главната памет на компютъра съхранява програмата и данните. Кешът може да бъде организиран по същия начин, т.е. да съхранява инструкции и данни. В този случай той се нарича унифициран кеш. Разделянето на кеша на кеш за инструкции и кеш за данни осигурява следните предимства:

- стратегията за запис се прилага само в кеша за данни (като се приеме, че кодът не е саmomодифициращ се, т.е. инструкциите не се променят);



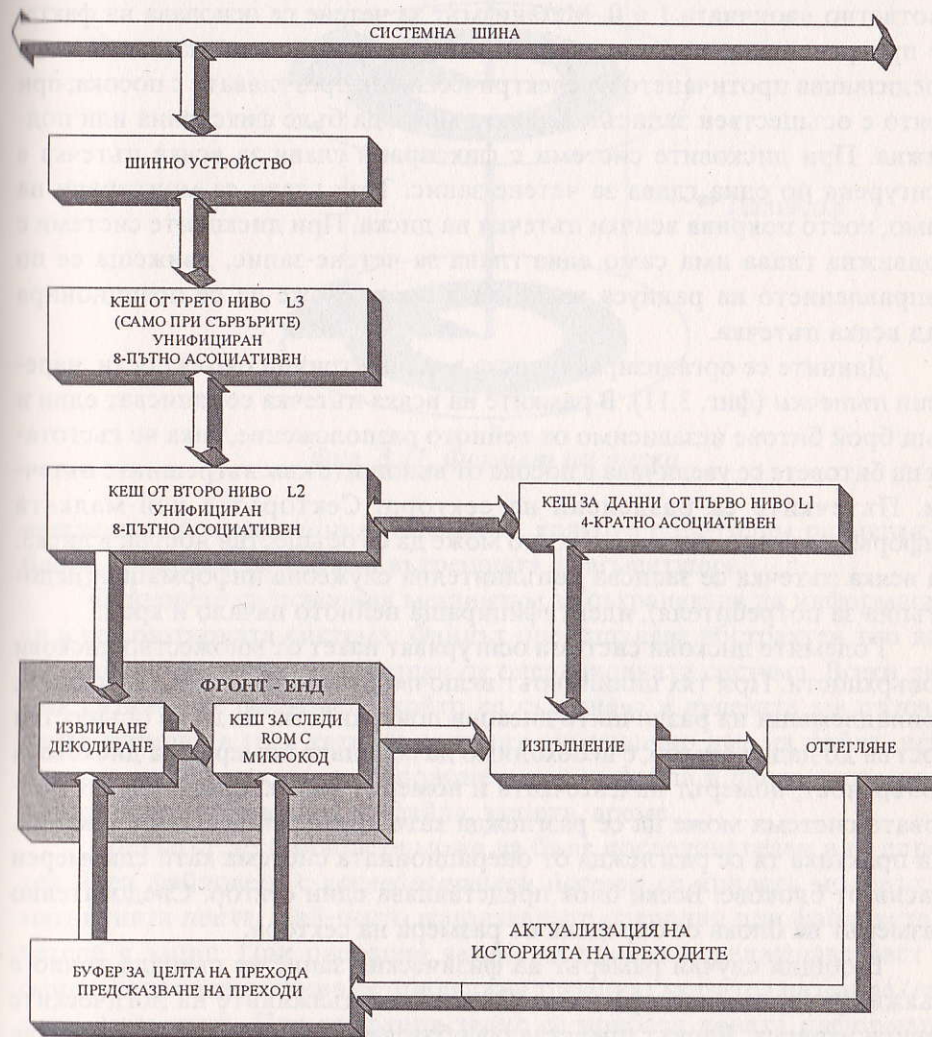
Фиг. 3.9. Кеш с асоциативна организация по множества

- отделни шини осигуряват свързването на двата кеша с процесора, което дава възможност за едновременен трансфер на инструкции и на данни (от особено голямо значение за процесори с конвейерна обработка);
- двата кеша могат да имат различен капацитет, различна организация и различна дължина на линията.

По този начин се изграждат кешовете от първо ниво L1, които се вграждат в чипа на процесора. Това ограничава капацитета на кеша, но във всички случаи увеличаването на капацитета на кеша намалява неговото бързодействие. Популярен подход е да се осигури втори кеш между главната памет и кеша от първо ниво. Той е със значително по-голям капацитет от кеша на първо ниво. При достъп до паметта процесорът първо се обръща към кеша от първо ниво. При липса процесорът се обръща към кеша от второ ниво. В случай че отново възникне липса и в кеша от второ ниво, се извършва достъп до главната памет. Съдържанието на клетките в главната памет се предава първо към кеша от второ ниво и после към кеша от първо ниво. Всяка кеширана клетка на главната памет има две копия – едно в кеша от второ ниво и едно в кеша от първо ниво, т.е. се изпълнява *принципът на включването*. Най-често използваната стратегия за заместване на линии в кеша и на двете нива е заместване на линия, която най-дълго не е била използвана (LRU – Least Recently Used). За запис на информация между двата кеша се използва запис през кеша, така че копията и на двете нива да са еднакви. Размерът на линията в кеша L2 трябва да бъде поне равен или по-голям от рамера на линията в кеша L1, тъй като при липса в кеша L1 трансферът трябва да се осъществи с една транзакция. Интерфейсът между процесора и кеша L2 трябва да бъде много бърз, като трябва да се осигури отделна шина за него, наричана задна шина (backside bus) за разлика от предната шина (frontside bus), свързваща процесора с главната памет.

Кеш памети в микроархитектурата Intel® Netburst™. Тази архитектура поддържа до три нива на вградена в чипа кеш памет (фиг. 3.10). В процесора Pentium IV™ е имплементиран кеш на две нива. Кешът от първо ниво е най-близко до изпълнителното ядро и съдържа отделни кешове за инструкции и за данни. Това са кешът за данни от първо ниво и кешът за следи. Всички останали кешове са унифицирани, т.е. съдържат инструкции и данни. Нивата в йерархията на кеша не спазват условието за включване, т.е. ако дадена линия се съдържа в кеша от *i*-то ниво, това не означава, че тя се съдържа непременно в *i+1*-вото ниво. Кешът за данни от първо ниво има капацитет 8KB и е 4-пътно асоциативен, като рамерът на линията е 64 байта. Кешът за следи може да съдържа до 12 микрооперации и е 8-пътно асоциативен. В Pentium IV™ е осигурено автоматично хардуерно предварително извличане на инструкциите, базирано на предходни достъпи. Кешът от второ ниво има капацитет до 512 KB и е 8-пътно асоциативен при размер на линията 64 байта. Кешът от трето ниво има капацитет до 1MB, също е 8-пътно асоциативен с размер на линията

64 байта. При процесори без кеш от трето ниво липсата в кеша от второ ниво иницира транзакция по системната шина за зареждане на две линии от главната памет. При процесори с кеш от трето ниво липсата в кеша от трето ниво иницира транзакция по системната шина за зареждане на две линии от главната памет. Цикълът на шината включва няколко цикъла на процесора.



Фиг. 3.10. Йерархия на кеш паметите при микроархитектурата Intel® Netburst™

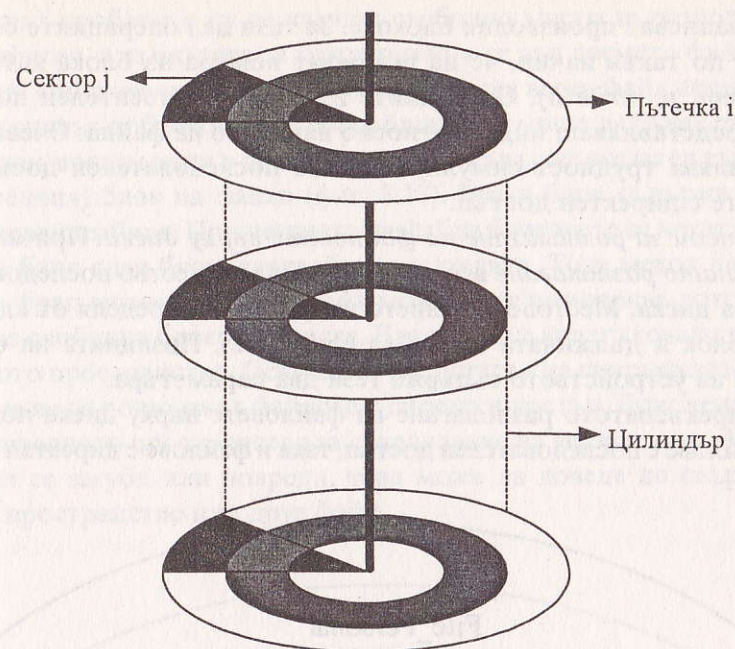
3.1.4. Външна памет

При магнитния диск записът и четенето на информацията се осъществява от глава, която е стационарна, а дискът се върти под нея. Механизмът за запис се основава на факта, че електрическият ток, протичащ през главата, индуцира магнитно поле, което в зависимост от посоката на протичащия ток създава различни магнитни отпечатъци, кодиращи съответно двоичната 1 и 0. Механизмът за четене се основава на факта, че при ротацията на диска под главата, движещото се магнитно поле предизвиква протичането на електрически ток през главата с посока, при която е осъществен записът. Главата може да бъде фиксирана или подвижна. При дисковите системи с фиксирани глави за всяка пътека е осигурена по една глава за четене-запис. Тези глави са монтирани на рамо, което покрива всички пътечки на диска. При дисковите системи с подвижна глава има само една глава за четене-запис, движеща се по направлението на радиуса на диска и която може да се позиционира над всяка пътека.

Данните се организират в диска в концентрични окръжности, наречени *пътечки* (фиг. 3.11). В рамките на всяка пътека се записват един и същ брой битове независимо от нейното разположение, така че гъстотата на битовете се увеличава в посока от външните към вътрешните пътечки. Пътеките са разделени на сектори. Секторът е най-малката информационна единица, до която може да се осъществи достъп в диска. За всяка пътека се записва допълнителна служебна информация (недостъпна за потребителя), идентифицираща нейното начало и край.

Големите дискови системи осигуряват пакет от множество дискови повърхности. При тях цилиндърът включва пътеките с еднакъв радиус, принадлежащи на различните дискови повърхности. За да се осъществи достъп до даден сектор, е необходимо да се зададе номерът на дисковата повърхност, номерът на пътеката и номерът на сектора. В случая дисковата система може да се разглежда като тримерен масив от сектори. На практика тя се разглежда от операционната система като едномерен масив от блокове. Всеки блок представлява един сектор. Следователно размерът на блока се определя от размера на сектора.

В общия случай рамерът на физическия запис не съвпада точно с дължината на логическия запис. Освен това дължините на логическите записи варират. Блокът представлява базовата единица за разпределяне на дисковото пространство. Това означава, че известно място в последния блок може да остане неизползвано (празно). Това явление се нарича



Фиг. 3.11. Формат на диска

вътрешна фрагментация. Очевидно, колкото е по-голям размерът на блока, толкова е по-голяма вътрешната фрагментация.

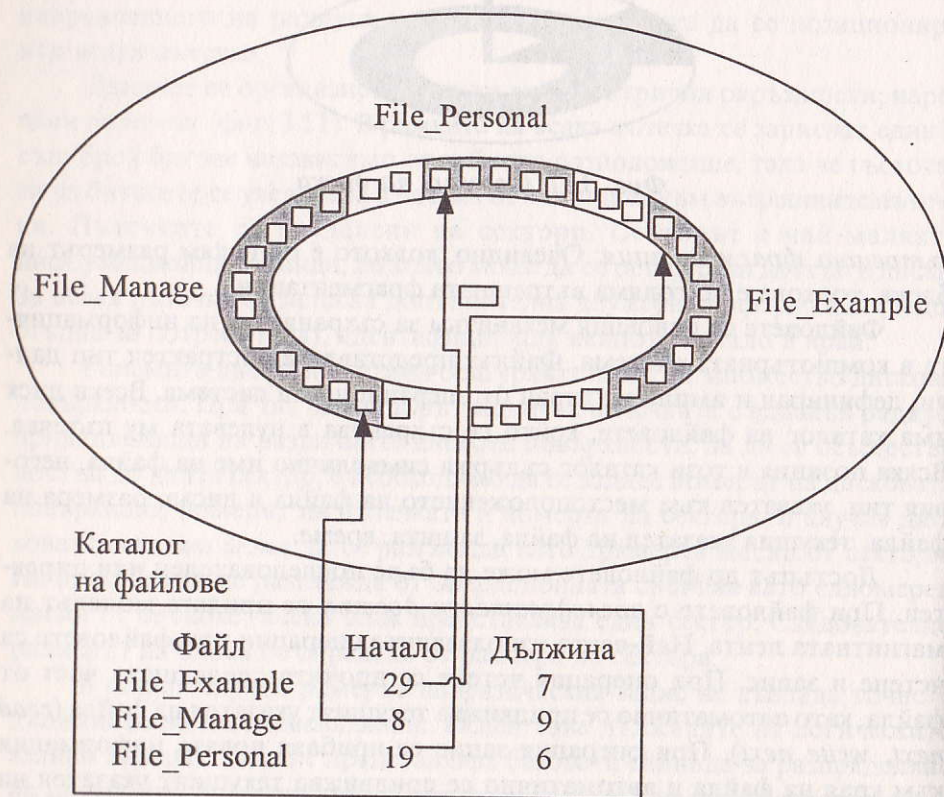
Файловете са основния механизъм за съхраняване на информацията в компютърната система. Файлът представлява абстрактен тип данни, дефиниран и имплементиран от операционната система. Всеки диск има каталог на файловете, който се съхранява в нулевата му пътека. Всяка позиция в този каталог съдържа символично име на файла, неговия тип, указател към местоположението на файла в диска, размера на файла, текущия указател на файла, защита, време.

Достъпът до файловете може да бъде последователен или директен. При файловете с *последователен достъп* се прилага моделът на магнитната лента. Най-често използваните операции при файловете са четене и запис. При операция четене се прочита следващата част от файла, като автоматично се придвижва текущият указател на файла (*read next, write next*). При операция запис се прибавя новата информация към края на файла и автоматично се придвижва текущият указател на файла към новия край на файла. При файловете с *директен достъп* се прилага моделът на диска. При него съществува възможност да се че-

тат или записват произволни блокове. За тази цел операциите се модифицират по такъв начин, че да включват номера на блока като параметър (*read n*, *write n*). Операциите използват относителен номер на блока, представляващ индекс относно началото на файла. Очевидно не представлява трудност симулирането на последователен достъп при файловете с директен достъп.

Методи за разполагане на файловете върху диска. При метода с непрекъснато разполагане всеки файл заема множество последователни адреси на диска. Местоположението на файла се определя от адреса на първия блок и дължината на файла (фиг. 3.12). Позицията на файла в каталога на устройството съдържа тези два параметъра.

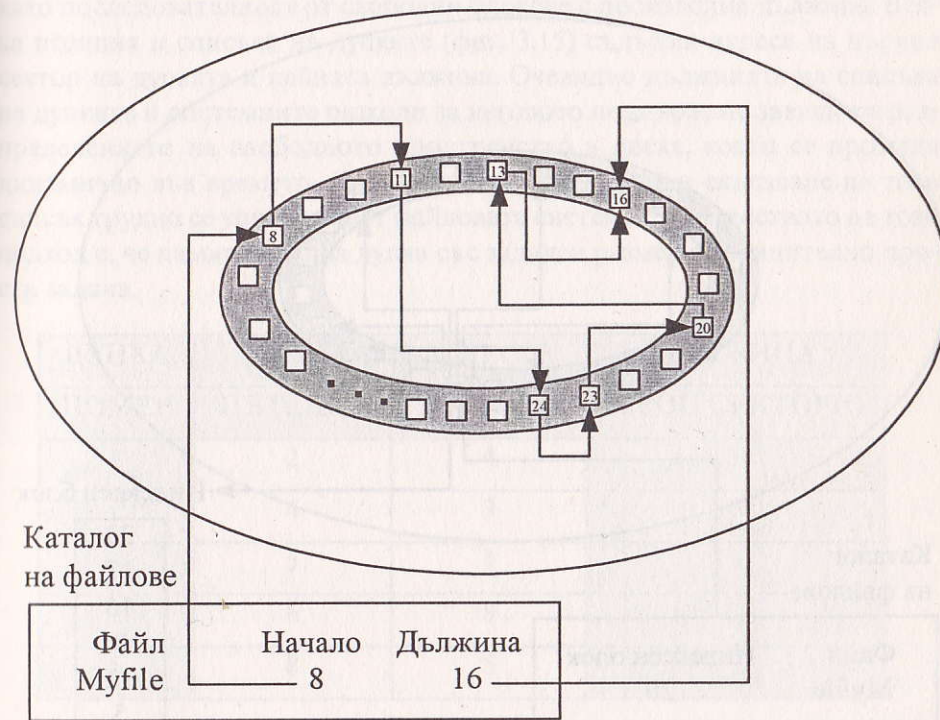
Непрекъснатото разполагане на файловете върху диска поддържа както файлове с последователен достъп, така и файлове с директен достъп.



Фиг. 3.12. Непрекъснато разполагане на файл върху диска

Основният проблем е да се намери свободно място за разполагане на нови файлове, а също така за разширяващи се във времето файлове.

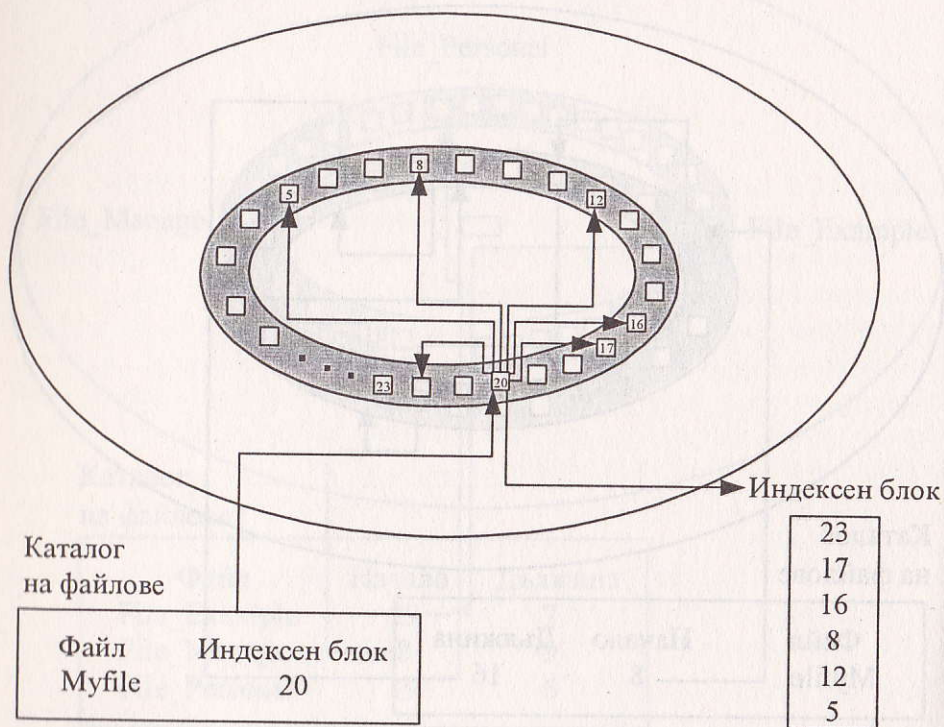
При списъчното разполагане на файлове всеки файл представлява свързан списък от блокове, при което блоковете могат да бъдат разхвърляни на произволни места в диска. Каталогът съдържа указател към първия (и последния) блок на файла (фиг. 3.13). Всеки блок съдържа указател към следващия блок. При четенето на файла блоковете се четат последователно блок след блок, следвайки указателите. Този метод не допуска външна фрагментация. Даден файл може да се разширява дотогава, докато има свободни блокове в диска. Никога не се налага компактиране на дисковото пространство. Основният недостатък на списъчното разполагане е, че не се поддържат файлове с директен достъп. Друг недостатък е допълнителното пространство за съхраняване на указателите. Ако даден указател се загуби или повреди, това може да доведе до свързване на празно пространство или друг файл.



Фиг. 3.13. Списъчно разполагане на файловете върху диска

При индексното разполагане на файла върху диска (фиг. 3.14) всички указатели се съхраняват в един блок, наречен индексен блок (блок с индексите). Всеки файл разполага със свой собствен блок на индексите, който съдържа фактически масив от адресите на заеманите блокове в диска. В таблицата на индексите i -тата позиция съдържа адреса на i -тия блок на файла. Методът с индексно разполагане на файловете не допуска външна фрагментация и поддържа файлове с директен достъп. Допълнителните разходи за поддържане на блока на индексите са по-големи от тези за поддържане на указателите при списъчното разполагане. Блокът на индексите обикновено заема един блок в диска. За поддържане на големи файлове няколко файлове с индексно разполагане могат да бъдат свързани. В този случай за индексните блокове се използва списъчно разполагане. В повечето случаи за големите файлове се оказват достатъчни две нива на индексите.

Сравнителният анализ на методите за разполагане на файловете върху диска по отношение на ефективността на използване на дисковото



Фиг. 3.14. Индексно разполагане на файловете върху диска

пространство показва следното: списъчното разполагане е ефективно при файлове с последователен достъп, но не може да се използва при файлове с директен достъп. Някои системи използват непрекъснато разполагане, за да поддържат файловете с директен достъп и списъчно разполагане за файлове с последователен достъп. Добра алтернатива е да се използва непрекъснато разполагане за малки файлове и автоматично да се превключва към индексно разполагане, ако файлът стане голям.

За ефективното използване на дисковото пространство е необходимо да се поддържат механизми за управление на свободното дисково пространство. Поради ограничения капацитет на диска и непрекъснатото създаване на нови файлове и разширяване на съществуващите файлове освободеното пространство при изтриване на файлове трябва да бъде използвано отново. Файловата система поддържа списък на всички свободни блокове на диска, наречен списък на свободното пространство. Популярна практика е списъкът на свободното пространство да се организира като списък на дупките или битова карта. Дупката се дефинира като последователност от свободни блокове с произволна дължина. Всяка позиция в списъка на дупките (фиг. 3.15) съдържа адреса на първия сектор на дупката и нейната дължина. Очевидно дължината на списъка на дупките и системните разходи за неговото поддържане зависи от разпределението на свободното пространство в диска, което се променя динамично във времето. Динамичното удължаване и скъсяване на този списък трудно се управлява от файловата система. Предимството на този подход е, че намирането на дупка със зададен размер е сравнително проста задача.

ДУПКА НОМЕР	НАЧАЛЕН АДРЕС		ДЪЛЖИНА (БРОЙ СЕКТОРИ)
	ПЪТЕЧКА	СЕКТОР	
1	2	4	3
2	4	3	6
3	5	1	8
4	6	8	3
5	8	5	9

Фиг. 3.15. Списък на дупките

Предимството на алтернативния подход с поддържане на битова карта (фиг. 3.16) е постоянния размер на битовата карта и простотата на нейната поддръжка. Промените в статуса на даден блок се отразяват чрез съответна промяна на състоянието на бита използван/свободен. Откриването на дупка със зададен размер е трудна задача.

НОМЕР	НОМЕР НА СЕКТОР									
ПЪТЕЧКА	0	1	2	3	4	5	6	7	8	9
0	1	0	0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1	1	0	0
2	0	1	1	0	0	1	1	1	1	1
3	1	0	0	0	0	1	1	1	0	0

Фиг. 3.16. Битова карта

Дисковото пространство може да се разглежда като колекция от използвани и свободни сегменти, като всеки сегмент съдържа множество от последователни блокове. Сегмент, който не се използва, се нарича дупка. Предназначението на *динамичното преразпределяне на дисковото пространство* е да осигури механизъм за обслужване на заявки за намиране на свободно пространство с размер n от списъка на дупките. За да се открие дупка с оптимален размер, е необходимо да се извърши търсене в списъка на дупките. Възможни са няколко стратегии:

- *Първо попадение (First-fit)* – избира се първата дупка, достатъчно голяма да удовлетвори изискванията на заявката. Търсенето в списъка на дупките може да започне от началото на списъка или от мястото, където е привършило предходното търсене.

- *Най-добро попадение (Best-fit)* – претърсва се целият списък до откриване на най-малката дупка, която удовлетворява заявката. Ако списъкът се поддържа сортиран по големината на дупките, не е необходимо да се търси в целия списък. При тази стратегия новосъздадените дупки са най-малки, поради което възниква външна фрагментация.

- *Най-лошо попадение (Worst-fit)* – при нея за обслужване на заявката се избира най-голямата дупка, при което новосъздадените дупки са най-големи. Ако списъкът не е сортиран, се налага цялостното му претърсване.

Стратегията с първо попадение е най-бързата от трите разгледани стратегии. По отношение на използването на дисковото пространство

стратегии с първо и най-добро попадение са по-ефективни от стратегията с най-лошо попадение. Недостатък и на трите стратегии е, че предизвикват външна фрагментация (празното пространство съдържа множество разхвърляни малки дупки в диска). В тези случаи дискът съдържа голямо свободно пространство, което е фрагментирано на множество и разпръснати малки дупки. За да се преодолее негативният ефект на външната фрагментация, е необходимо да се изпълни програма за компактиране, реализираща следното:

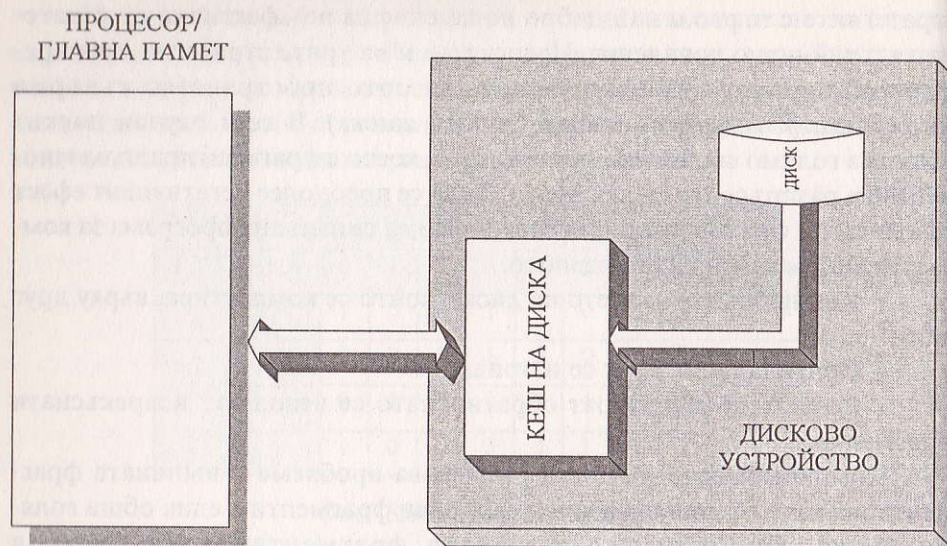
- Копира съдържанието на диска, който се компактира, върху друг диск;
- Оригиналният диск се изтрива;
- Файловете се копират обратно, като се използват непрекъснати фрагменти.

Компактирането на диска разрешава проблема с външната фрагментация като обединява всички свободни фрагменти в една обща голяма дупка. Проблемът с външната фрагментация на диска и необходимостта от компактиране на диска е най-актуален при метода с непрекъснато разполагане.

Бързодействието на магнитните дискове е значително по-малко (обикновено с няколко порядъка) от бързодействието на главната памет. Разликата се увеличава с напредъка на полупроводниковите технологии тъй като времето за достъп при диска и скоростта на трансфер се ограничава от механични фактори. Често процесорът не може да продължи изпълнението на текущата програма, докато се реализира трансфер между диска и главната памет, дори когато този режим се осъществява от контролер за директен достъп до паметта без участието на процесора.

За намаляване на негативните ефекти от голямото закъснение на дисковата памет се използва концепцията за използване на *кеш на диска* в интерфейса главна памет/диск (фиг. 3.17).

Кешът може да бъде включен в дисковото устройство или може да бъде организиран извън него в компютърната система. Капацитетът на дисковия кеш е много голям, имплементира се с RAM и съдържа блокове информация, които има вероятност да бъдат използвани в близкото бъдеще. Контролерът на кеша на диска активира информационния трансфер. Ефективността на дисковия кеш се повишава, като се използва принципът на локалността, който редуцира ефективното време за В/И трансфер. Някои операционни системи като UNIX използват софтуерни техники за кеширане на диска при поддържането на В/И буфер в главната памет. Единицата за трансфер между диска и кеша може да бъде сектор,



Фиг. 3.17. Имплементиране на кеш на диска в рамките на дисковото устройство

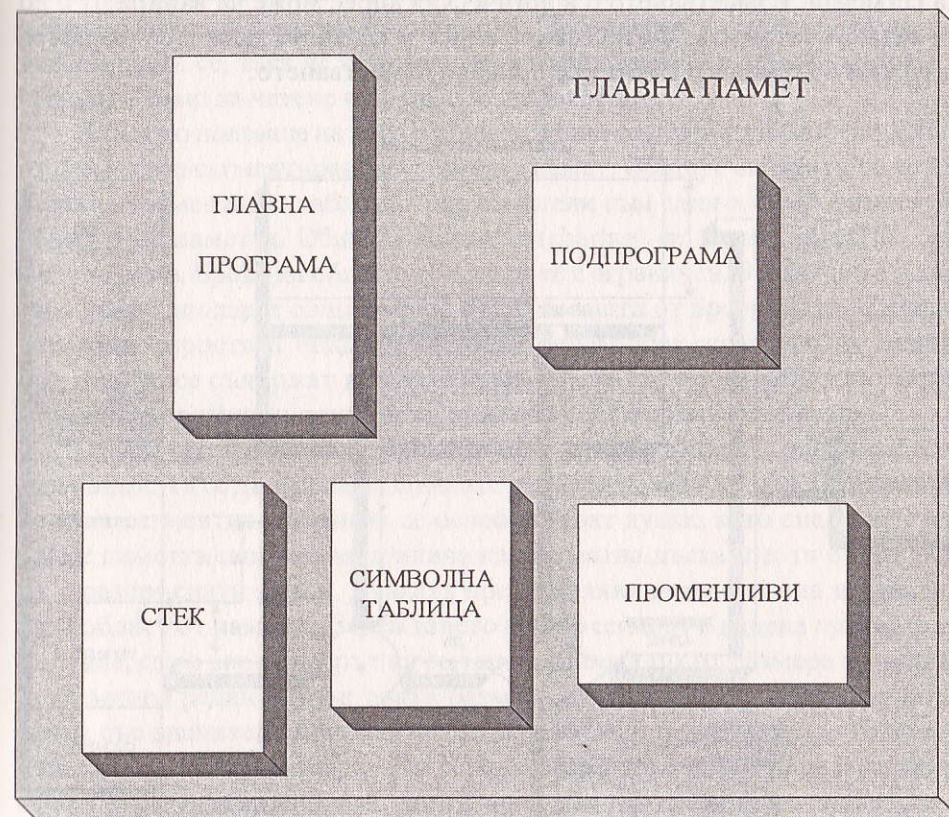
няколко сектора, пътечка или няколко пътечки. Кешът на диска естествено има вложени механизми за откриване и коригиране на грешки.

3.2. УПРАВЛЕНИЕ НА ПАМЕТТА В КОМПЮТЪРНИТЕ СИСТЕМИ

3.2.1. Сегментиране на паметта

Концептуален модел. Сегментирането е схема за управление на паметта, която е видима за програмиста и му осигурява удобство при организирането на програмата и данните. Сегментирането дава възможност на програмиста да разглежда паметта като колекция от множество адресни пространства или сегменти. Сегментите са дялове от паметта с различен размер и съдържат семантично дефинирана част от програмата. Следователно логическото адресно пространство може да се разглежда като колекция от сегменти (фиг. 3.18). Всеки сегмент има име и размер. Елементите в рамките на сегмента се идентифицират чрез тяхното изместване спрямо началото на сегмента. Всеки адрес задава името на сегмента и изместването в него. Практически в процеса на компилиране на

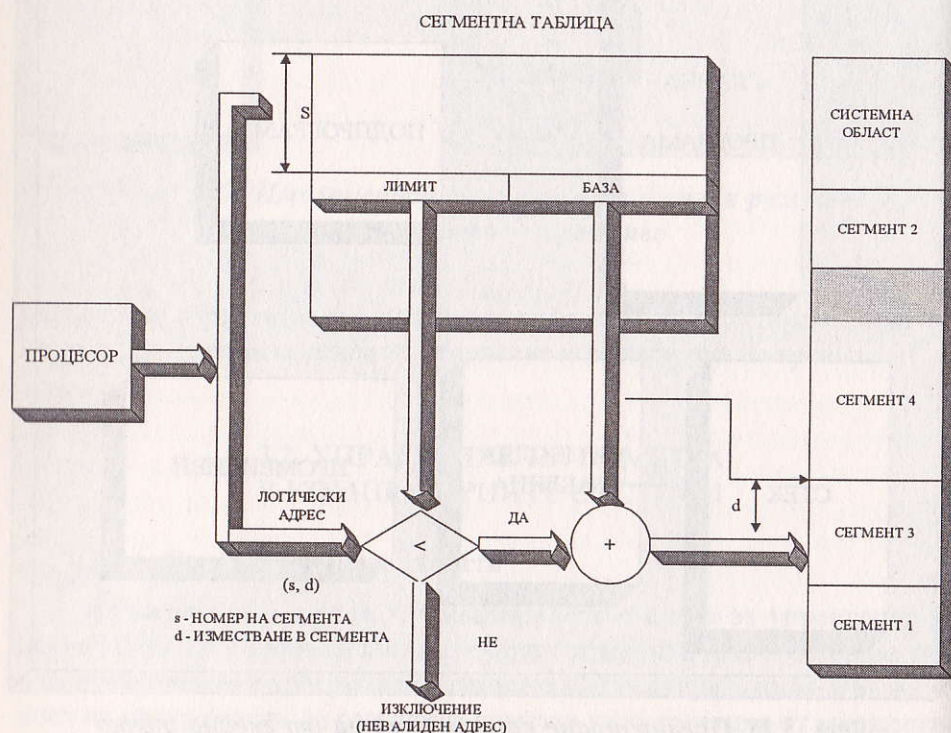
програмата компилаторът автоматично конструира сегментите в зависимост от спецификата на обработваната програма. За простота сегментите се номерират и обръщението към тях се осъществява чрез техните номера. Така например компилаторът създава отделни сегменти за глобалните променливи, стека за извикване на процедури, кода на всяка процедура и функция и за локалните променливи на всяка процедура и функция.



Фиг. 3.18. Организиране на програмата от гледна точка на потребителя

Очевидно при сегментирането потребителят има възможност да се обръща към обектите на програмата посредством двумерен адрес, въпреки че адресното пространство на компютърната система фактически представлява едномерна (линейна) последователност от думи. Операци-

онната система трябва да трансформира двумерните адреси, зададени от потребителя, в едномерни физически адреси. Механизмът за преобразуване на адресите е представен на фиг. 3.19. При сегментирането всеки логически адрес съдържа номер на сегмента s и изместването d в рамките на дефинирания сегмент. Номерът на сегмента се използва като индекс в таблицата на сегментите. За всеки сегмент в сегментната таблица се съдържат базата (началният адрес) на сегмента и лимитът (размерът) на сегмента. Изместването d в логическия адрес може да варира от 0 до лимита на сегмента. Физическият адрес в главната памет се формира като към базата на сегмента се прибави изместването.



Фиг. 3.19. Преобразуване на логическия адрес във физически адрес при сегментирана памет

За да се намалят допълнителните системни разходи за преобразуването на адресите, препоръчително е таблицата на сегментите да се имплементира с високоскоростни регистри, към които обръщението се

извършва много бързо. За всеки сегмент в таблицата на сегментите се съдържа съответно неговата база, размер и поле за достъп, дефиниращо правата за достъп до сегмента, например право на изпълнение за програмните сегменти и право на четене/запис за сегментите с данни. Използването на асоциативни регистри (кеш) ускорява процеса на преобразуване на адреса.

Сегментирането улеснява механизмите за защита. Сегментите могат да съдържат само инструкции или само данни поради семантичната дефинираност на тяхното съдържание. Програмните сегменти са немодифициращи се, така че тези сегменти могат да бъдат дефинирани като сегменти само за четене или само за изпълнение.

Общото ползване на кода и данните също се поддържа особено добре от схемата за сегментиране на главната памет. Общите сегменти се представят в сегментната таблица чрез указатели към едни и същи физически области от паметта. Общото ползване (sharing) се осъществява на сегментно ниво. Броят на общите сегменти не е ограничен. Възможно е също така да се използват общо и само отделни части от програмата. Сегментирането опростява поддържането на динамични структури от данни, тъй като те се съдържат в отделен сегмент, който операционната система може да разшири или стесни в зависимост от текущите нужди.

Недостатък на сегментирането е, че при него възниква *външна фрагментация*. Тя се дължи на различните размери на сегментите. При връщането на сегментите на диска се освобождават дупки, като след известно време паметта започва да прилича на шахматна дъска, т.е. тя е изпълнена с разпръснати дупки. Дупката представлява неизползвана непрекъсната област от паметта. Зареждането на нов сегмент в дадена дупка може да стане, само ако размерът на сегмента е по-малък от размера на дупката (в много редки случаи двата размера ще съвпадат). Възниква нова дупка със значително по-малък размер, която трудно може да бъде използвана за разполагане на нов сегмент. При това паметта се изпълва с малки и разпръснати дупки, които не могат да осигурят разполагането на цял сегмент в някоя от тях, тъй като той изисква непрекъсната област. Получава се парадокс: в главната памет има достатъчно свободно място, за да поеме новия сегмент, но това практически не може да стане поради фрагментирането на свободната памет. При сегментирането се използват няколко алгоритъма за използване на дупките: първо попадение, най-добро попадение и най-лошо попадение. При алгоритъма с първо попадение се прави търсене в таблицата на дупките от началото ѝ докато се открие първата дупка с достатъчно голям размер за разполагането на

сегмента. Алгоритмът може да се модифицира по такъв начин, че да се игнорират дупки, при използването на които за разполагане на новия сегмент новосъздадената дупка ще бъде много малка и няма да може да бъде използвана отново. При алгоритъма с най-добро попадение се търси дупка с размер, най-близък до търсения, като новосъздадената дупка е минимална. Използването на този алгоритъм предизвиква генерирането на множество малки неизползваеми дупки, разпръснати из цялата памет и води до изключително висока интензивност на външната фрагментация. В този случай може да се приложи компактиране, за да се конструира свободна област, достатъчно голяма, за да поеме разполагането на целия сегмент. Компактирането на паметта не е проблем, тъй като фактически схемата за сегментиране поддържа алгоритъм за динамично преразпределяне на паметта, но от друга страна изисква време. При алгоритъма с най-лошо попадение за разполагането на новия сегмент се използва най-голямата налична дупка, при което се разчита, че новосъздадената дупка ще бъде достатъчно голяма да поеме разполагането на сегмент. Степента на външна фрагментация зависи от рамера на сегмента. По правило, колкото е по-голям размерът на сегмента, толкова е по-голяма вероятността за външна фрагментация.

Сегментиране на паметта при системи, базирани на Intel® Pentium™ и Xeon™. Средствата за управление на паметта при архитектурата IA-32 поддържат сегментирането като механизъм за изолиране на индивидуалните модули код, данни и стек на различните програми (или задачи) по такъв начин, че те да бъдат изпълнявани от процесора, без да си пречат. В защитен режим не е възможно да се забрани сегментирането.

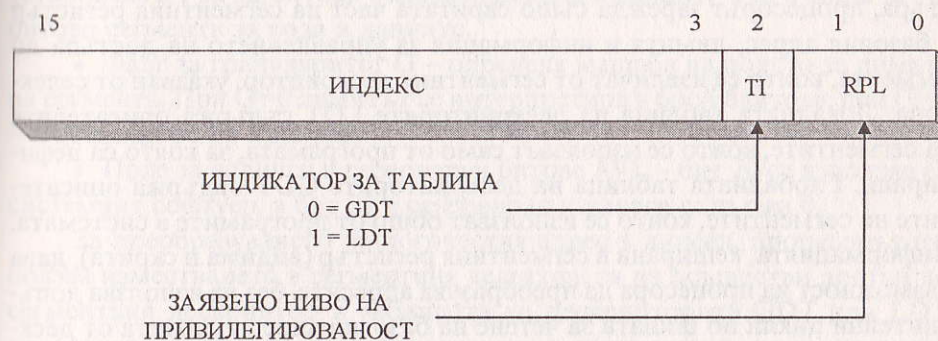
Сегментирането осигурява възможност за разделяне на адресното пространство (наречено линейно адресно пространство) на по-малки адресни пространства, наречени сегменти. Ако в даден момент повече от една програма (или задача) се изпълняват от процесора, всяка програма разполага със собствено множество сегменти. Процесорът налага граници между тези сегменти по такъв начин, че изпълнението на една програма да не попречи на останалите, като записва в техните сегменти. Всички сегменти в системата се съдържат в линейното адресно пространство на процесора. В защитен режим архитектурата IA-32 осигурява физическо адресно пространство от 4 Gbytes (2^{32} байта), като адресите варират от 0 до FFFFFFFFH. Физическото адресно пространство се дефинира като обхвата от адреси, които процесорът може да генерира по адресната шина. То се разпределя между RAM, ROM и В/И адреси. Линейното адресно пространство съдържа всички сегменти и системни таблици.

Сегментният селектор представлява 16-битов идентификатор на сегмента. Той не указва директно сегмента, а вместо това указва дескриптора (описателя) на сегмента, който го дефинира. Сегментният селектор (фиг. 3.20) съдържа:

- Индекс (битове от 3 до 15), който избира един от 8192 дескриптора в глобалната дескрипторна таблица GDT или в локалната дескрипторна таблица LDT.
- Флаг-идентификатор на таблицата TI – при 0 избира GDT, при 1 – LDT.
- Заявено ниво на привилегированост RPL – определя нивото на привилегированост на селектора. То може да бъде в обхвата от 0 до 3, като ниво 0 е най-високо привилегированото.

Сегментните селектори са видими за приложния програмист като част от указател, но те се присвояват от свързващия редактор, а не от приложните програми.

За да се намали времето за преобразуване на адреса и сложността на кода, процесорът съдържа регистри, които могат да съхраняват до 6 сегментни селектора. За да може да се изпълни една програма, поне регистрите на програмния сегмент (CS), сегмента за данни (DS) и сегмента на стека (SS) трябва да бъдат заредени с валидни сегментни селектори. Процесорът също осигурява три регистъра за допълнителни сегменти за данни (ES, FS, GS), които могат да бъдат използвани от текущата програма.



Фиг. 3.20. Формат на селектора на сегмента

За да може програмата да осъществи достъп до сегмент, съответният сегментен селектор трябва да се зареди в един от сегментните регистри.

Въпреки че в системата могат да бъдат дефинирани много сегменти, в даден момент могат да бъдат използвани само 6 сегмента. Използването на другите сегменти може да се осъществи, като техните селектори се заредят в сегментните регистри по време на изпълнението на програмата.

Всеки сегментен регистър (фиг. 3.21) има „видима“ част и „скрита“ част. Скрытата част се нарича още „кеш на дескрипторите“ или „регистър в сянка“.

ВИДИМА ЧАСТ		СКРИТА ЧАСТ	
СЕЛЕКТОР НА СЕГМЕНТА	БАЗОВ АДРЕС, ЛИМИТ, ПРАВА ЗА ДОСТЪП		
		CS	
		SS	
		DS	
		ES	
		FS	
		GS	

Фиг. 3.21. Сегментни регистри

Когато сегментният селектор се зареди във видимата част на регистъра, процесорът зарежда също скритата част на сегментния регистър с базовия адрес, лимита и информация за управлението на достъпа до сегмента, които се извличат от сегментния дескриптор, указван от селектора. Локалната таблица на дескрипторите LDT съдържа описателите на сегментите, които се използват само от програмата, за която са дефинирани. Глобалната таблица на дескрипторите GDT съдържа описателите на сегментите, които се използват общо от програмите в системата. Информацията, кеширана в сегментния регистър (видима и скрита), дава възможност на процесора да преобразува адресите, без да използва допълнителни цикли по шината за четене на базовия адрес и лимита от дескриптора.

Дескрипторът на сегмента представлява структура данни в глобалната или локалната таблица на дескрипторите, която съдържа информация за размера и местоположението на сегмента, неговия статус и права за достъп. Дескрипторите на сегментите се създават от компила-

торите, свързващите редактори, зареждащите програми или от операционната система, но не от приложните програми. На фиг. 3.22 е представен общият формат на дескриптор за всички типове сегментни регистри. Форматът на дескриптора съдържа следните полета:

- Поле на *лимита*, което определя размера на сегмента. Процесорът интерпретира съдържанието на това поле по два начина в зависимост от състоянието на флага за грануляритет G – при $G=0$ размерът на сегмента варира от 1 byte до 1 Mbyte при инкремент от 1 byte, докато при $G=1$ размерът на сегмента варира от 4 Kbytes до 4 Gbytes при инкремент от 4 Kbytes.

- Поле на базовия адрес – задава адреса на нулевия байт на сегмента в 4 Gbytes линейно адресно пространство. Процесорът използва трите полета на базовия адрес, за да формира 32-битовия базов адрес.

- Поле за типа на сегмента – идентифицира типа на сегмента, разрешените видове за достъп и посоката на нарастване на сегмента (към големите или малките адреси). Кодирането на това поле е различно за дескрипторите на сегментите на кода, данните и системните дескриптори.

- Флаг S – показва дали сегментът е системен, сегмент за код или данни.

- Поле за ниво на привилегированост DPL – определя нивото на привилегированост на сегмента (от 0 до 3).

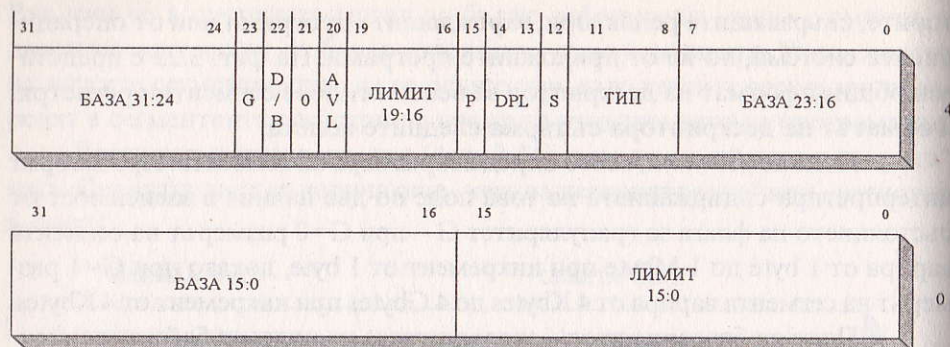
- Флаг P за наличност – показва дали сегментът присъства в паметта ($P=1$) или не присъства в паметта ($P=0$).

- Поле за параметри по премълчаване DB – този флаг винаги се установява в 1 за 32-битови сегменти за кода и данните и се нулира за 16-битови сегменти за кода и данните.

- Флаг за грануляритет G – определя мащаба на полето за лимита на сегмента. При $G=0$ лимитът се интерпретира в брой байтове, при $G=1$ – в единици от по 4 Kbytes.

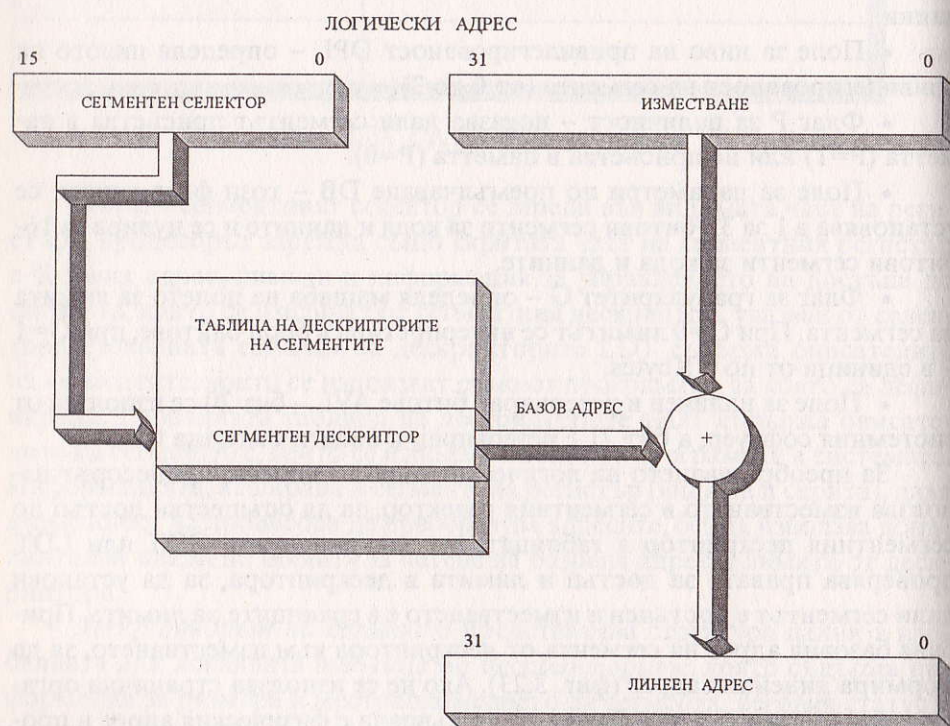
- Поле за наличен и резервиран битове AVL – бит 20 се използва от системния софтуер, а бит 21 е резервиран и винаги съдържа 0.

За преобразуването на логическия адрес в линейен, процесорът използва изместването в сегментния селектор, за да осъществи достъп до сегментния дескриптор в таблицата на дескрипторите GDT или LDT, проверява правата за достъп и лимита в дескриптора, за да установи дали сегментът е достъпен и изместването е в границите на лимита. Прибавя базовия адрес на сегмента от дескриптора към изместването, за да формира линейния адрес (фиг. 3.23). Ако не се използва странична организация на паметта, линейният адрес съвпада с физическия адрес и процесорът директно го подава по адресната си шина.



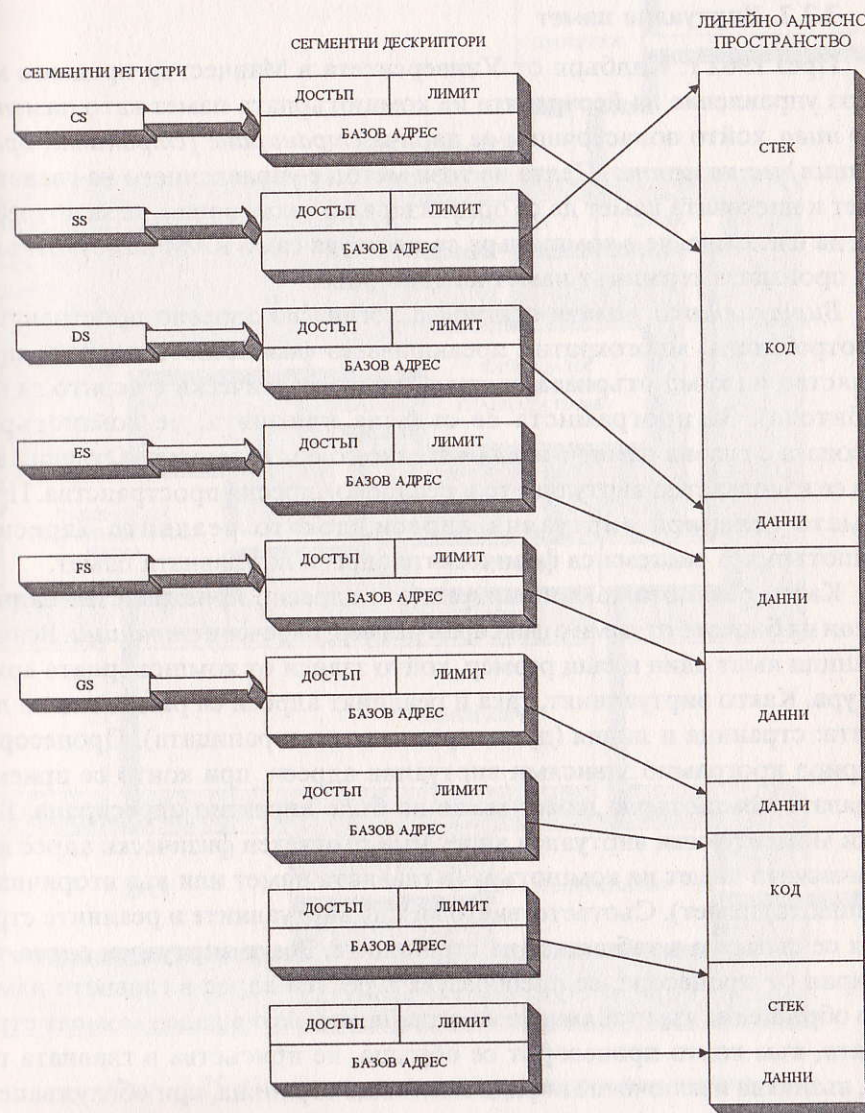
AVL - НАЛИЧЕН ЗА ИЗПОЛЗВАНЕ ОТ СИСТЕМНИЯ СОФТУЕР
D/B - 0=16-БИТОВ СЕГМЕНТ; 1=32-БИТОВ СЕГМЕНТ
DPL - НИВО НА ПРИВИЛЕГИРОВАНОСТ НА СЕГМЕНТА
G - ГРАНУЛЯРИТЕТ
P - НАЛИЧНОСТ НА СЕГМЕНТА
S - ТИП НА ДЕСКРИПТОРА (0 = СИСТЕМЕН; 1 = КОД ИЛИ ДАННИ)

Фиг. 3.22. Формат на дескриптора на сегмента



Фиг. 3.23. Преобразуване на логическия адрес в линеен

При мултисегментния модел всяка програма (или задача) разполага със собствени сегменти и собствена таблица на сегментните регистри (фиг. 3.24). Сегментите могат да се използват само от програмата или да бъдат използвани общо между програмите. Достъпът до сегментите и изпълнителните среди от изпълняваните програми в системата се управляват от хардуера.



Фиг. 3.24. Мултисегментен модел

Проверките за достъп се използват не само за защита при обръщане към адрес извън лимита на сегмента, а също така и за предотвратяване на забранени операции в определени сегменти. Например сегментите с кода се дефинират с права само за четене и хардуерът предотвратява запис в този тип сегменти.

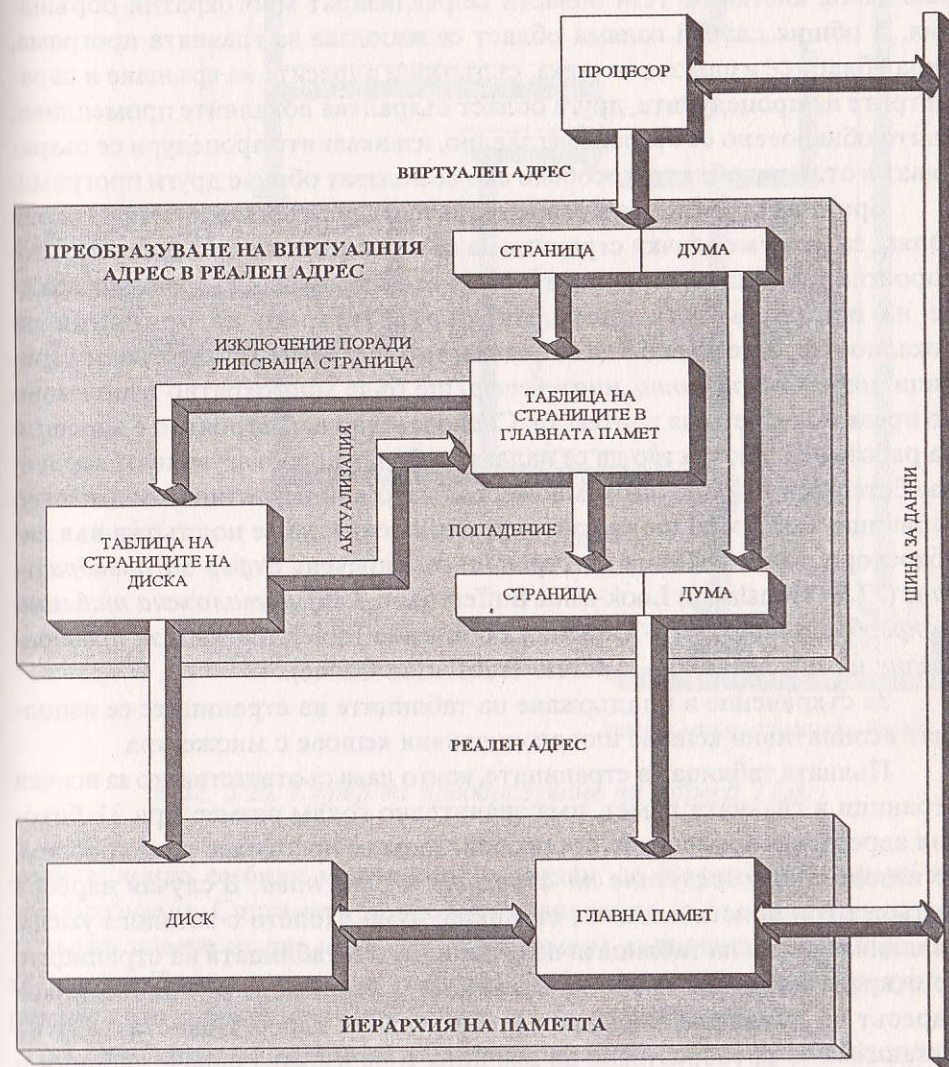
3.2.2. Виртуална памет

През 1962 г. Килбърн от Университета в Манчестър предлага метод за управление на йерархията на компютърната памет като *памет с едно ниво*, който понастоящем се нарича *страниране (странична организация) на паметта*. Целта на този метод е управлението на главната памет и дисковата памет да се организира по такъв начин, че за потребителя да изглежда, че в компютъра съществува само RAM памет, откъдето и произлиза терминът памет на едно ниво.

Виртуалната памет осигурява логическо адресно пространство за потребителя, многократно превишаващо физическото адресно пространство на компютърната система (което фактически е скрито за потребителя). За програмиста се създава илюзията, че компютърът разполага с главна памет с изключително голям капацитет. Отделни адреси се използват за виртуалното и реалното адресни пространства. Програмата генерира виртуални адреси, докато реалните адреси в компютърната система са физическите адреси на главната памет.

Както реалното така и виртуалното адресно пространство са разделени на блокове от думи с фиксиран размер, наречени *страници*. Всички страници имат един и същ размер, който зависи от компютърната архитектура. Както виртуалният, така и реалният адреси са разделени на две полета: страница и линия (дума в рамките на страницата). Процесорът генерира програмно зависими виртуални адреси, при които се приема, че цялата компютърна памет може да бъде директно адресирана. Във всеки момент всеки виртуален адрес има съответен физически адрес във физическата памет на компютъра (в главната памет или във вторичната (външната) памет). Съответствието между виртуалните и реалните страници се съдържа в таблицата на страниците. Всеки виртуален адрес, генериран от процесора, се преобразува в реален адрес в главната памет чрез обръщение към таблицата на страниците. Ако в даден момент страницата, към която процесорът се обръща, не присъства в главната памет, възниква изключение поради липсваща страница, при обслужването на което се активира подпрограма, която автоматически зарежда липс-

ващата страница в главната памет и съответно актуализира съдържанието на таблицата на страниците. На фиг. 3.25 е представен обобщен модел на компютърна система с виртуална памет. На преобразуване подлежи само полето на страницата в адреса, тъй като полетата на линията са еднакви във виртуалния и физическия адреси. Преобразуването на вир-



Фиг. 3.25. Обобщен модел на компютърна система с виртуална памет

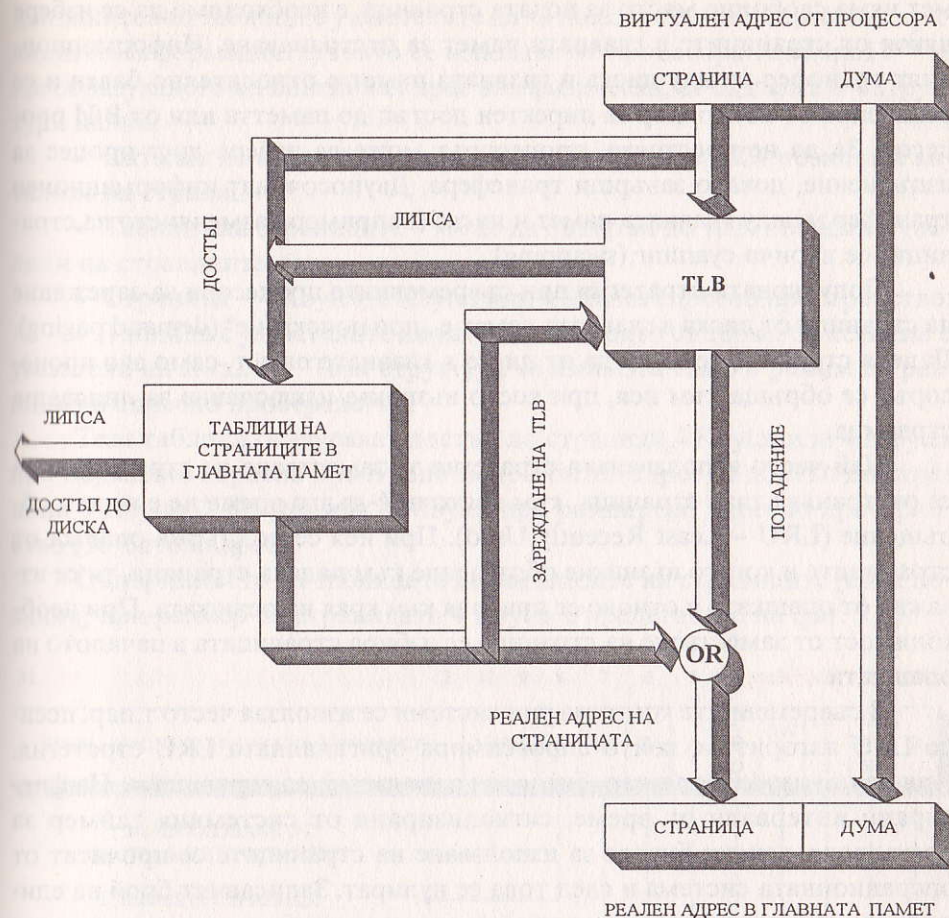
туалния адрес във физически се осъществява задължително при всяко обръщение към паметта.

Успехът на странирането на паметта в значителна степен зависи от принципа на локалността (по време и пространство) на изпълняваните програми. Обръщенията към паметта се групират в отделни области, като към някои клетки от тези области се реализират многократни обръщания. В общия случай голяма област се използва за главната програма, една област се използва за стека, съдържащ адресите на връщане и параметрите на процедурите, друга област съхранява локалните променливи, които обикновено се съхраняват заедно, извикваните процедури се съхраняват в отделни области, особено ако се ползват общо с други програми.

Броят на страниците в съвременните компютърни системи е твърде голям, за да може всички страници да се поддържат само в една високоскоростна таблица на страниците. Като се имат предвид характеристиките на изпълняваната програма, съответстващи на принципа на локалността, естествено е да се очаква, че определено множество от страници, наречено *работно множество*, ще бъде многократно използвано до превключването на контекста. Следователно най-вероятно е адресите на работното множество да се налага да бъдат преобразувани от хардуера. Естествено е работното множество като най-вероятното множество страници, към което ще се извърши обръщение, да се поддържа във високоскоростната таблица на страниците, наречена *буфер за преобразуване* (TLB - Translation Look-aside Buffer) (фиг. 3.26), *каталожна таблица за преобразуване* (DLT - Directory Look-aside Table), или *кеш за преобразуване на адреса* (ATC - Address Translation Cache).

За съхранение и поддържане на таблиците на страниците се използват асоциативни кешове или асоциативни кешове с множества.

Пълната таблица на страниците, която дава съответствието за всички страници в главната памет, има значително голям размер при 32-битовия адрес на съвременните процесори. За да се преодолее този проблем, се използва *преобразуване на адресите на две нива*. В случая адресът съдържа три полета: каталог, страница, дума. Полето с каталога указва началният адрес на таблицата на страниците. В таблицата на страниците се открива началния адрес на страницата, към който се присъединява адресът на думата за формирането на реалния адрес. Недостатъкът на механизма за преобразуване на две нива е, че изисква повече таблици и две обръщания към таблиците, за да се извлече адресът на страницата. Очевидното му предимство обаче е, че индивидуалните таблици имат много по-малък размер и заемат по-малко памет. Множеството от таб-



Фиг. 3.26. Буфер за преобразуване на адреса TLB

лиците, които се били използвани последни, се съхранява в буфера за преобразуване. Системата за преобразуване може да бъде разширена да съдържа повече от две нива, като се формира дървовидна структура от указатели. Такива структури улесняват работата с таблиците и редуцират пълната таблица със страниците, които не се използват.

При липса на страница, към която процесорът се обръща, възниква изключение, като операционната система е отговорна за зареждането на страницата от диска. Местоположението на търсената страница в диска се открива чрез таблицата на страниците в диска. Когато в главната па-

мет няма свободно място за новата страница, е необходимо да се избере някоя от страниците в главната памет за отстраняване. Информационният трансфер между диска и главната памет е относително бавен и се реализира от контролер за директен достъп до паметта или от В/И процесор. За да не престоива, процесорът може да избере друг процес за изпълнение, докато завърши трансфера. Двупосочният информационен трансфер между главната памет и диска (например разменянето на страници) се нарича суапинг (swapping).

Популярната стратегия при съвременните процесори за зареждане на страници от диска в главната памет е „при поискване“ (demand paging). Дадена страница се зарежда от диска в главната памет, само ако процесорът се обръща към нея, при което възниква изключение за липсваща страница.

Най-често използваната стратегия за заместване на страници е да се отстранява тази страница, към която най-дълго време не е имало обръщение (LRU – Least Recently Used). При нея се поддържа опашка от страниците и когато възникне обръщение към дадена страница, тя се изважда от опашката и отново се прибавя към края на опашката. При необходимост от заместване на страница се избира страницата в началото на опашката.

В съвременните компютърни системи се използва често т.нар. псевдо-LRU алгоритъм, който апроксимира оригиналната LRU-стратегия. При нея се използват т.нар. битове за използване на страницата. На фиксирани интервали от време, сигнализирани от системния таймер за прекъсване, всички битове за използване на страниците се прочитат от операционната система и след това се нулират. Записаният брой на единичните състояния на бита за използване дава приблизителна представа за честотата на използването на дадена страница в рамките на определен интервал от време. Колкото е по-малък интервалът, толкова приблизителната оценка се доближава до оценката на LRU-алгоритъма.

Управление на паметта при IA-32 архитектурата. Средствата за управление на паметта са разделени на две части: сегментиране и страниране. В защитен режим архитектурата поддържа странична организация на паметта, като се използва механизъм за зареждане на страниците в главната памет при поискване. Размерът на поддържаните страници може да бъде 4 Kbytes, 2 Kbytes и 4 Mbytes. Архитектурата поддържа и метод за разширяване на физическия адрес до 36 бита, като става възможно адресирането на физическа памет над FFFFFFFFH. Това разширение е възможно само при странична организация. При него се разчита на

допълнителни таблици с указателите на каталозите на таблиците на страниците. Информацията, която се използва от процесора Pentium IV™ за преобразуването на линейния адрес във физически, се съдържа в 4 структури данни:

- Каталог на страниците – може да съдържа до 1024 позиции с каталози на страниците;
- Таблица на страниците – може да съдържа до 1024 позиции с таблици на страниците;
- Страница – 4 Kbytes, 2 Kbytes или 4 Mbytes адресно пространство;
- Таблица с указателите на каталозите, която съдържа 64 позиции с указатели на каталози. Тази структура се използва само в режим на разширено адресно пространство.

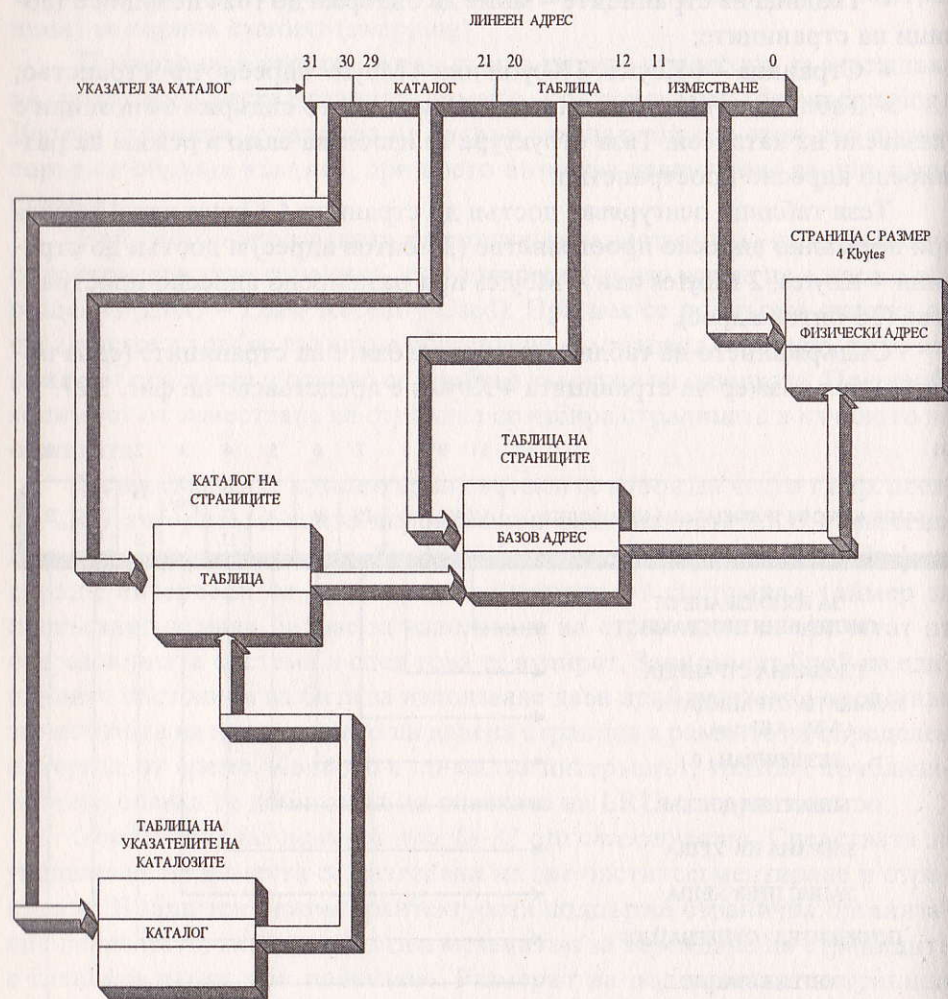
Тези таблици осигуряват достъп до страници 4 Kbytes или 4 Mbytes при нормално адресно пространство (32-битов адрес) и достъп до страници 4 Kbytes, 2 Mbytes или 4 Mbytes при разширено адресно пространство (36-битов адрес).

Съдържанието на таблицата на каталозите на страниците (една позиция) при размер на страницата 4 Kbytes е представено на фиг. 3.27.



Фиг. 3.27. Съдържание на таблицата на каталозите на страниците, (една позиция) при размер на страницата 4 Kbytes

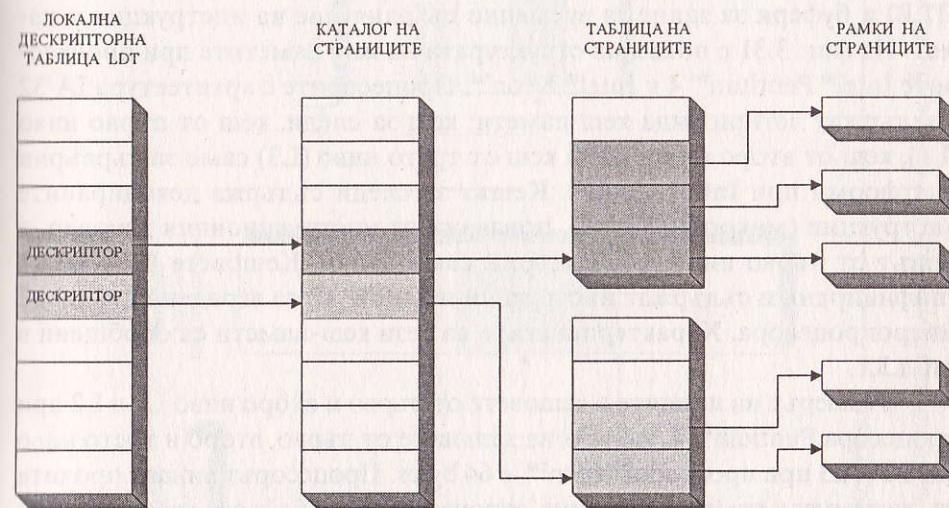
На фиг. 3.28 е показана йерархията на указателите на каталога на страниците, каталогът на страниците и таблицата на страниците при преобразуването на линейния адрес във физически адрес при страница с размер 4 Kbytes. Този метод за страниране може да се прилага за адресиране до 2^{20} страници, които обхващат адресно пространство от 2^{32} байта (4 Gbytes). Линейният адрес е разделен на 4 секции: таблица на указате-



Фиг. 3.28. Преобразуване на линейния адрес във физически адрес при размер на страницата 4 Kbytes в 4 Gbytes физическо адресно пространство

лите на каталозите, таблица на каталозите, таблица на страниците и изместване в рамките на страницата. Битове 30 и 31 съдържат изместване за една от четирите позиции в таблицата на указателите на каталозите. Избраната позиция съдържа базовия физически адрес на каталога на страниците. Битове от 21 до 29 съдържат изместване към позиция в избрания каталог. Избраната позиция съдържа базовия физически адрес на таблицата на страниците. Битове от 12 до 20 съдържат изместване към позиция в избраната таблица на страниците. Избраната позиция съдържа базовия физически адрес на страница във физическата памет. Битовите от 0 до 11 съдържат изместването в рамките на страницата.

Комбинираното използване на сегментиране и странична организация осигурява поддръжка на различни подходи за управлението на паметта. Един от ефективните подходи за комбиниране, който опростява софтуера за управление на паметта, е всеки сегмент да разполага със собствена таблица на страниците (фиг. 3.29).



Фиг. 3.29. Управление на паметта с присвояване на таблица на страниците за всеки сегмент

Виртуализацията на линейното адресно пространство се осъществява от механизма на процесора за страниране. Сегментът се разделя на страници, които се съхраняват в главната памет или на диска. Операционната система поддържа каталог на страниците и множество таблици на страниците, за да проследява движението на страниците. Сегментите

могат да бъдат по-малки от страницата. Ако такъв сегмент бъде поставен в страница, която не е обща и за друг сегмент, празното място ще остане неизползвано. Архитектурата IA-32 не налага съответствие между границите на страниците и сегментите. Дадена страница може да съдържа края на един сегмент и началото на друг сегмент. Аналогично даден сегмент може да съдържа края на една страница и началото на друга страница.

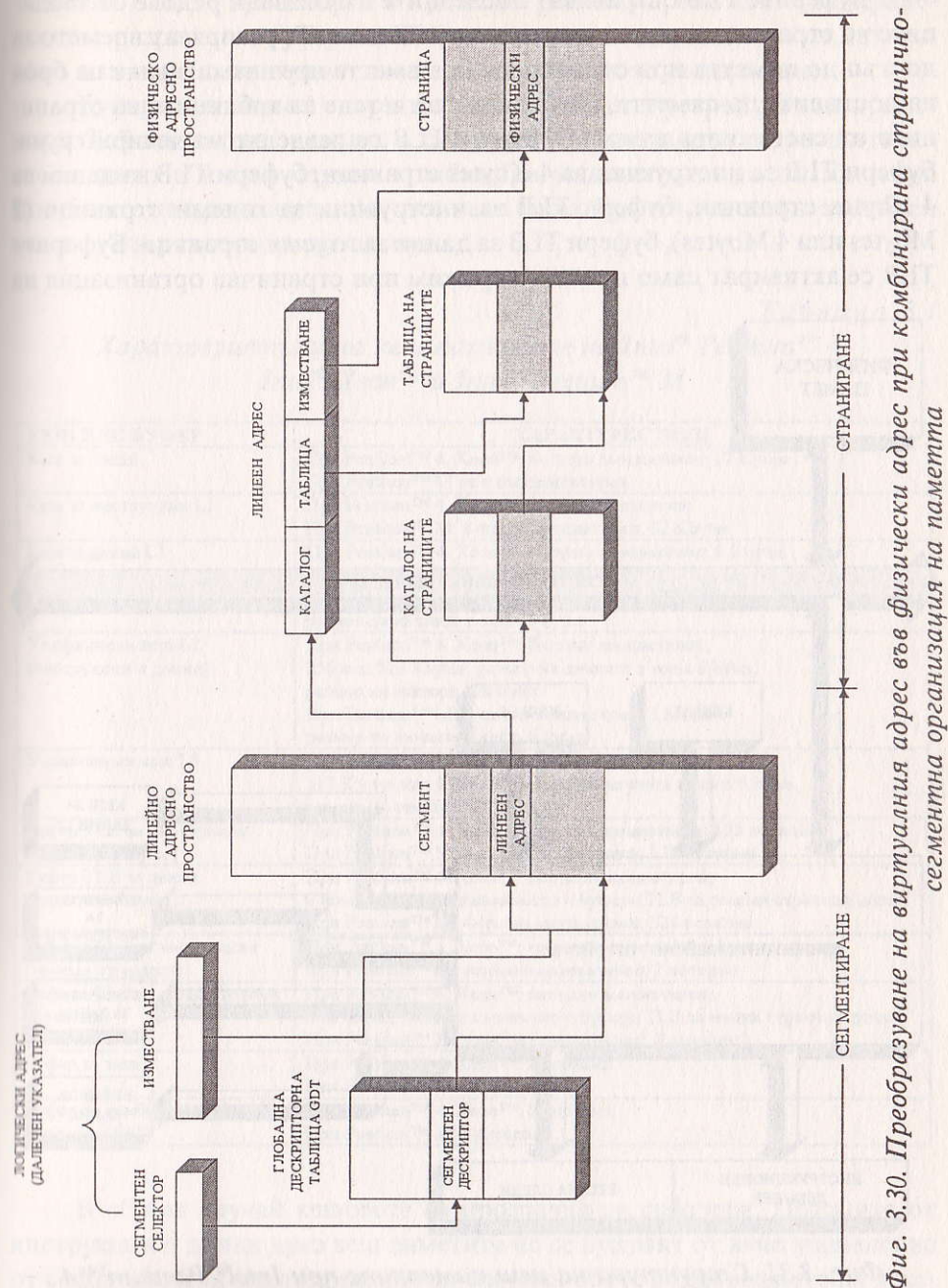
Преобразуването на адреса при комбиниране на сегментна със странична организация включва два етапа: преобразуване на логическия адрес в линеен и преобразуване на линеиния адрес във физически адрес в главната памет (фиг. 3.30).

3.2.3. Управление на кеш паметта

Архитектурата на съвременните компютърни системи поддържа кешове (максимално до три нива), буфери за преобразуване на адреса (TLB) и буфери за запис за временно съхраняване на инструкции и памет. На фиг. 3.31 е показана структурата на кеш паметите при процесорите Intel® Pentium™ 4 и Intel® Xeon™. Процесорите с архитектура IA-32 поддържат четири вида кеш памети: кеш за следи, кеш от първо ниво (L1), кеш от второ ниво (L2) и кеш от трето ниво (L3) само за сървърни платформи при Intel® Xeon™. Кешът за следи съдържа декодираните инструкции (микрооперации), подавани от инструкционния декодер, а кешът от първо ниво (L1) съдържа само данни. Кешовете L2 и L3 са унифицирани и съдържат инструкции и данни. Те са вградени в чипа на микропроцесора. Характеристиките на тези кеш-памети са обобщени в табл.3.1.

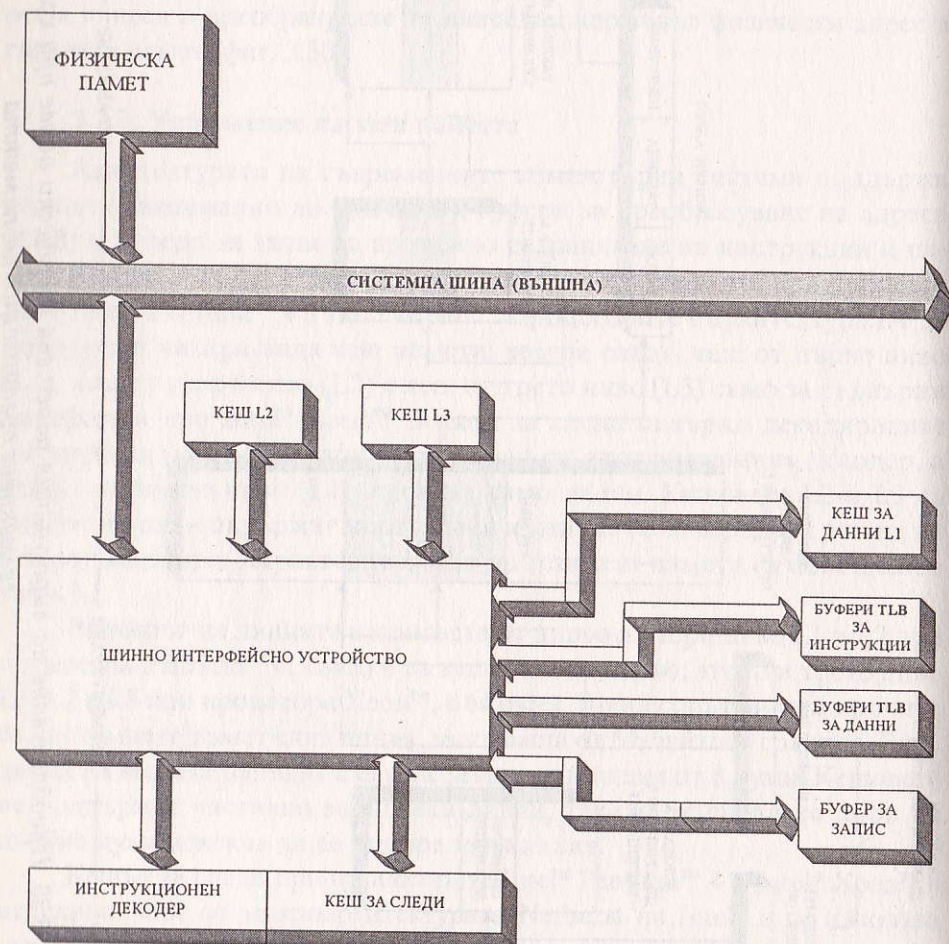
Размерът на линиите в кешовете от първо и второ ниво L1 и L2 при процесора Pentium™ 4, както и на кешовете от първо, второ и трето ниво L1, L2 и L3 при процесора Xeon™, е 64 bytes. Процесорът винаги прочита от системната памет една линия, започваща от 64-байтова граница. Една линия на кеша се запълва с експлозивна транзакция от 8 думи. Кешовете не поддържат частично запълнени линии, така че кеширането дори на двойна дума изисква да се кешира цяла линия.

Кешът за следи при процесорите Intel® Pentium™ 4 и Intel® Xeon™ е неделима част от микроархитектурата Netburst на Intel® и се използва във всички режими на изпълнение: защитен режим, режим на управление на системата SMM и режим с реални адреси. Аналогично кешовете от първо, второ и трето ниво L1, L2 и L3 са налични за всички режими.



Фиг. 3.30. Преобразуване на виртуалния адрес във физически адрес при комбинирана странично-сегментна организация на паметта

Буферите TLB съхраняват последните използвани редове от таблиците на страниците и каталозите на страниците. Те ускоряват времето за достъп до паметта при страниране на паметта чрез намаляване на броя на достъпите до паметта, необходими за четене на таблиците на страниците от системната памет. Буферите TLB се разделят на четири групи: буфери TLB за инструкции за 4-Kbytes страници, буфери TLB за данни за 4-Kbytes страници, буфери TLB за инструкции за големи страници (2 Mbytes или 4 Mbytes), буфери TLB за данни за големи страници. Буферите TLB се активират само в защитен режим при странична организация на



Фиг. 3.31. Структура на кеш паметите при Intel® Pentium™ 4 и Intel® Xeon™

паметта. Когато процесорът е в режим с реални адреси или страниране не е разрешено, буферите TLB поддържат съдържанието си, докато явно или неявно принудително се изчисти тяхното съдържание.

Буферът за запис е свързан с изпълнителните устройства на процесора. Той поддържа записите в системната памет и/или вътрешните кешове, като в някои случаи комбинира записите, за да оптимизира достъпите на процесора по системната шина. Буферът за запис е активен във всички режими на процесора.

Таблица 3.1
Характеристики на кеш паметите на Intel® Pentium™ 4, Intel® Xeon™ и Intel® Pentium™ M

КЕШ ИЛИ БУФЕР	ХАРАКТЕРИСТИКИ
Кеш за следи	При Pentium™ 4, Xeon™: 8-пътно асоциативен; 12 Кмops При Pentium™ M не е имплементиран
Кеш за инструкции L1	При Pentium™ 4, Xeon™ не е имплементиран При Pentium™ M: 8-пътно асоциативен; 32 Kbytes
Кеш за данни L1	При Pentium™ 4, Xeon™: 4-пътно асоциативен; 8 Kbytes ; размер на линията в кеша 6 bytes При Pentium™ M: 8-пътно асоциативен; 32 Kbytes; размер на линията в кеша 6 bytes
Унифициран кеш L2 (инструкции и данни)	При Pentium™ 4, Xeon™: 8-пътно асоциативен; 256 или 512 Kbytes; размер на линията в кеша 6 bytes, размер на сектора 128 bytes При Pentium™ M: 8-пътно асоциативен; 1 Мбайт; размер на линията в кеша 6 bytes
Унифициран кеш L3	При Intel® Xeon™: 8-пътно асоциативен; 512 Kbytes или 1 Мбайт; размер на линията в кеша 6 bytes, размер на сектора 128 bytes
Буфер TLB за инструкции (страници Kbytes)	При Pentium™ 4, Xeon™: 4-пътно асоциативен; 128 позиции При Pentium™ M: 4-пътно асоциативен; 128 позиции
Буфер TLB за данни (страници Kbytes)	При Pentium™ 4, Xeon™: напълно асоциативен; 6 позиции; ползван съвместно с буфери TLB за големи страници данни При Pentium™ M: 4-пътно асоциативен; 128 позиции
Буфери TLB за инструкции (големи страници)	При Pentium™ 4, Xeon™: големите страници са фрагментирани При Pentium™ M: напълно асоциативен; 2 позиции
Буфери TLB за данни (големи страници)	При Pentium™ 4, Xeon™: напълно асоциативен; 6 позиции; ползван съвместно с буфери TLB за малки страници данни При Pentium™ M: напълно асоциативен; 8 позиции
Буфер за запис	При Pentium™ 4, Xeon™: 2 позиции При Pentium™ M: 16 позиции
Буфер за комбиниране на записите (WC)	При Pentium™ 4, Xeon™: 6 позиции При Pentium™ M: 6 позиции

В общия случай кешовете са прозрачни за софтуера. Потоките от инструкции и данни през кеш паметите не се нуждаят от явно управление от софтуера. При мултипроцесорни системи обаче поддържането на кохерентността на кешовете може да изисква в някои случаи интервенцията на

софтуера. За тази цел процесорът осигурява инструкции за управление на кеша, които дават възможност съдържанието на кешовете да се изчисти и също така да се поддържа редът на достъпите до паметта.

Архитектурата IA-32 поддържа две групи механизми за управление на кеширането на данни и инструкции, както и за управление на реда на операциите за четене и запис между процесора, кешовете и паметта. Първата група механизми използва специализирани регистри и битове в управляващите регистри. Втората група механизми използва инструкции за управление на кеш паметта.

При тази архитектура за управление на кеширането на виртуални страници и области на физическата памет в кешовете на трите нива се използват няколко специализирани регистъра и различни битове в рамките на управляващите регистри и редовете в таблиците и каталозите на страниците. Флаговете CD (бит 30 – Cache Disable) и NW (бит 20 – Not Write Through) в управляващия регистър CR0 управляват кеширането на системната памет. Комбинация 00 на двата флага активира кеширането на цялата физическа памет на всички нива с обратен запис в главната памет при попадение. При CD=1 кеширането е деактивирано. Флаговете PCD (бит 4 – Page-level Cache Disable) и PWT (бит 3 – Page-level Writes Transparent) в управляващия регистър CR3 управляват цялостното кеширане и стратегията за запис в каталозите на страниците. Когато е нулиран, флагът PCD разрешава кеширането на каталога на страниците и го забранява, когато е установен в единица. Флагът PWT активира кеширане с обратен запис на каталога на страниците, когато е нулиран и запис през кеша в паметта, когато е установен в 1. Флагът PCD в реда на каталога и таблицата на страниците управлява съответно кеширането на индивидуалните таблици и страници. Флагът PWT в реда на каталога и таблицата на страниците управлява стратегията съответно за запис на индивидуалните таблици и страници. Глобалният флаг G в реда на каталога и таблицата на страниците управлява изчистването на редове в буферите TLB за индивидуалните страници. Флагът PGE (бит 7 – Page Global Enable) в управляващия регистър CR4 разрешава дефинирането на глобални страници с флага G.

Архитектурата IA-32 осигурява няколко инструкции за управление на трите нива на кеш паметите. Инструкцията INVD изчиства (прави невалидни) всички линии на кешовете, след което генерира специален цикъл по шината, индициращ че всички външни кешове също трябва да бъдат изчистени, т.е. съдържанието на редовете им да бъде маркирано като невалидно. Тази инструкция не предизвиква обратен запис на модифици-

раните редове в паметта и следователно тяхното съдържание ще се загуби. Инструкцията WBINVD първо записва обратно всички модифицирани редове в паметта и след това прави невалидно съдържанието на кешовете L1, L2 и L3. Тя поддържа кохерентност на кеша и главната памет. Тази инструкция генерира два специални цикъла по системната шина за индициране на контролерите на външната кеш памет да осъществят обратен запис на модифицираните данни, след което да маркират съдържанието на външните кешове като невалидно. Инструкциите за явно кеширане PREFETCHH дават възможност на програмата да предложи на процесора да бъде извлечена предварително линия от системната памет в йерархията на кеш паметите. Инструкцията за явно изчистване на кеша CLFLUSH осигурява възможност за избрани линии от кеша да бъдат премахнати от паметта. Инструкциите MOVNTI, MOVNTQ, MOVNTDQ, MOVNTPS, MOVNTPD осъществяват трансфер на съдържанието на универсалните регистри, MMX и XMM регистрите директно в системната памет, без да бъдат кеширани в който и да е кеш от трите нива.

Процесорът осигурява възможности за кеширане в L1, L2 и L3 на произволна област от паметта. За индивидуалните страници или области от системната памет той дефинира и типа на кеширане, известен още и като тип памет. Типовете памет, дефинирани за архитектурата IA-32, са както следва:

- *Силно некеширана* – съдържанието на клетките на системната памет не се кешира. Всички операции за четене и запис в паметта се изпълняват съобразно реда, заложен в програмата. Този тип памет се използва за В/И устройства, използващи адресното пространство на паметта.

- *Некеширана* – има същите характеристики като силно некешираната памет.

- *Комбиниране на записи* – съдържанието на клетките на системната памет не се кешира. Спекулативни операции за четене са разрешени. Операциите за запис могат да бъдат забавени и комбинирани в буфера за запис WC за намаляване на броя на обръщенията към паметта. Този тип управление на кеша е препоръчителен за рамковия видео буфер (обикновено няколко Mbytes), тъй като редът на записите не е важен. Операциите за запис в случая актуализират съдържанието на паметта, за да се използва за графичния дисплей.

- *Запис през кеша* – всички операции за четене и запис от и в системната памет са кеширани. Спекулативни операции за четене са разрешени. Всички операции за запис осъществяват едновременно запис в

линията на кеша и в системната памет. Комбинирането на записите е разрешено. Този тип кеширане е подходящ за рамковите буфери или когато към системната шина са свързани устройства, осъществяващи обръщение към паметта, но не правят „подслушване“ на достъпите до паметта. При този тип кеширане се поддържа кохерентност на вътрешните кешове на процесора и системната памет.

- *Обратен запис* – всички операции за запис и четене от/в системната памет са кеширани. Операциите за четене използват информацията, съдържаща се в линиите на кеша при попадение. Липсите при четене попълват липсващата линия в кеша. Спекулативни операции за четене са разрешени. Тенденцията е по възможност всички операции за запис да бъдат осъществявани в кешовете. Комбинирането на записите е разрешено. Този тип кеширане редуцира трафика по системната шина, тъй като се избягват много ненужни записи в паметта. Последователните записи се акумулират в линията на кеша, а не се придвижват направо към паметта. Съдържанието на модифицираната линия на кеша се записва в паметта по-късно, когато се изпълнява операция за обратен запис. Операция за обратен запис на кеш-линията се стартира, когато кешът е пълен и тази линия трябва да се отстрани, за да освободи място на нова линия. Операция за обратен запис на кеш-линията се стартира и от механизмите за поддържане на кохерентността на паметта. Този тип управление на кеша осигурява най-висока системна производителност, но изисква всички устройства, които осъществяват достъп до системната памет през системната шина, да бъдат в състояние да подслушват достъпите до паметта, за да осигурят кохерентността на кеш паметите и системната памет.

- *Защита на записа* – записите се предават по системната шина и предизвикват съответните линии в кешовете на другите процесори да бъдат маркирани като невалидни. Спекулативни операции за четене са разрешени.

Регистрите за типа на паметта MTRR обвързват определени области от физическото адресно пространство на системната памет с конкретен тип памет (метод на кеширане). Те дават възможност на процесора да оптимизира операциите за различните видове памет като RAM, ROM, рамкова буферна памет и В/И устройства, използващи адресното пространство на паметта. Механизмът с MTRR регистрите разрешава да се дефинират до 96 области във физическата памет, като типът памет за всяка област се задава чрез регистрите за модела MSR.

Процесорите, поддържащи архитектурата IA-32, съхраняват временно всяка операция за запис в буфер за запис. Това подобрява производи-

телността на процесора, тъй като му дава възможност да продължи изпълнението на инструкциите, без да се налага да изчаква завършването на записа в кеша или паметта. Този механизъм също така дава възможност записите да бъдат забавяни, за да се осигури по-ефективно използване на циклите за достъп до паметта. В общия случай, съществуването на буфера за запис е прозрачно за софтуера дори и при мултипроцесорни конфигурации. Процесорът осигурява реда на записите винаги да съответства на реда, заложен в програмата.

Комбинираните записи се задържат във вътрешния буфер WC, който е отделен от кешовете L1, L2 и L3, както и от буфера за запис. Буферът WC не се подслушва и не поддържа кохерентност на данните. Това буфриране се осъществява, за да се осигури възможност на процесора да зареди по-голямо количество модифицирани данни в буфера WC. При многократните записи в една и съща клетка на паметта ще бъде записана само последната стойност в паметта, като междинните стойности ще бъдат изгубени. Процесорите Pentium™ 4 и Xeon™ поддържат по няколко 64-байтови WC буфера. Когато буферът WC е пълен (всички байтове са валидни), процесорът изпълнява експлозивна транзакция за запис на 64 bytes по шината.

Контекстният режим на кеша за данни L1 е характерен за процесора Pentium™ 4 с хипернишкова технология. Базовата входно-изходна система BIOS е отговорна за конфигурирането на този режим. Възможните варианти са адитивен режим и режим на общо ползване. При адаптивния режим са възможни два варианта: двата логически процесора да използват еднакво целия кеш за данни L1 или двата логически процесора да се конкурират за използването на този кеш, като в този случай отделният логически процесор не може да използва целия капацитет на кеша.

Изпълнението на саmomодифициращ се код предизвиква редица проблеми при процесорите с хипернишкова архитектура. Записът на процесора на адрес в сегмента на кода предизвиква маркирането на съответната линия в кеша като невалидна. При процесорите Pentium™ 4 и Xeon™ запис или подслушване на инструкция в кодов сегмент, при която инструкцията е вече декодирана и съхранена в кеша за следи, прави невалидно цялото съдържание на кеша за следи. Следователно изпълнението на програми със саmomодифициращ се код може да предизвика значителна деградация на системната производителност при тези процесори.

В мултипроцесорни конфигурации, при които съществуват няколко подсистеми от кеш памети, проблемът за кохерентността на кешовете е изключително важен. В даден момент копия на съдържанието на клетка

от паметта може да се съдържат в кешовите на няколко процесора. За да се осигури кохерентността на подсистемите кеш памети, архитектурата IA-32 поддържа протокол на кеша MESI. Той се използва както в еднопроцесорни, така и в мултипроцесорни конфигурации.

При протокола MESI се проверяват дефинирани състояния на всяка линия на кеша. Тези състояния са четири (от първите им букви произлиза името на протокола):

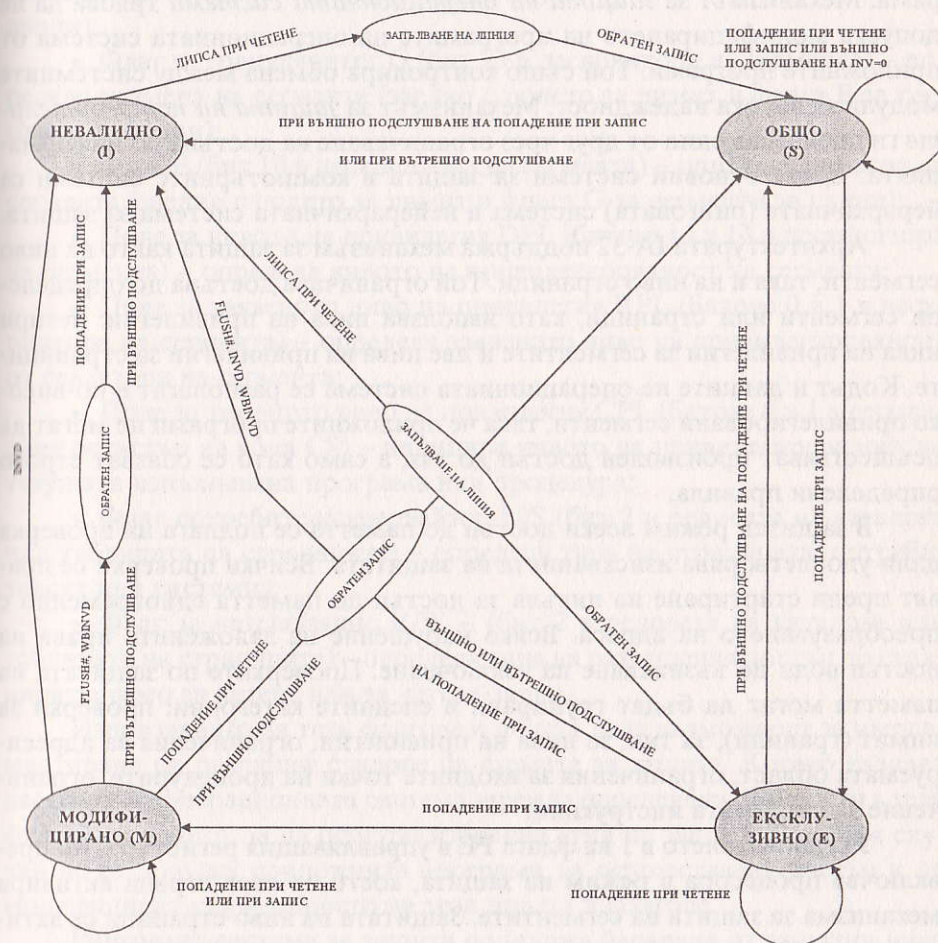
- *Модифицирано (M – Modified)* – линията в кеша съдържа информация, която е модифицирана поради осъществен запис в кеша. Това предизвиква подслушване на системната шина от кешовете и запис на модифицираната линия обратно в паметта, когато при подслушването се получи попадение в линията на кеша;
- *Ексклузивно (E – Exclusive)* – копието в кеша е единствено и никой друг кеш в системата не разполага с друго копие;
- *Общо (S – Shared)* – линията се съдържа евентуално в няколко кеша и всички копия са актуални;
- *Невалидно (I – Invalid)* – състояние, след начално установяване (reset), показва, че състоянието на линията е невалидно.

Всяка линия в кеша има индикатор на състоянието, който показва състоянието на информацията, съдържаща се в кеша. Преходи от едно състояние в друго се предизвикват от операции за запис и четене на локалния процесор или при подслушване на шината, когато друго управляващо устройство на шината иницира цикъл. За да се поддържа кохерентността на множество кешове в системата, е необходимо да се осигурят допълнителни сигнали за комуникация между кешовете.

На фиг. 3.32 е представена диаграма на преходите в кеша за данни. При еднопроцесорната имплементация на протокола MESI всяка линия на кеша може да премине през последователност от състояния след първото си зареждане в кеша за данни L1 или в унифицираните кешове L2 или L3. Преходите на състоянията зависят от вида на достъпите до паметта на процесора или другите управляващи устройства на шината.

При силно свързани паралелни компютърни системи (мултипроцесори) всички процесори ползват обща (глобална) памет. Виртуално при всички мултипроцесорни системи всеки процесор използва локална кеш памет с обратен запис, за да се редуцира трафикът по системната шина. По този начин всеки процесор през по-голямата част от времето осъществява достъп до локалната си кеш памет и рядко до бавната DRAM или ROM системна памет. При системи с множество управляващи устройства по шината, които не използват обратен запис, съществува опасност

от претоварване на трафика по шината от многобройните достъпи до паметта, осъществявани от всяко управляващо устройство на шината. При мултипроцесорни системи повече от един процесорен комплекс може да има копия на едни и същи линии на кеша. Протоколът MESI осигурява кохерентността на множеството кешове и предотвратява ненужното инвалидиране на линиите на кеша. Действията на всеки процесор зависят от текущото състояние на линията на кеша, типа на текущата транзакция по системната шина и състоянието на сигналите за подслушване, генерирани от останалите процесори.



Фиг. 3.32. Диаграма на преходите при протокола MESI в кеша за данни

3.3. Защита

Основните насоки при защитата в компютърната система включват защита на паметта, защита на операционната система от потребителите и защита на потребителите помежду им. Механизмът за защита на паметта трябва да открива всяка грешка при адресирането преди извършването на неразрешен достъп. Всяка инструкция се проверява дали има право на определен вид достъп. Тази проверка се извършва от устройството за управление на паметта, като при възникване на изключение достъпът се санкционира и се извиква съответната обслужваща подпрограма. Механизмът за защита на операционната система трябва да не допуска модифицирането на програмите на операционната система от приложните програми. Той също контролира обмяна между системните модули за висока надеждност. Механизмът за защита на потребителите ги предпазва един от друг чрез ограничаване на достъпа до информацията. Двете основни системи за защита в компютърните системи са йерархичната (ринговата) система и нейерархичната система за защита.

Архитектурата IA-32 поддържа механизъм за защита както на ниво сегменти, така и на ниво страници. Той ограничава достъпа до определени сегменти или страници, като използва нива на привилегии: четири нива на привилегии за сегментите и две нива на привилегии за страниците. Кодът и данните на операционната система се разполагат в по-високо привилегировани сегменти, така че приложните програми не могат да осъществяват произволен достъп до тях, а само като се спазват строго определени правила.

В защитен режим всеки достъп до паметта се подлага на проверка дали удовлетворява изискванията на защитата. Всички проверки се правят преди стартиране на цикъла за достъп до паметта едновременно с преобразуването на адреса. Всяко нарушение на заложените права на достъп води до възникване на изключение. Проверките по защитата на паметта могат да бъдат групирани в следните категории: проверки за лимит (граница), за тип, за нива на привилегии, ограничения за адресируемата област, ограничения за входните точки на процедурите, ограничение за системата инструкции.

Установяването в 1 на флага PE в управляващия регистър CR0 превключва процесора в режим на защита, което от своя страна активира механизма за защита на сегментите. Защитата на ниво страници се активира автоматично с активирането на странирането на паметта при установяването в 1 на флага PG в управляващия регистър CR0.

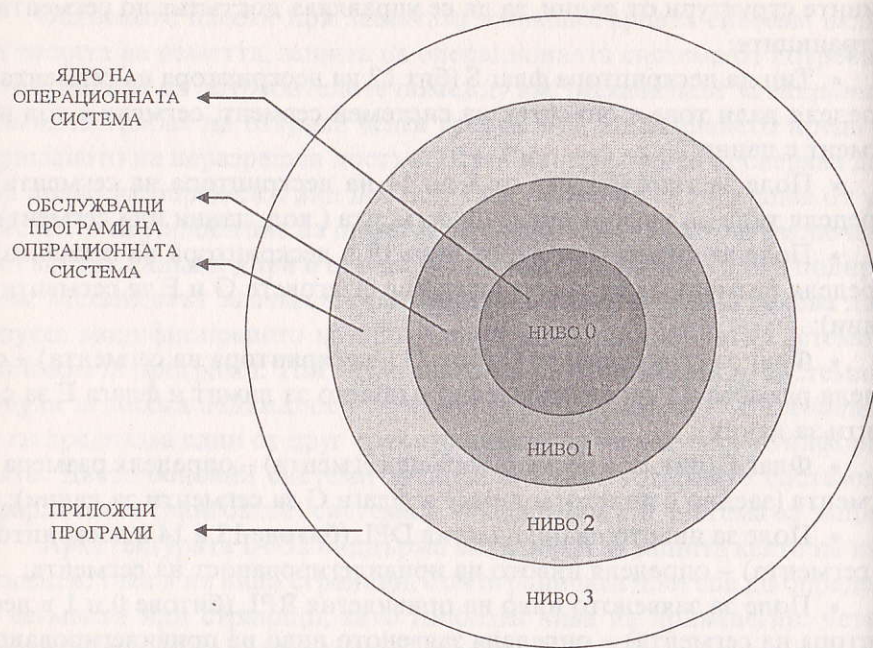
Механизмите за защита използват следните флагове и полета в системните структури от данни, за да се управлява достъпът до сегментите и страниците:

- Тип на дескриптора флаг S (бит 12 на дескриптора на сегмента) – определя дали това е описател на системен сегмент, сегмент с код или сегмент с данни;
- Поле за типа (битове от 8 до 11 на дескриптора на сегмента) – определя типа на информацията в сегмента (код, данни или сегмент);
- Поле за лимит (битове от 0 до 19 в дескриптора на сегмента) – определя размера на сегмента (заедно с флаговете G и E за сегменти за данни);
- Флаг за грануляритет G (бит 23 в дескриптора на сегмента) – определя размера на сегмента (заедно с полето за лимит и флага E за сегменти за данни);
- Флаг E (бит 10 в дескриптора на сегмента) – определя размера на сегмента (заедно с полето за лимит и флага G за сегменти за данни);
- Поле за нивото на привилегия DPL (битове 13 и 14 в дескриптора на сегмента) – определя нивото на привилегированост на сегмента;
- Поле за заявеното ниво на привилегия RPL (битове 0 и 1 в дескриптора на сегмента) – определя заявеното ниво на привилегированост от селектора на сегмента;
- Поле за текущото ниво на привилегия CPL (битове 0 и 1 в сегментния регистър на кода CS) – индицира нивото на привилегированост на текущата изпълнявана програма или процедура;
- Флаг потребител/супервайзор U/S (бит 2 в редовете на каталога или таблицата на страниците) – определя типа на страницата: потребителска или системна;
- Флаг за четене/запис R/W – (бит 2 в редовете на каталога или таблицата на страниците) – определя типа на разрешения достъп до страницата: само за четене или за четене-запис.

Използването на тези флагове и полета дава възможност за имплементиране на различни стилове на схемите за защита. Когато създава дескриптор, операционната система зарежда определени стойности в тези флагове и полета, за да осигури избрания стил на защита. В общия случай не се допуска приложните програми да осъществяват достъп и да модифицират съдържанието на тези полета и флагове.

Ринговата система за защита поддържа йерархия от защитни нива (рингове) с различна привилегированост (фиг. 3.33).

РИНГОВЕ НА ЗАЩИТА



Фиг. 3.33. Нива на йерархията на ринговата защита

Процесорът използва нива на привилегиите, за да предотвратява достъпа на програми или задачи с по-ниско ниво на привилегированост до сегмент с по-високо ниво на привилегированост освен при спазване на строго дефинирани правила. Всяко ниво има отделен стек и указател на стека с цел пълна изолация между отделните нива и отделните дескрипторни таблици. Когато процесорът установи нарушение на нивото на привилегии, възниква изключение по обща защита (#GP).

Операционната система заема най-високите нива в ринговата йерархия, за да се защити от приложните програми. Централният ринг има най-високото ниво на привилегированост 0 и се използва от сегментите, съдържащи ядрото на операционната система (код, данни и стекове). Първото и второто ниво се използват от системните обслужващи програми, а най-ниското ниво (ниво 3) на привилегированост се използва от приложните програми. Системи, които използват само две от възможните четири нива на привилегированост, трябва да използват нулевото и третото нива.

За да извършва проверките за нивата на привилегии на сегментите на кода и данните, процесорът разпознава следните типове нива на привилегированост:

- *Текущо ниво на привилегированост CPL* – това е нивото на привилегированост на текущата изпълнявана програма или задача. То се съхранява в битове 0 и 1 на сегментните регистри CS и SS. В общия случай то е равно на нивото на привилегированост на сегмента на кода, за който се извличат инструкции. Процесорът променя текущото ниво на привилегированост, когато управлението се предава на кодов сегмент с различно ниво на привилегированост.

- *Ниво на привилегированост на дескриптор DPL* – представлява нивото на привилегированост на сегмента. Съхранява се в полето DPL на дескриптора на сегмента. Когато текущият изпълняван сегмент на кода прави опит за достъп до друг сегмент, DPL на сегмента, към който се прави обръщението, се сравнява с CPL и RPL на сегмента, заявил достъпа. При обръщение към сегмент за данни DPL показва най-високото ниво на привилегированост на сегмента, към който е разрешен достъпът. Например, ако DPL на сегмент за данни е 1, само текущо изпълнявана програма при CPL=0 или CPL=1 може да осъществява достъп до този сегмент данни.

- *Заявено ниво на привилегированост CPL* – присвоява се на сегментните селектори и се съдържа в битове 0 и 1 на сегментния селектор. Процесорът извършва проверки на RPL и CPL, за да определи дали достъпа до сегмента е разрешен.

Нивата на привилегированост се проверяват при зареждането на сегментния селектор в сегментния регистър. Проверките за достъп до данните се различават от проверките при предаване на управление между различни кодови сегменти.

Привилегированите инструкции са защитени за изпълнение от приложните програми. Тези инструкции управляват функционирането на системата (напр. зареждането на системните регистри). Те могат да бъдат изпълнявани само на най-високото ниво на привилегии CPL=0. Ако се направи опит за изпълнение на привилегирована инструкция, когато CPL≠0, възниква изключение по обща защита (#GP). При архитектурата IA-32 привилегированите инструкции са:

- LGDT – зареждане на регистъра на глобалната дескрипторна таблица;
- LLDT – зареждане на регистъра на локалната дескрипторна таблица;

- LTR – зареждане на регистъра на задачата;
- LIDT – на регистъра на дескрипторната таблица на прекъсванията;
- MOV (управляващи регистри) – зареждане и запис на управляващите регистри;
- LMSW – зареждане на думата за състояние на машината;
- CLTS – нулиране на флага за превключване на задача в управляващия регистър CR0;
- MOV (регистра за настройка) – зареждане и запис на регистрите за настройка;
- INVD – съдържанието на кеша става невалидно, без да се запише обратно в паметта;
- WBINVD – съдържанието на кеша става невалидно, като предважително се записва съдържанието му обратно в паметта;
- INVLPG – ред в TLB се маркира като невалиден;
- HLT – спиране на процесора;
- RDMSR – четене на регистрите, специфични за модела;
- WRMSR – запис в регистрите, специфични за модела;
- RDPMSR – четене на брояча за мониторинг на производителността;
- RDTSC – четене на брояча за отбелязване на времето.

Защитата на ниво страници може да бъде осъществявана самостоятелно или в комбинация със защитата на сегментите. Аналогично на защитата при сегментите всеки достъп до паметта се проверява дали съответства на правата за достъп. Всички проверки се правят преди да е стартиран цикълът за достъп до паметта. Нарушаването на правата за достъп предотвратява достъпа и предизвиква изключение при грешка на страница. Проверките се правят паралелно с преобразуването на адреса, така че не предизвикват понижаване на производителността.

Процесорът прави проверки за защитата на страниците на две нива: ограничаване на адресируемата област (режим супервайзор и режим потребител) и проверки за типа на страницата (разрешена само за четене или за четене/запис).

Информацията за защита на страниците се съдържа в два флага в редовете на каталозите или таблиците на страниците: флаг за четене/запис (бит 1) и флаг режим супервайзор/потребител (бит 2).

Механизмът за защита на ниво страници ограничава достъпа на страниците на две нива:

- *режим супервайзор* (U/S флагът е нулиран) – това е най-привилегирован режим, който се използва от операционната система или от

други системни програми (като например драйверите на устройствата) и защитени системни данни (като например таблиците на страниците);

- *потребителски режим* (U/S флагът е установен в 1) – това е най-ниско привилегирован режим, който се използва от приложните програми и данни.

Съответствието между нивата на привилегированост на сегментите и тези на страниците е организирано по следния начин: ако текущото ниво на привилегированост на процесора CPL е 0, 1 или 2, той е в режим супервайзор и има достъп до всички страници, ако е в потребителски режим се разрешава достъп само до потребителските страници. За обезпечаване на този механизъм за защита сегментите на кода и тези на данните трябва да използват поне две нива на привилегии: ниво 0 за кода и данните на супервайзора и ниво 3 за кода и данните на потребителя.

Механизмът за защита на ниво страници разпознава два типа страници:

- с разрешен достъп само за четене (флагът R/W=0);
- с разрешен достъп за четене и запис (флагът R/W=1).

След инициализация (reset), когато процесорът е в режим супервайзор и флагът WP=0 в управляващия регистър CR, всички страници са разрешени за четене и запис. Когато процесорът е в режим потребител, той може да осъществява запис само в потребителските страници, които са разрешени за запис. Страниците на супервайзора са забранени както за четене, така и за запис в потребителски режим.

За всяка страница атрибутите за защита, заложи в таблицата от първо ниво, могат да се различават от тези, заложи в таблицата от второ ниво. Процесорът прави проверки за защита на дадена страница в каталога и в таблицата на страниците.

Следните типове достъп се проверяват независимо от текущото ниво на привилегированост CPL на процесора:

- достъп до сегментните дескриптори в глобалната GDT и локалната LDT дескрипторни таблици, както и до дескрипторната таблица на прекъсванията;
- достъп до стек в по-високо ниво на привилегированост по време на извикване (CALL) на програма от друго ниво или при извикване на подпрограма, обслужваща изключение или прекъсване при промяна на нивата на привилегированост.

При комбинирана защита по таблици и сегменти процесорът първо прави проверките за защита на сегментите и след това тези, свързани със защитата на страниците. Ако процесорът установи нарушение на прави-

лата на защита, независимо дали това става на ниво сегменти или на ниво страници, достъпът се санкционира (отказва) и възниква изключение по защита. Ако възникне изключение по защита на сегментите, правата за достъп до страниците не се проверяват.

Защитата на ниво страници не може да отмени правата за достъп, заложиени на ниво сегмент. Например, ако даден сегмент на кода е разрешен само за четене, установяването на флага за четене/запис R/W за неговите страници няма да промени правата за достъп и опитите за достъп до тези страници ще бъдат санкционирани от защитата на ниво сегменти.

Защитата на ниво страници може да се използва за подобряване на ефективността на защитата на ниво сегменти. Така например ако голям сегмент за данни, допускащ четене/запис, е страниран, то механизмът за защита на страниците може да бъде използван за защита по запис на индивидуалните страници. В този ред на мисли, правата за достъп могат да бъдат стеснявани в посока от нивото на сегментите към нивото на страниците.

3.4. АРХИТЕКТУРНА ПОДДРЪЖКА НА МУЛТИПРОЦЕСОРНИ СИСТЕМИ

Архитектурата на съвременните процесори осигурява поддръжка на конфигурирането на мултипроцесори, които представляват тясно свързани паралелни компютърни системи с обща памет. Архитектурата IA-32 поддържа механизми за управление и подобряване на производителността на множество процесори, свързани към системната шина.

Тези механизми включват:

- заключване на шината и/или управление на кохерентността на кеш паметите за осъществяване на атомични (неделими) операции в системната памет;
- инструкции за сериализация;
- програмируем контролер на прекъсванията, вграден в чипа на процесора;
- кеш на второ ниво L2, вграден в чипа на процесорите Pentium™ 4, Intel® Xeon™ и фамилията процесори P6, който е тясно свързан с процесора;
- кеш от трето ниво при процесора Xeon™, който е вграден в чипа на процесора и е тясно свързан с него;

- хипернишкова технология, която е разширение на архитектурата IA-32 и осигурява възможност един процесор да изпълнява едновременно две или повече нишки.

Тези механизми са особено полезни при симетрични мултипроцесори (паралелни системи, в които всички процесори са еднакви), но също така могат да се използват и за приложения, включващи общи конфигурации от процесор с IA-32 архитектура и специализиран процесор като например комуникационен процесор, графичен процесор, видео процесор и др., свързани към обща системна шина.

Основните цели на механизмите за поддръжка на мултипроцесори са:

- *Поддръжка на кохерентността на системната памет.* Когато два или повече процесора правят опит едновременно да осъществят достъп до един и същ адрес в системната памет, е необходим специален механизъм за комуникация или протокол за достъпа до паметта, за да се осигури кохерентността на данните или в конкретни случаи да се осигури възможност на някой от процесорите да заключи временно достъпа до определена клетка от паметта.

- *Поддръжка на кохерентността на кеша.* Когато даден процесор осъществява достъп до данни, кеширани от друг процесор, той не трябва да получава некоректна информация. Ако той модифицира данни, всички други процесори, които осъществяват достъп до тях, трябва да получат модифицираните данни.

- *Осигуряване на предсказуем ред на записите в паметта.* Тази цел е актуална в случаите, когато редът на записите в паметта трябва да следва прецизно реда, заложен в програмата.

- *Разпределяне на обработката на прекъсванията между процесорите.* Когато множество процесори функционират паралелно в системата е препоръчително да се използва централизиран механизъм за получаване на заявките за прекъсване и за разпределянето им за обслужване от наличните процесори.

- *Повишаване на системната производителност* чрез оползотворяване на паралелизма на ниво нишки и възможностите за обработка на множество процеси при съвременните операционни системи и приложения програми.

Заклучените атомични операции в клетки от системната памет се използват обикновено за управление на общи структури данни (като например семафори, сегментни дескриптори, таблици на страниците) в случаите, когато два или повече процесора се опитват едновременно да модифицират един и същ флаг или едно и също поле. За осъществяване-

то на атомични операции процесорите могат да използват три независими механизма:

- Гарантирани атомични операции;
- Заклучване на шината с използване на сигнала LOCK# и префикса LOCK на инструкцията;
- Протоколи за поддържане на кохерентността на кешовете, които гарантират изпълнението на атомични операции над кеширани структурни данни (заклучване на кеша).

Определени базови транзакции в паметта като например четенето или запис в системната памет на байт се осъществяват гарантирано винаги като атомични операции. Веднъж стартирана, процесорът гарантира, че операцията ще бъде завършена преди да бъде предоставен достъп на друг процесор или агент по шината до тази клетка на паметта. Процесорът също така поддържа заключване на шината за избрани операции в паметта като четене-модифициране-запис в обща област на паметта. Интензивно използваните клетки от паметта често се кешират в кешовете от първо и второ ниво и следователно атомичните операции могат да бъдат осъществявани в кешовете на процесора без заключване на шината. В тези случаи протоколите, поддържащи кохерентността на кешовете в системата, са отговорни за предизвикване на адекватни действия на другите процесори, които кешират клетката, над която се реализира атомичната операция.

В архитектурата IA-32 са гарантирани автоматично следните атомични операции:

- Четене или запис на байт;
- Четене или запис на дума, разположена на 16-битова граница;
- Четене или запис на двойна дума, разположена на 32-битова граница;
- Четене или запис на четворна дума, разположена на 64-битова граница.

Процесорът разполага със сигнала LOCK#, който се активира автоматично при изпълнението на критични операции за заключване на системната шина. Докато този сигнал е активен, заявките за управление на шината на другите процесори или агенти по шината се блокират. По програмен път заключването на шината може да бъде активирано чрез използването на префикса за инструкция LOCK.

Процесорът автоматично заключва шината при изпълнението на следните операции:

- *Изпълнение на инструкцията XCHG*, която се обръща към паметта;

- *При установяване в 1 на флага B (busy) на дескриптора на сегмента на задачата TSS*. Процесорът тества и установява в 1 този флаг в полето за типа на дескриптора TSS, когато превключва задача. За да се осигури, че два процесора няма да превключат към една и съща задача едновременно, се използва заключването на шината при тестването и установяването на този флаг.

- *При модифициране на сегментните дескриптори*. Целта е да се предотврати едновременното модифициране на даден сегментен дескриптор от два процесора.

- *При актуализация на каталозите или таблиците на страниците*. Процесорът използва заключени цикли за установяване на флаговете за достъп и за „мръсна“ страница в редовете на каталозите и таблиците на страниците.

- *Потвърждаване на прекъсванията*. Процесорът осигурява заключен цикъл за изпращането на вектора на прекъсване от контролера на прекъсванията по шината за данни, за да осигури, че няма да бъдат подавани други данни, докато трае предаването на вектора на прекъсването.

Заклучването на шината може да се управлява по програмен път, като се използва префиксът LOCK пред инструкции за модифициране на клетки в паметта. Изключение за невалиден код на операция се генерира, когато се използва този префикс за друга инструкция или когато не се осъществява запис в паметта. Заклучването на шината може да се активира по програмен път при следните инструкции:

- Инструкции за тестване и модифициране на бит BTS, BTR и BTC.
- Инструкции за размяна XADD, CMPXCHG и CMPXCHG8B.
- Префиксът LOCK се използва автоматично при инструкцията XCHG.

- Аритметическите и логическите инструкции с един операнд INC, DEC, NOT и NEG.

- Аритметическите и логическите инструкции с два операнда ADD, ADC, SUB, SBB, AND, OR, XOR.

Софтуерът трябва да използва достъп до семафори, като използва еднакви адреси и еднакви дължини на операндите. Времетраенето на заключената операция обхваща толкова цикъла, колкото е необходимо за завършване на критичната операция. „Заклучените“ инструкции могат да бъдат използвани за синхронизация на процесорите, когато единият

процесор записва данни, които трябва да бъдат прочетени от друг процесор.

Когато процесорът записва код в текущо изпълняван сегмент на кода с намерението да го изпълни, този код се нарича саомодифициращ се код (self-modifying code). Когато един процесор записва код в текущо изпълняван сегмент на кода на друг процесор с намерението вторият процесор да го изпълни, се нарича кръстосан модифициращ код (cross-modifying code). Изпълнението на саомодифициращ се код, както и на кръстосан модифициращ код, води до значително понижаване на системната производителност, като степента на понижение зависи от честотата на модификациите и специфичните характеристики на кода.

Редът на достъпите се отнася до реда, в който процесорът изпълнява четенето (load) и записите (store) от/в системната памет през системната шина. Програмният ред се отнася до стриктното спазване на реда на записите и четенията през системната шина съгласно инструкционния поток. Той се нарича още „силно“ подреждане на достъпите до паметта. За да се оптимизира изпълнението на инструкциите, е възможно да се поддържа по-малко стриктно програмният ред на достъпите. При Pentium™ 4 и Xeon™ процесорите спазват програмният ред на записите с изпращане в буфера за запис.

В мултипроцесорни конфигурации се спазват следните правила за реда на достъпите до паметта:

- Индивидуалните процесори използват същите правила за реда както еднопроцесорните системи;
- Записите на един и същ процесор се наблюдават в същия ред от останалите процесори;
- Записите на индивидуалните процесори през системната шина не спазват определен ред помежду си.

Архитектурата IA-32 осигурява няколко механизма за „засилване“ или „отслабване“ на модела на спазване на програмния ред:

- Входно-изходните инструкции, инструкциите за заключване на шината, префиксът LOCK и инструкциите за сериализация налагат посилен модел на спазване на реда на достъпите до паметта;
- Регистрите за типа на паметта MTTR могат да се използват за засилване или отслабване на модела на спазване на реда на достъпите до паметта;
- Таблицата с атрибутите на страницата може да се използва за засилване на модела на спазване на реда на достъпите до паметта за определена страница или група страници.

Входно-изходните устройства, свързани към системната шина, имат високи изисквания към реда, в който се извършват записите в техните входно-изходни буфери. Изпълнението на инструкции за вход и изход (IN и OUT) налагат стриктно спазване на програмния ред. Преди изпълнението на В/И инструкция процесорът изчаква да завърши изпълнението на всички инструкции в програмата, предхождащи В/И инструкция, както и да се извършат всички буферирани записи в паметта. Изпълнението на всички инструкции, следващи В/И инструкция в програмата, не започва, докато процесорът не установи, че изпълнението ѝ не е завършило. Операциите, съпроводени от заключване на шината, аналогично на В/И операции изчакват завършването на всички предходни инструкции и предаването на буферираните записи в паметта.

Синхронизацията в програмите може да бъде осъществена чрез *инструкции за сериализация*. Тези инструкции се използват за критични секции или граници на задачи. Те принуждават процесора да изчака завършването на всички предходни инструкции и изпразването на буфера за запис преди да се осъществи преход към нова секция в кода или преди да се извърши контекстно превключване. Изпълнението на инструкциите за сериализация при Pentium 4 и Xeon ограничават спекулативните изчисления, тъй като резултатите, получени от спекулативно изпълнените инструкции, се загубват.

Непривилегированите инструкции SFENCE, LFENCE, MFENCE осигуряват ефективен начин за осигуряване на реда на четенията и записите от/в паметта между подпрограми, генериращи данни и подпрограми, които ги консумират. Инструкцията SFENCE сериализира всички операции за запис, които я предхождат в инструкционния поток, но не оказва влияние върху операциите за четене. Инструкцията LFENCE сериализира всички операции за четене, които я предхождат в инструкционния поток, но не оказва влияние върху операциите за запис. Инструкцията MFENCE сериализира всички операции за запис и четене, които я предхождат в инструкционния поток на програмата.

Архитектурата IA-32 осигурява също така привилегировани инструкции за поддържане на реда на достъпите като MOV (управляващите регистри и регистрите за настройка), INVD, INVLPG, WBINVD, LGDT, LLDT, LIDT, LTR, както и непривилегированите инструкции CPUID, IRET, RSM. Инструкцията CPUID може да бъде използвана на всяко ниво на привилегированост за сериализация на изпълнението на инструкциите, без да се оказва влияние на програмния поток с изключение на съдържанието на регистрите EAX, EBX, ECX, EDX.

При използването на техники за предсказване на прехода за повишаване на производителността се извлича предварително инструкцията на прехода преди да се изпълни самата инструкция за преход. Като следствие изпълнението на инструкциите не е детерминистично сериализирано при изпълнението на инструкцията за преход.

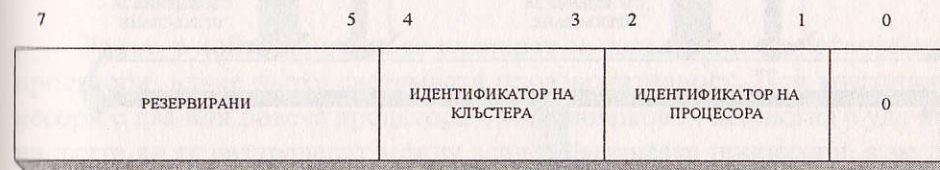
В мултипроцесорните системи, когато даден процесор променя ред в таблица или каталог на страниците, промените трябва да станат достояние на всички останали процесори. Предаването на промените в редовете на каталозите или таблиците на страниците към другите процесори може да се осъществи чрез използване на семафори в общата памет или междупроцесорни прекъсвания IPI.

Архитектурата IA-32 дефинира протокол за инициализация на мултипроцесорни системи, наречен *Multiprocessor Specification Version 1.4*. Този протокол дефинира два типа процесора: процесор за първоначално зареждане BSP и приложни процесори AP. След включване на захранването или RESET системният хардуер динамично избира един от процесорите по системната шина за BSP, а останалите процесори за AP. Процесорът BSP изпълнява кода за първоначално зареждане, за да конфигурира средата за контролерите на прекъсване, конфигурира системните структури данни и започва инициализация на приложните процесори. След инициализацията на процесорите BSP и AP процесорът BSP започва изпълнението на кода за инициализация на операционната система. След включване на захранването или RESET приложните процесори осъществяват минимална самоконфигурация, после изчакват сигнал за стартиране (съобщение SIPI) от процесора BSP. След получаването на съобщението SIPI приложните процесори изпълняват код за конфигуриране, който завършва в поставянето на приложните процесори в състояние HALT.

При наличие на процесори с хипернишкова технология протоколът третира всеки от логическите процесори по системната шина като отделен процесор с уникален идентификатор на контролера за прекъсване. По време на първоначалното зареждане един от логическите процесори се избира за BSP, а останалите логически процесори се дефинират като приложни процесори.

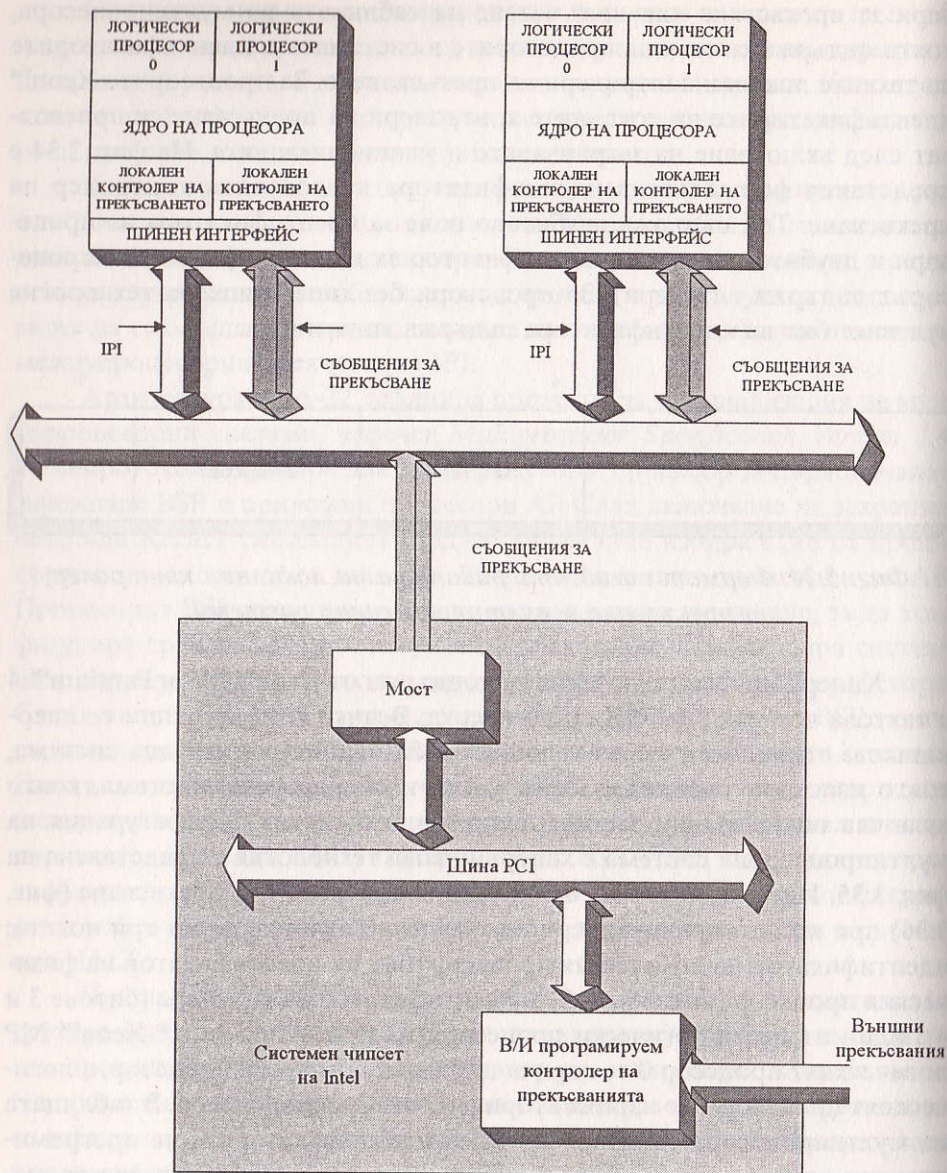
След като базовата входно-изходна система привърши изпълнението на протокола за инициализация, всеки процесор може да бъде идентифициран чрез уникалния идентификатор на неговия контролер за прекъсване. Софтуерът може да осъществява достъп до тези идентификатори посредством четене на идентификаторите на локалните контро-

лери за прекъсване или чрез четене на таблицата на мултипроцесора, която съдържа списък на процесорите в системата и идентификаторите на техните локални контролери на прекъсването. За процесорите Xeon™ идентификаторите на локалните контролери на прекъсване се присвояват след включване на захранването и инициализацията. На фиг. 3.34 е представен форматът на идентификатора на локалния контролер на прекъсване. Той съдържа двубитово поле за идентификатора на процесора и двубитово поле за идентификатор за клъстера (ако мултипроцесорът съдържа клъстери). За процесори без хипернишкова технология нулевият бит на идентификатора съдържа винаги нула.



Фиг. 3.34. Формат на идентификатора на локалния контролер на прекъсване в мултипроцесорни системи

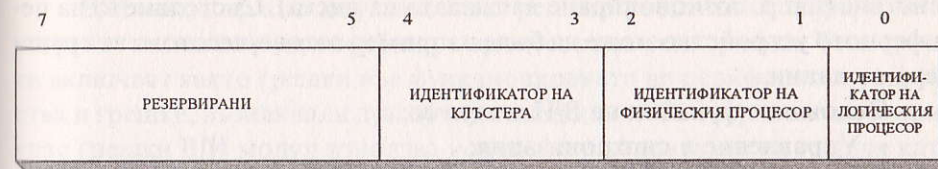
Хипернишкова технология се поддържа от Xeon™ MP и Pentium™ 4 с тактова честота 3.06 GHz и по-висока. Всички конфигурации с хипернишкова технология изискват чипсет и базова входно/изходна система, която използва тази технология, както и операционна система, която включва оптимизации за хипернишкова технология. Конфигурация на мултипроцесорна система с хипернишкова технология е представена на фиг. 3.35. Идентификаторът на локалния контролер за прекъсване (фиг. 3.36) при поддръжка на хипернишкова технология съдържа три полета: идентификатор на логическия процесор (бит 0), идентификатор на физическия процесор (битове 1 и 2) и идентификатор на клъстера (битове 3 и 4). Един от двата логически процесори на процесора Intel® Xeon™ MP логическият процесор 0 се нарича първичен логически процесор, а логическият процесор 1 се нарича вторичен логически процесор. В таблицата на мултипроцесора са включени само идентификаторите на програмируемите контролери на прекъсванията на първичните логически процесори на физическите процесори. Идентификаторите на всички логически процесори са включени в таблицата ACPI, която се конструира от базовата входно/изходна система при изпълнението на протокола за инициализация на мултипроцесорната система.



Фиг. 3.35. Мултипроцесор с хипернишкова технология

Форматът на таблицата ACPI се базира на спецификацията ACPI, която представлява индустриален стандарт за спецификация на конфигурации на платформи за мултипроцесорни системи. В тази таблица иденти-

фикаторите на първичните логически процесори заемат първата половина, следвани от идентификаторите на вторичните логически процесори.



Фиг. 3.36. Формат на идентификатора на локалния контролер за прекъсване при поддръжка на хипернишкова технология

Редът, в който нишките се планират за изпълнение от логическите процесори, влияе върху системната производителност. При мултипроцесори с два или повече процесора с хипернишкова технология е удачно нишките да се разпределят между всички физически процесори, а не да се концентрират в логическите процесори на един или два физически процесора.

3.5. ВХОДНО-ИЗХОДНА СИСТЕМА

Входно-изходната система на компютъра осигурява интерфейса му с външния свят. Тя включва множество входно-изходни модули. Всеки В/И модул осигурява интерфейса към системната шина и управлява едно или повече периферни устройства. Компютърната система може да обхваща широк спектър от периферни устройства, всяко от които има различни електромеханични характеристики и различно бързодействие. Скоростта на трансфер при периферните устройства е значително по-малка от тази на процесора или на паметта. Затова те не се свързват към високоскоростната системна шина, а към значително по-бавна входно-изходна шина. Освен това обикновено периферните устройства използват различен формат на данните и различна дължина на основните информационни единици от компютъра, към който те са свързани.

Интерфейсът на В/И модул включва сигнали за управление, статус (състояние), данни и адреси. От една страна, той трябва да осигури интерфейса към компютърната система, а от друга страна – интерфейса към конкретното периферно устройство. Управляващите сигнали определят конкретната операция, която трябва да извърши периферното устройство като например изпращане на данни към В/И модул (INPUT,

READ), приемане на данни в периферното устройство (OUTPUT, WRITE), индициране на състоянието на периферното устройство или специфично действие (напр. позициониране на главата на диска). Състоянието на периферното устройство може да бъде например готово/неготово за трансфера на данни.

Основните функции на В/И модул са:

- Управление и синхронизация;
- Комуникация с процесора;
- Комуникация с периферното устройство;
- Буфериране на данните;
- Откриване на грешки.

Комуникацията между процесора и В/И модул включва следните стъпки:

- *Декодиране на командата.* В/И модул получава команди от процесора под формата на сигнали по управляващата шина. Някои от тези команди са съпроводени от параметри, подавани по шината за данни. Така например устройството на диска може да получи команда за четене на сектор, запис на сектор и др.

- *Данни.* Обменът на данните между процесора и В/И модул се осъществява по шината за данни.

- *Индициране на статуса.* Поради ниското бързодействие на периферните устройства е важно да се знае във всеки момент какво е неговото състояние: заето с изпълнението на задача (BUSY), готов за изпълнението на задача (READY) или възникнала грешка (ERR).

- *Разпознаване на адреса.* В/И модул трябва да може да разпознава уникалните адреси на периферните устройства, които управлява.

От друга страна В/И модул трябва да комуникира с периферното устройство, което управлява, като подава команди, получава информация за състоянието му и обменя данни с него.

Разликата между скоростта на трансфера на данни при процесора и главната памет от една страна и периферните устройства от друга страна е значителна и може да варира в широки граници. Ето защо една от важните задачи, които В/И модул изпълнява, е буферирането на данните. Обикновено подаването на данни от главната памет се осъществява в експлозивни цикли. Те се буферират във В/И модул и се подават към периферното устройство със скорост, съобразена с неговото бързодействие. В посоката от периферното устройство към паметта данните също се налага да бъдат буферирани, за да не се забавя главната памет от ниските скорости на трансфер. Следователно В/И модул трябва да е в състо-

яние да функционира с две скорости – тази на главната памет и тази на периферното устройство.

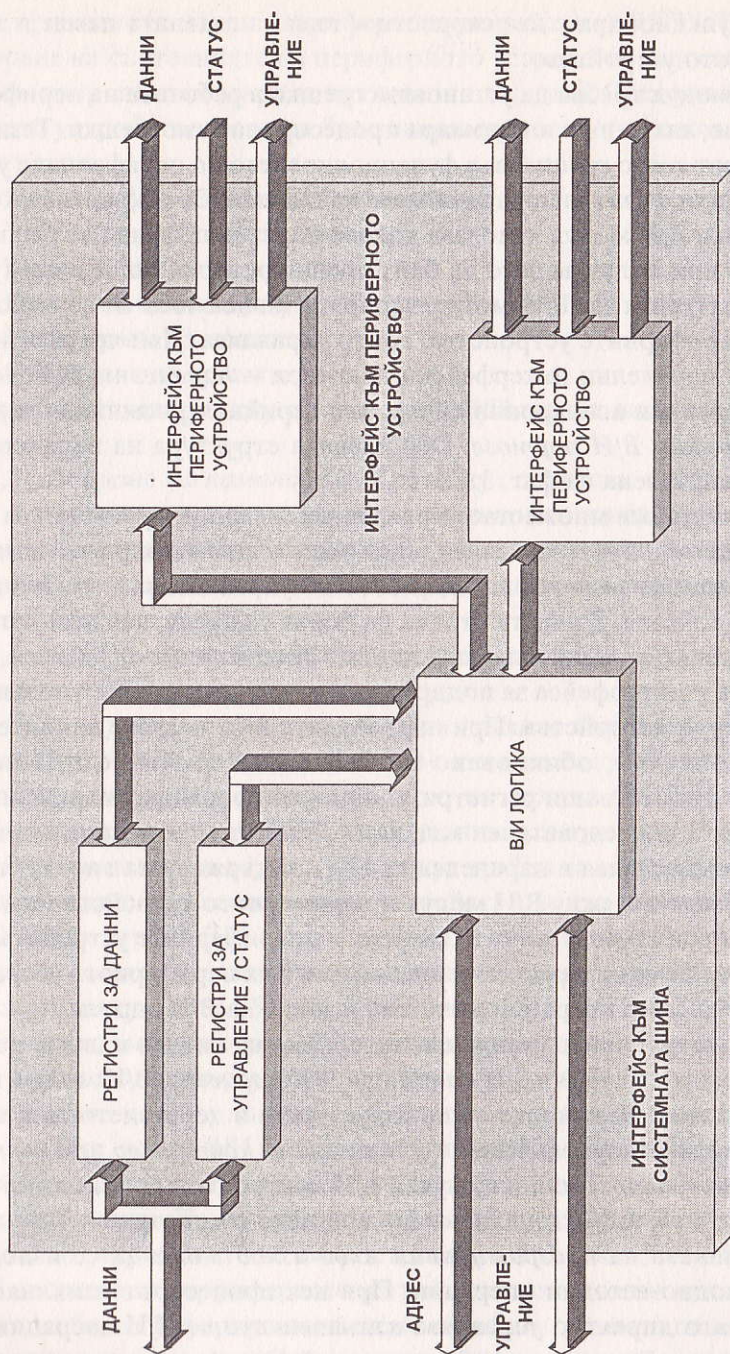
В/И модул трябва да установява грешки в работата на периферното устройство, както и да информира процесора за тези грешки. Тези грешки включват както грешки във функционирането на периферните устройства и грешки, възникнали при обмена на данните. За откриване на втория клас грешки В/И модул използва кодове за установяване на грешки като например при получаването на байт прави проверки по четност (parity).

Структурата на В/И модул варира в зависимост от сложността и броя на периферните устройства, които управлява. Най-често се използват т.нар. паралелни интерфейсни адаптери – *паралелни В/И портове* или универсални асинхронни/синхронни серийни приемници – *предаватели – серийни В/И портове*. Обобщената структура на паралелен В/И порт е представена на фиг. 3.37.

Той съдържа множество управляващи регистри, регистри за състоянието, както и буфер за данни. В/И порт съдържа адресен декодер, с чиято помощ се разпознават адресите на управляваните от него периферни устройства. В лявата страна на порта се представя логиката и интерфейса, осигуряващи комуникацията с процесора, а от дясната страна – логиката и интерфейса за поддръжка на комуникацията с управляваните периферни устройства. При паралелните В/И портове данните се обменят паралелно, обикновено по байтове. Серийните В/И портове съдържат преместващи регистри, които трансформират паралелния код на данните в последователен код, като обменът между процесора и В/И порт се осъществява в паралелен код за поддържане на висока скорост, докато обменът между В/И модул и периферното устройство е в последователен код поради ниската скорост на периферните устройства. Така процесорът не се интересува от спецификата на периферното устройство, а изпълнява само операции за четене и запис на В/И адреси.

В/И модул, който осигурява интерфейс на високо ниво за централния процесор, се нарича *В/И канал* или *В/И процесор*. В/И модул, който е по-примитивен по отношение на структурата и действието си и изисква по-детайлизирано управление, се нарича *В/И контролер* или *контролер на устройството*. В общия случай В/И контролери се използват в микрокомпютрите, а В/И канали – в по-мощните компютри.

Техниката на *програмираня вход-изход* може да се използва за всички входно-изходни операции. При нея процесорът изпълнява програма, която директно управлява изпълнението на В/И операция включително тестване на състоянието на устройството, изпращане на команда



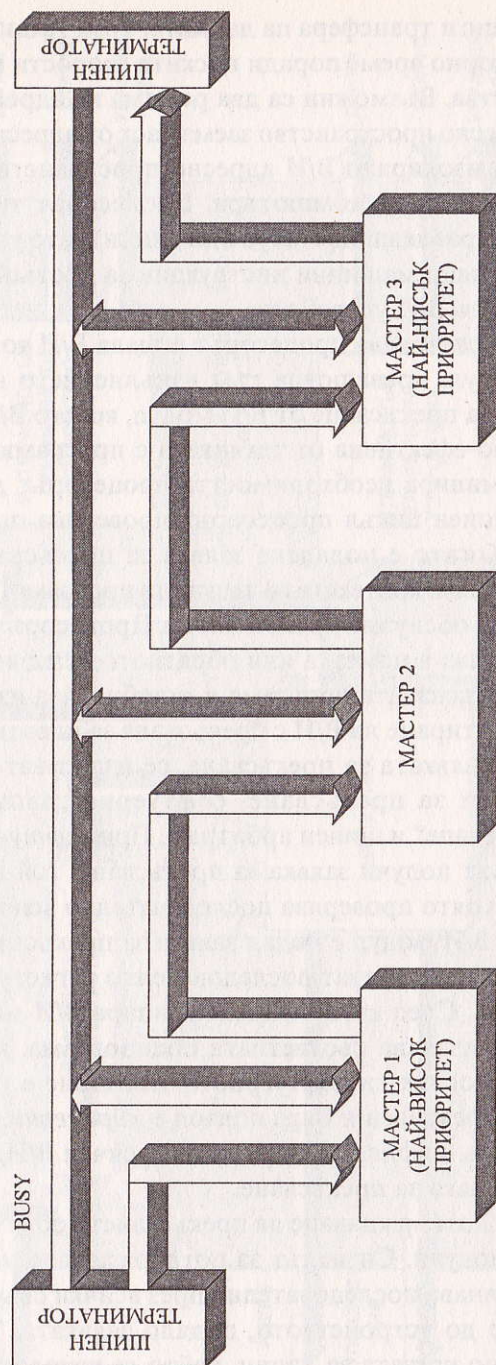
Фиг. 3.37. Обобщена структура на паралелен В/И порт

за четене или запис и трансфера на данните. Тази техника води до големи загуби на процесорно време поради ниските скорости на работа на периферните устройства. Възможни са два режима на адресиране на В/И устройства: В/И адресно пространство заема част от адресното пространство на компютъра и изолирано В/И адресно пространство, отделно от адресното пространство на компютъра. Процесорът третира регистрите на В/И модули (управляващи/статус и за данни) като клетки от паметта и използва едни и същи машинни инструкции за достъп както до клетките в паметта така и до В/И устройства.

При *В/И с прекъсване* процесорът подава В/И команда за изпълнение към В/И модул, превключва към изпълнението на други задачи и получава сигнал за прекъсване от В/И модул, когато В/И обмен завърши. Тази техника е по-ефективна от техниката с програмиран вход-изход, тъй като се елиминира необходимостта процесорът да чака. В края на всеки инструкционен цикъл процесорът проверява постъпилите заявки за прекъсване. Когато е подадена заявка за прекъсване от В/И модул, процесорът съхранява контекста на текущата програма (програмния брояч и регистрите си) и обслужва прекъсването. Процесорът чете думата от В/И модул и я записва в паметта или обратното, след което възстановява контекста на прекъснатата програма и възобновява изпълнението ѝ.

За имплементиране на В/И с прекъсване за идентифицирането на В/И модул, подал заявката за прекъсване, се използват няколко подхода: множество линии за прекъсване, софтуерно „запитване“ (polling), хардуерно „запитване“ и шинен арбитраж. При *софтуерното запитване* когато процесорът получи заявка за прекъсване, той извиква обслужващата програма, която проверява последователно всички В/И модули, за да установи кой В/И модул е подал заявка за прекъсване. Тази проверка се осъществява, като се четат последователно регистрите на състоянието на В/И модули. След като се идентифицира В/И модул, предизвикал прекъсването, се извиква съответната подпрограма, която да го обслужи. Очевиден недостък на софтуерното запитване е голямото време за реализация. По-ефективен и бърз подход е *хардуерният полинг* (маргаритка), представен на фиг. 3.38, при който всички В/И модули използват обща линия за заявка за прекъсване.

Линията за потвърждаване на прекъсването обхожда последователно всички В/И модули. Сигналът за потвърждаване на прекъсването от процесора преминава последователно през всички свързани В/И модули, докато достигне до устройството, подало заявката. То отговаря, като предава вектор по шината за данни, който се използва от процесора да



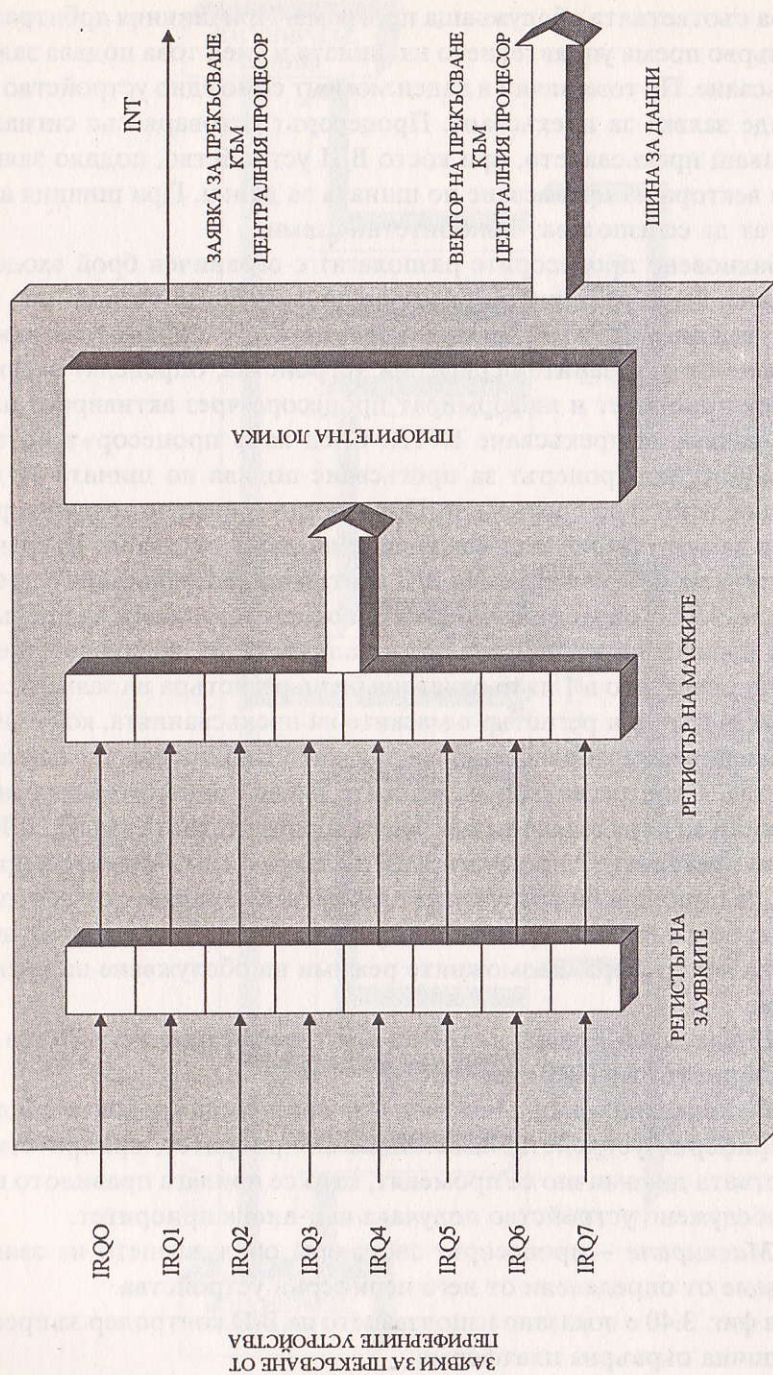
Фиг. 3.38. Хардуерен полинг (маргаритка)

активира съответната обслужваща програма. При шинния арбитраж В/И модул първо поема управлението на шината и след това подава заявката за прекъсване. По този начин в даден момент само едно устройство може да подаде заявка за прекъсване. Процесорът отговаря със сигнал, потвърждаващ прекъсването, при което В/И устройство, подало заявката, предава вектора на прекъсване по шината за данни. При шинния арбитраж могат да се използват приоритетни схеми.

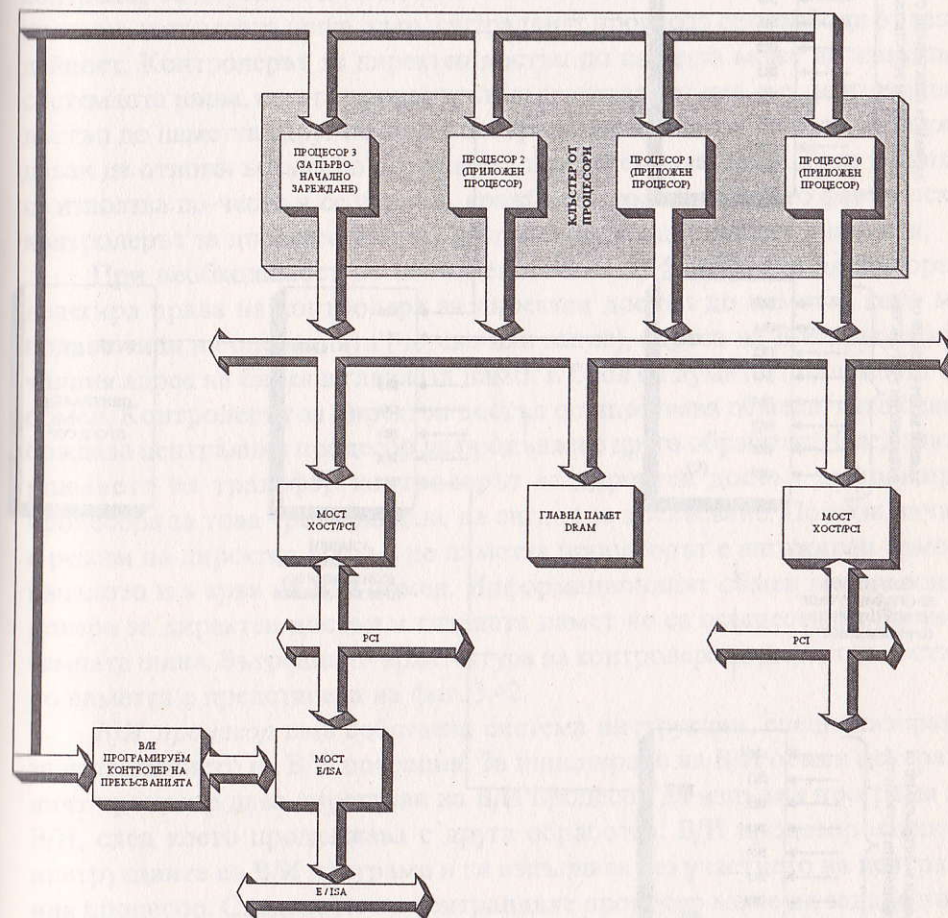
Обикновено процесорите разполагат с ограничен брой входове за прекъсване. За да се разшири броят на обслужваните устройства, се използват външни арбитри на прекъсванията. Те приемат заявките за прекъсване от свързаните периферни устройства, определят заявката с най-висок приоритет и информират процесора чрез активиране на сигнала за заявка за прекъсване INTR. След като процесорът потвърди прекъсването, контролерът за прекъсване подава по шината за данни съответния вектор на прекъсване. Процесорът директно комуникира с В/И модул за получаване или изпращане на дума от данни. Вътрешната архитектура на програмируемия В/И контролер за прекъсване е представен на фиг. 3.39. Той може да приеме едновременно заявки за прекъсване от осем периферни устройства, като наличието на заявки се отбелязва чрез установяването в 1 на съответния бит в регистъра на заявките. Контролерът съдържа и регистър с маските на прекъсванията, който използва за маскиране (забраняване) на заявките от определени периферни устройства. След регистъра на маските следва приоритетната логика, която служи за определяне на най-високоприоритетната заявка. В/И контролер за прекъсване е програмируем и следователно маските и приоритетната дисциплина на обслужване на заявките се задават от централния процесор посредством зареждане на управляваща дума в съответния регистър на контролера. Възможните режими на обслужване на прекъсванията са:

- *С фиксиран приоритет* – на всяко периферно устройство се задава приоритет от 0 (IR0) до 7 (IR7).
- *Ротация на приоритетите* – за приложения, в които обслужваните периферни устройства имат еднакъв приоритет, приоритетите на устройствата динамично се променят, като се прилага правилото последното обслужено устройство получава най-нисък приоритет.
- *Маскиране* – процесорът забранява обслужването на заявки за прекъсване от определени от него периферни устройства.

На фиг. 3.40 е показано използването на В/И контролер за прекъсване в типична сървърна платформа.

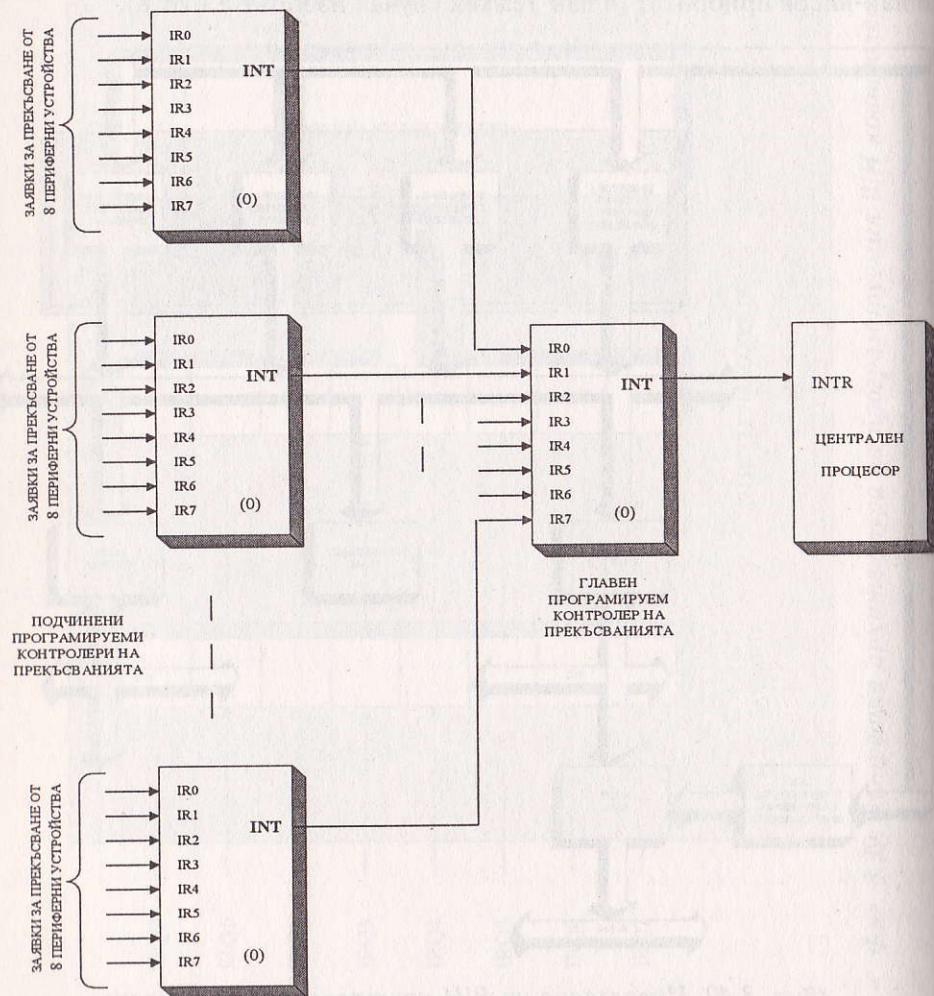


Фиг. 3.39. Вътрешна архитектура на програмируемия В/И контролер за прекъсване



Фиг. 3.40. Използване на В/И контролер за прекъсване в типична свързана платформа

Изходът за заявка за прекъсване INT на всеки подчинен контролер е свързан към съответния вход за заявка за прекъсване на главния контролер (master) от втория етаж на каскадата IR0 до IR7. Главният контролер прави селекция на второ ниво на най-високоприоритетната заявка от избраните 8 заявки от подчинените контролери и изпраща към централния процесор избраната заявка за прекъсване (в най-тежкия случай 1 от 64), съпроводена от съответния вектор на прекъсване след потвърждаване на прекъсването от централния процесор.



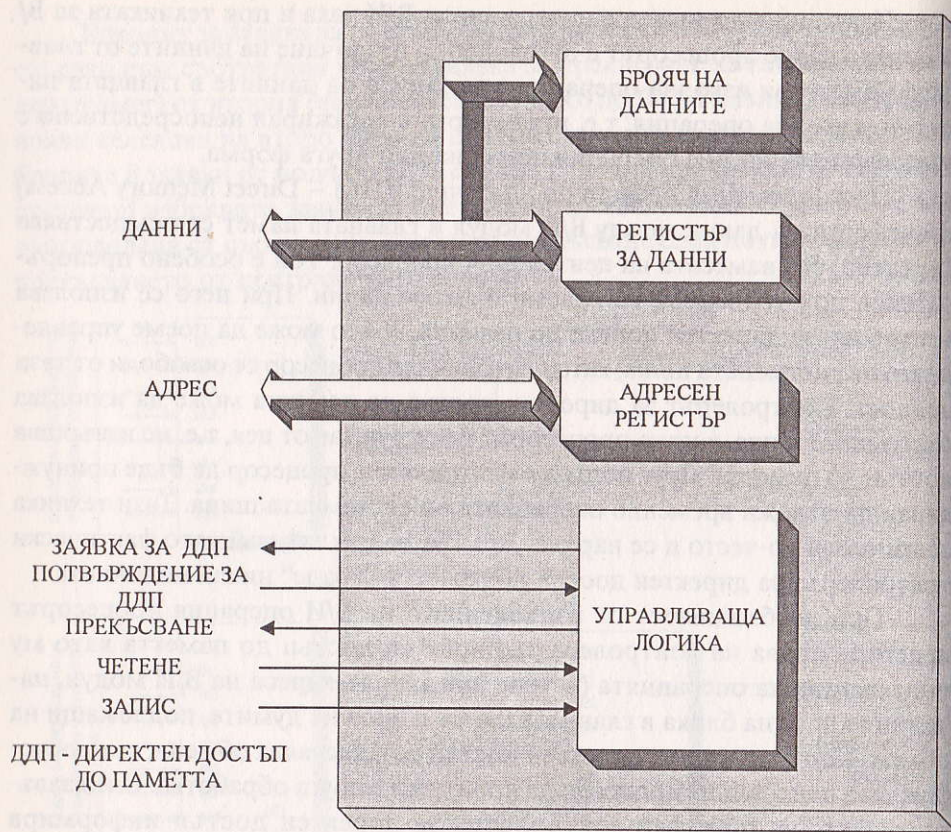
Фиг. 3.41. Каскадно свързване на програмируеми В/И контролери на прекъсванията

Както при техниката с програмиран В/И така и при техниката за В/И с прекъсване процесорът е отговорен за извличане на данните от главната памет при изходна операция и за запис на данните в главната памет при входна операция, т.е. процесорът е ангажиран непосредствено с изпълнението на В/И операция под една или друга форма.

При директния достъп до паметта (DMA – Direct Memory Access) трансферът на данни между В/И модул и главната памет се осъществява директно, без намесата на централния процесор. Той е особено препоръчителен при трансфера на големи блокове данни. При него се използва контролер за директен достъп до паметта, който може да поеме управлението на системната шина, като централният процесор се освободи от тази дейност. Контролерът за директен достъп до паметта може да използва системната шина, когато процесорът не се нуждае от нея, т.е. не извършва достъп до паметта. Друг подход е централният процесор да бъде принуждаван да отложи временно операцията по системната шина. Тази техника се използва по-често и се нарича „кражба на цикли“, тъй като фактически контролерът за директен достъп до паметта „краде“ цикли по шината.

При необходимост от изпълнението на В/И операция процесорът делегира права на контролера за директен достъп до паметта като му подава типа на операцията (четене или запис), адреса на В/И модул, началния адрес на блока в главната памет и броя на думите, подлежащи на обмен. Контролерът за директен достъп осъществява обмена, като освобождава централния процесор да продължи с друга обработка. След завършването на трансфер контролерът за директен достъп информира процесора за това чрез подаване на сигнал за прекъсване. По този начин в режим на директен достъп до паметта процесорът е ангажиран само в началото и в края на В/И обмен. Информационният обмен между контролера за директен достъп и главната памет не се осъществява по системната шина. Вътрешната архитектура на контролера за директен достъп до паметта е представена на фиг. 3.42.

В/И процесор има собствена система инструкции, специализирана за изпълнението на В/И операции. За инициране на В/И обмен централният процесор дава директиви на В/И процесор да изпълни програма за В/И, след което продължава с друга обработка. В/И процесор извлича инструкциите на В/И програма и ги изпълнява без участието на централния процесор. Следователно централният процесор може да зададе последователност от В/И операции за изпълнение на В/И процесор, след изпълнението на които той ще бъде информиран чрез получаване на сигнал за прекъсване.



Фиг. 3.42. Вътрешна архитектура на контролер за директен достъп до паметта

В/И канал представлява разширение на концепцията за директен достъп до паметта. При компютърни системи с В/И канал централният процесор изобщо не е ангажиран с изпълнението на В/И, а само с неговото инициране. Изпълняваната програма от В/И канал определя периферните устройства и областите от главната памет, между които ще се реализира обменът, приоритета на обслужване и спецификата на обработката на евентуално възникнали грешки. В/И канал следва тези инструкции и управлява В/И обмен.

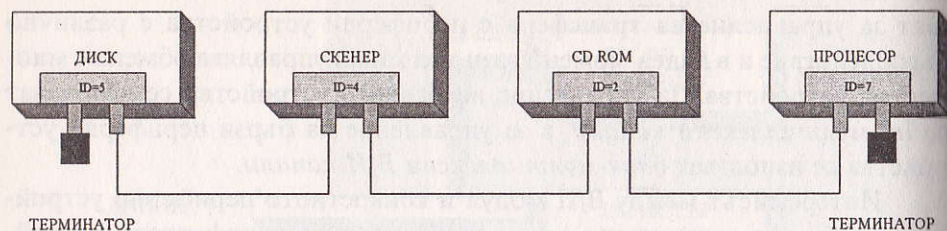
В/И канали могат да бъдат селекторни и мултиплексни. Селекторните В/И канали се използват за управление на трансфера с високоскоростни периферни устройства и в даден момент този тип канал управлява

обмена само с едно устройство. Мултиплексните В/И канали се използват за управление на трансфера с периферни устройства с различно бързодействие и в даден момент този тип канал управлява обмена с множество устройства. За управление на бавните устройства се използват байт-мултиплексни канали, а за управление на бързи периферни устройства се използват блок-мултиплексни В/И канали.

Интерфейсът между В/И модул и конкретното периферно устройство трябва да се адаптира към спецификата на периферното устройство. Една от основните характеристики е дали интерфейсът е паралелен или сериен (последователен). При паралелния интерфейс съществуват множество линии за данни, по които едновременно се предават битовете. При серийния интерфейс съществува само една линия за данни и по нея последователно във времето се подават битовете. Паралелният интерфейс се използва при по-високоскоростните периферни устройства като дисковете. Сериен интерфейс се използва за по-бавни устройства като клавиатурата и принтерите. Свързването между В/И модул и периферните устройства в компютърната система може да бъде точка-с-точка (point-to-point) или с множество точки. Интерфейсът от тип точка-с-точка се използва за монополизирана линия между В/И модул и периферното устройство. При малките компютърни системи (персонални компютри и работни станции) този тип интерфейс се използва за клавиатурата, принтера и модема. Външните интерфейси с многоточково свързване се използват за външни масови паметни (дискови устройства) и устройства за мултимедия (CD-ROM, видео, аудио), като на практика представляват външни шини.

Интерфейсът *SCSI (Small Computer System Interface)* се използва при персонални компютри и работни станции. SCSI е стандартен интерфейс за устройства CD-ROM, аудио оборудване и външна масова дискова памет. Той е паралелен интерфейс с 8, 16 или 32 линии за данни. Конфигурацията на интерфейса в общия случай се нарича шина, въпреки че устройствата са свързани в последователност от тип „маргаритка“. Всяко устройство, свързано към SCSI, има два конектора: един за вход и един за изход (фиг. 3.43).

Периферните устройства са свързани последователно едно с друго, като единият край на последователността е свързана с хоста. Всички периферни устройства функционират независимо и могат да обменят информация както с хоста, така и помежду си. Трансферът на данните се осъществява посредством пакети със съобщения. Интерфейсът SCSI поддържа асинхронен трансфер на данни, при който за всеки обменен



Фиг. 3.43. Примерна конфигурация на интерфейса SCSI

байт е необходимо да се използва протокол на ръкостискане (сигнали заявка REQ и потвърждение ACK). Синхронен режим на трансфер също се поддържа, но е необходимо изрично да бъде договорен. Предназначението на обменните съобщения е да управляват интерфейса. Форматът на съобщенията може да бъде еднобайтов, двубайтов или да съдържа три или повече байта. Тези съобщения носят информация за завършване на изпълнението на команда, предупреждение за прекъсване на връзката, установена грешка, договаряне на синхронен трансфер. Системата команди, използвана при протокола SCSI, включва множество команди, специфични за дадено устройство и освен това множество команди за всички устройства.

3.7. ШИННА АРХИТЕКТУРА

Шината представлява среда за предаване на сигнали между множество устройства, свързани към нея. В даден момент само едно устройство може да предава сигнали по шината, а едно или повече устройства могат да приемат тези сигнали. Шината включва множество линии, по всяка от които се предава двоичен сигнал. Шината, която свързва основните компоненти на компютърната система, се нарича *системна шина*. Съвременните системни шини съдържат около 100 линии, които в общия случай могат да бъдат разделени на три функционални групи: адресна шина (група адресни линии), шина за данни (група линии за данни) и управляващи линии. Броят на линиите за данни определят ширината на шината за данни и респективно броя на битовете, които могат да бъдат предавани едновременно. Следователно ширината на шината за данни определя в значителна степен пропускателната способност на шината. По адресната шина се индицира адресът на източника или получателя на

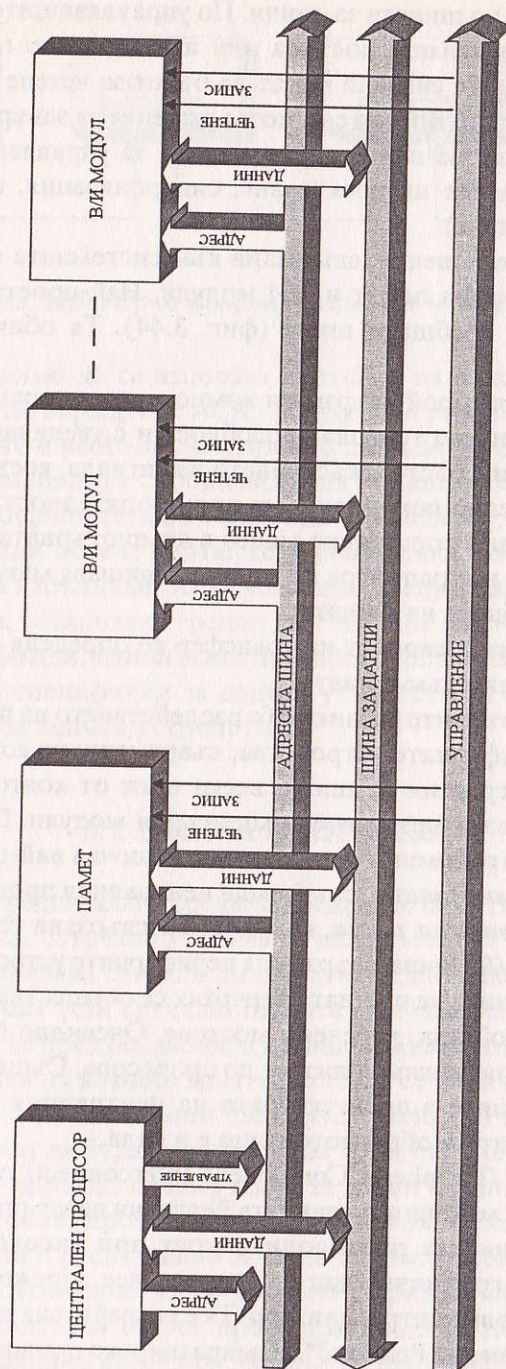
предаваните данни по шината за данни. По управляващите линии се предават сигнали, определящи достъпа или използването на предаваните данни. Управляващите сигнали могат да бъдат за четене в или запис от паметта, В/И запис или В/И четене, потвърждение за завършен трансфер, заявка за управление на шината, разрешение за управление на шината, заявка и потвърждение на прекъсване, синхронизация, инициализация (начално установяване).

Основните компоненти, свързвани към системната шина, са: централен процесор, главна памет и В/И модули. Най-простата конфигурация на свързване е общата шина (фиг. 3.44). Тя обаче има редица недостатъци:

- Допустимият брой свързвани компоненти е ограничен, тъй като шината има ограничена товарна способност и с увеличаване на дължината на общата шина расте закъснението на сигнала, което от своя страна води до значително понижаване на производителността;
- Общата шина става тясно място в компютърната система, когато интензивността на трансфера на данни превишава максималната пропускателна способност на шината;
- Максималната скорост на трансфер се определя от най-бавното устройство, свързано към шината.

Поради драстичните разлики в бързодействието на процесора, главната памет и периферните устройства, съвременните компютърни системи използват йерархия от шини, всеки етаж от която е съобразен с бързодействието на свързаните компютърни модули. Шинната йерархия на съвременните компютърни системи включва най-общо три етажа: *процесорна (локална) шина*, свързваща централния процесор с външната кеш памет, *системна шина*, към която е свързана главната памет и *разширителна (В/И) шина*, свързваща периферните устройства. Връзката между отделните нива на шинната йерархия се осъществява посредством буфериращи устройства, наречени мостове. Очевидно бързите устройства се намират в по-голяма близост до процесора. Съществено предимство е, че промените в архитектурата на централния процесор не се отразяват на шината и обратното също е в сила.

Шината PCI (Peripheral Component Interconnect) служи като междинна шина (етаж мецанин) в шинната йерархия на компютърната система и осигурява висока производителност при високоскоростни В/И подсистеми като графични адаптери за дисплеи, мрежови интерфейсни контролери, дискови контролери и др. Тя е разработена от фирмата Intel® за системи, базирани на Pentium™. Намира широко приложение при съвре-



Фиг. 3.44. Компютърна система с обща шина

менните персонални компютри, работни станции и сървъри. Шината PCI може да поддържа както еднопроцесорни, така и мултипроцесорни конфигурации. Тя е синхронна шина с централизиран арбитраж.

На фиг. 3.45 е представено използването на шината PCI в еднопроцесорна система, а на фиг. 3.46 – за мултипроцесорна конфигурация в сървърна платформа.

Връзката между високоскоростната процесорна шина и шината PCI при десктоп системи се осигурява от електронен мост, комбиниран с контролер на главната DRAM памет. Електронният мост изпълнява ролята на буфер, осигуряващ функционирането на двете шини, процесорната шина и шината PCI, с различна скорост. В мултипроцесорните конфигурации към системната шина се свързват една или повече шини PCI посредством електронни мостове. Системната шина поддържа само процесора с външната кеш памет, главната памет и PCI мостовете.

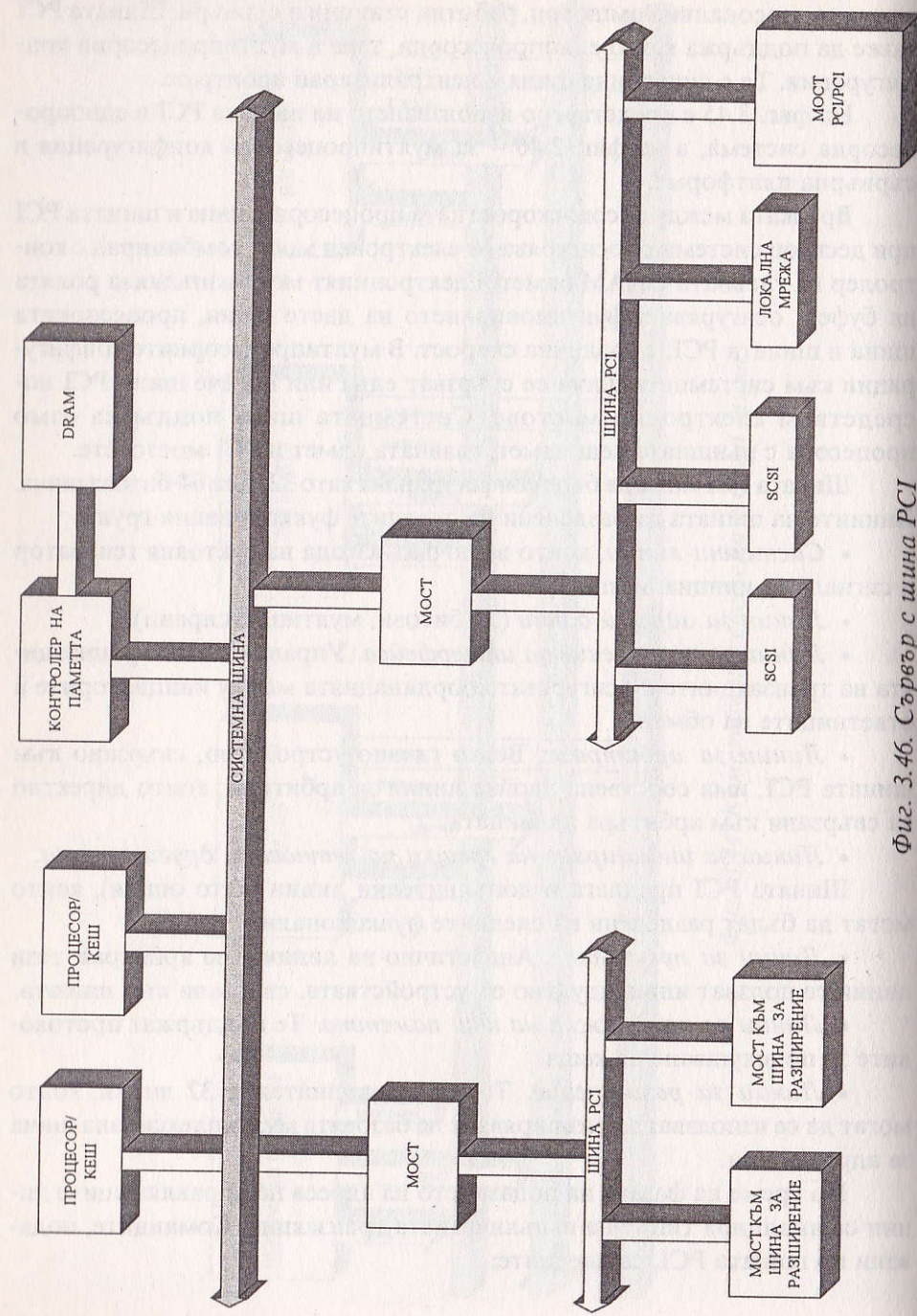
Шината PCI може да бъде конфигурирана като 32- или 64-битова шина. Линиите на шината са разделени на следните функционални групи:

- *Системни линии*, които включват изхода на тактовия генератор и сигнала за инициализация (reset);
- *Линии за адрес и данни* (32-битови, мултиплексирани);
- *Линии за управление на интерфейса*. Управяват синхронизацията на транзакциите и осигуряват координацията между инициаторите и ответниците на обмена;
- *Линии за арбитраж*. Всяко главно устройство, свързано към шината PCI, има собствена двойка линии за арбитраж, които директно са свързани към арбитража на шината;
- *Линии за индициране на грешка по четност и други грешки*.

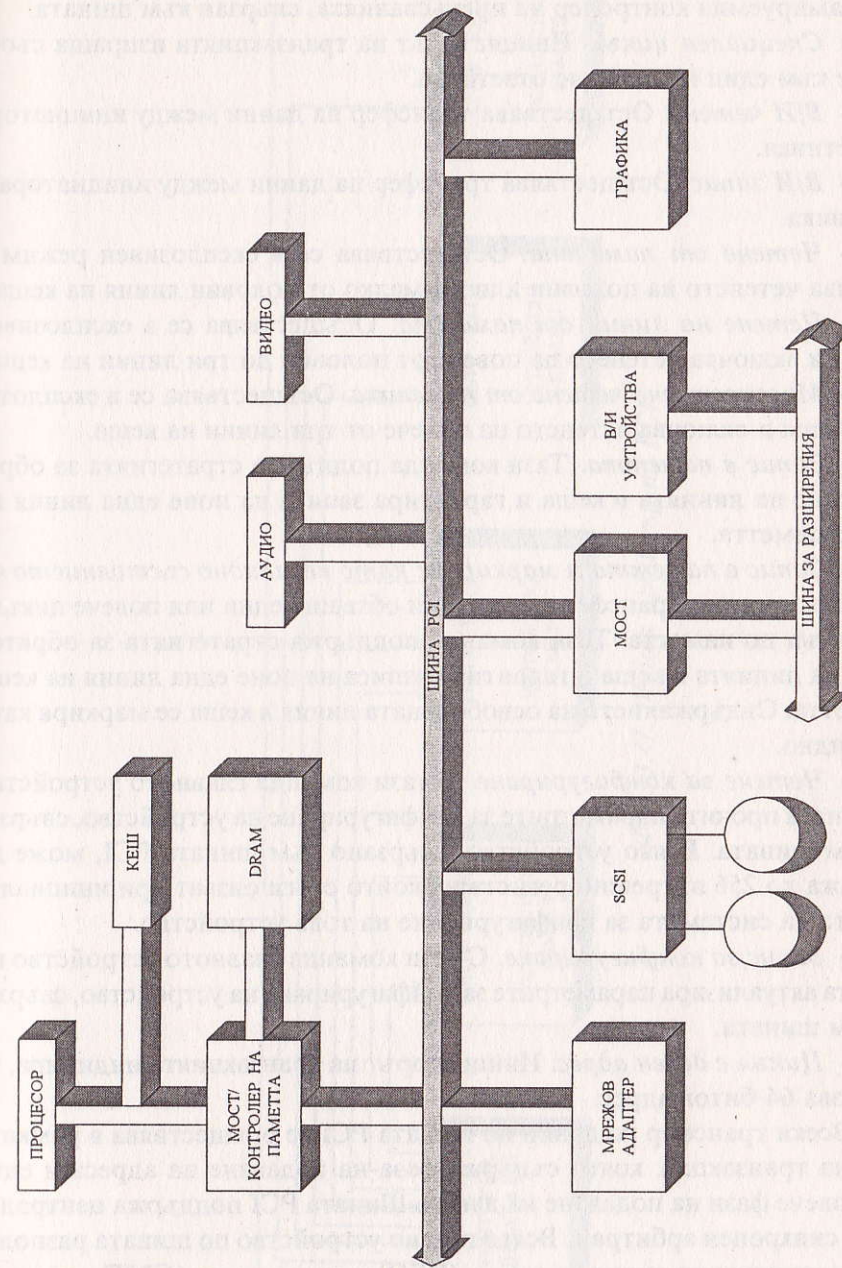
Шината PCI предлага и допълнителни линии (като опция), които могат да бъдат разделени на следните функционални групи:

- *Линии за прекъсване*. Аналогично на линиите за арбитраж тези линии се ползват индивидуално от устройствата, свързани към шината.
- *Линии за поддръжка на кеш паметта*. Те поддържат протоколите за подслушване на кеша.
- *Линии за разширение*. Това са допълнителни 32 линии, които могат да се използват за разширяване на базовата мултиплексирана шина за адрес/данни.

По време на фазата на подаването на адреса по управляващите линии се индицира типът на изпълняваната транзакция. Командите, подавани по шината PCI, са следните:



Фиг. 3.46. Сървър с шина PCI



Фиг. 3.45. Desktop система с шина PCI

- *Потвърждаване на прекъсването.* Това е команда за четене за програмируемия контролер на прекъсванията, свързан към шината.

- *Специален цикъл.* Инициаторът на транзакцията изпраща съобщение към един или повече ответника.

- *В/И четене.* Осъществява трансфер на данни между инициатора и ответника.

- *В/И запис.* Осъществява трансфер на данни между инициатора и ответника.

- *Четене от паметта.* Осъществява се в експлозивен режим и включва четенето на половин или по-малко от половин линия на кеша.

- *Четене на линия от паметта.* Осъществява се в експлозивен режим и включва четенето на повече от половин до три линии на кеша.

- *Множествено четене от паметта.* Осъществява се в експлозивен режим и включва четенето на повече от три линии на кеша.

- *Запис в паметта.* Тази команда поддържа стратегията за обратен запис на линията в кеша и гарантира записа на поне една линия на кеша в паметта.

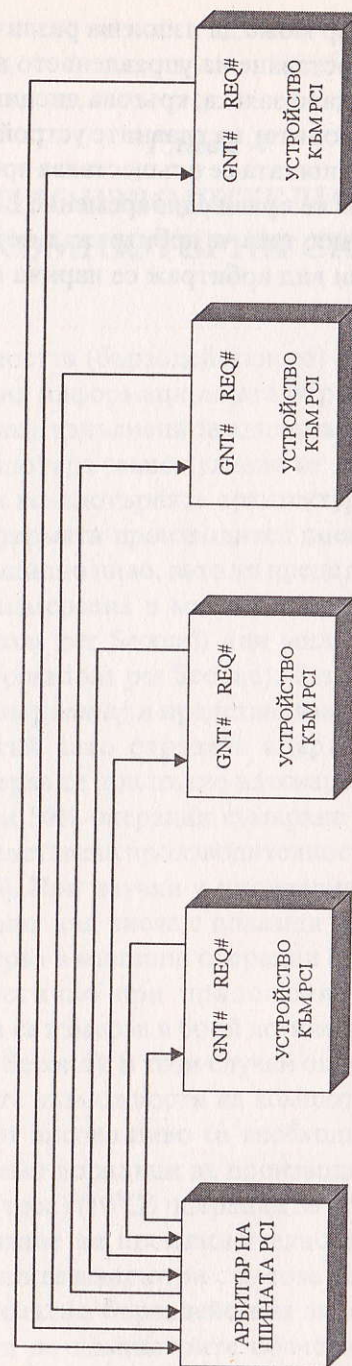
- *Запис в паметта и маркиране като невалидно състоянието на линията в кеша.* Трансферът на данни обхваща един или повече цикъла на достъп до паметта. Тази команда поддържа стратегията за обратен запис на линията в кеша и гарантира записа на поне една линия на кеша в паметта. Съдържанието на освободената линия в кеша се маркира като невалидно.

- *Четене за конфигуриране.* С тази команда главното устройство по шината прочита параметрите за конфигуриране на устройство, свързано към шината. Всяко устройство, свързано към шината PCI, може да съдържа до 256 вътрешни регистъра, които се използват при инициализацията на системата за конфигуриране на това устройство.

- *Запис за конфигуриране.* С тази команда главното устройство по шината актуализира параметрите за конфигуриране на устройство, свързано към шината.

- *Цикъл с двоен адрес.* Инициаторът на транзакцията индицира, че използва 64-битов адрес.

Всеки трансфер на данни по шината PCI се осъществява в рамките на една транзакция, която съдържа фаза на подаване на адреса и една или повече фази на подаване на данни. Шината PCI поддържа централизиран синхронен арбитраж. Всяко главно устройство по шината разполага с индивидуални линии за заявка (REQ) и разрешение (GNT), които са свързани към централния арбитър (фиг. 3.47).



Фиг. 3.47. Шинен арбитър на PCI

Централният арбитър може да използва различни стратегии за вземане на решение за предоставяне на управлението на шината: на първо-то главно устройство, подало заявка, кръгова дисциплина на обслужване или схема с фиксиран приоритет на главните устройства. За изпълнение-то на всяка транзакция по шината се осъществява арбитраж. Арбитражът за следващата транзакция се прави едновременно с трансфера на данни-те при текущата транзакция, така че арбитражът не изисква допълнител-ни цикли по шината. Този вид арбитраж се нарича скрит арбитраж.

Глава 4

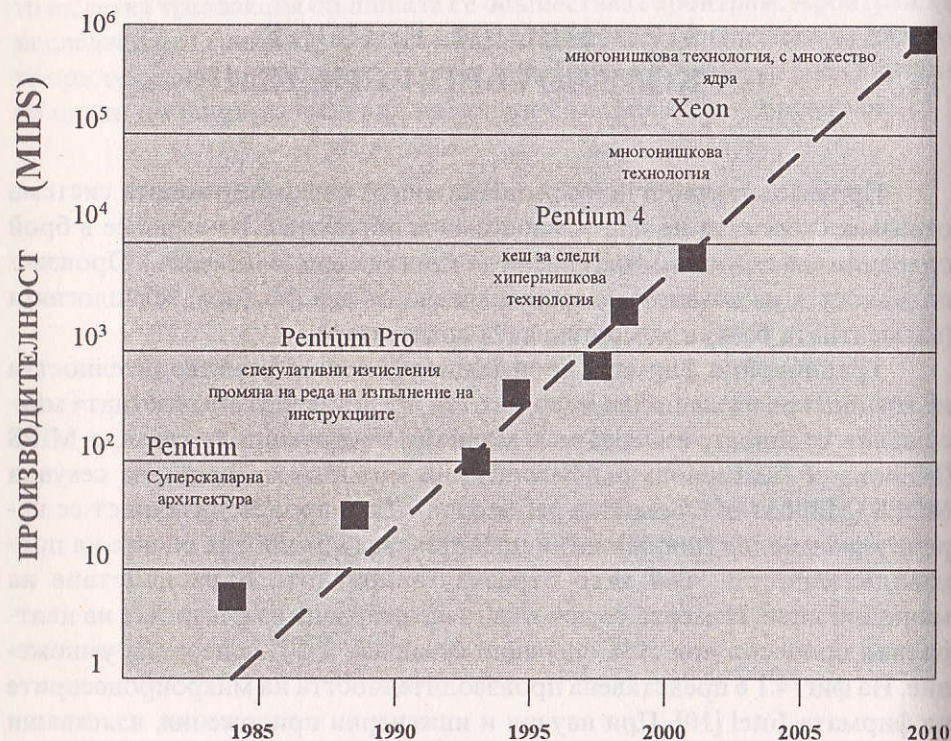
ПРОИЗВОДИТЕЛНОСТ НА КОМПЮТЪРНИ СИСТЕМИ

Производителността (бързодействието) на компютърната система отразява скоростта на информационната обработка. Измерва се в брой операции (инструкции), изпълнени за единица време (секунда). Произво-дителността на компютъра зависи главно от два фактора: технологията (елементната база) и компютърната архитектура.

Традиционно фирмата производител оценява производителността на компютъра на машинно ниво, като тя представлява абсолютната мак-симална стойност, измервана в милиони инструкции за секунда MIPS (Millions of Instructions per Second) или милиони операции за секунда MOPS (Millions of Operations per Second). Тази производителност се на-рича *максимална (апаратна)* и представлява абсолютна оценка на про-изводителността, тъй като отразява апаратното бързодействие на машинно ниво. Измерва се при пълно натоварване на ресурсите на цент-ралния процесор при 50% операции сумиране и 50% операции умноже-ние. На фиг. 4.1 е представена производителността на микропроцесорите на фирмата Intel [10]. При научни и инженерни приложения, изискващи голям обем изчисления над числа с плаваща точка, абсолютната произ-водителност се измерва в милиони операции с плаваща запетая за секун-да MFLOPS. Аналогично при приложения за изкуствен интелект производителността се измерва в брой логически изводи за секунда LIPS (Logic Inferences per Second). В този случай оценките не отразяват непосредствено апаратните възможности на компютъра, тъй като за реализа-цията на действия от високо ниво са необходими множество машинни операции. За по-големи порядъци на производителността се използва G (гига) (10^3 M) или T (тера) (10^3 G) операции за секунда.

Точното оценяване на производителността е особено важно при високопроизводителните компютри с висока цена, при които потребите-лите изискват максимално бързодействие за приложните си програми. Производителността на компютрите обаче зависи от много взаимно

свързани фактори като конкретното приложение, обема на изчислителната работа, избрания алгоритъм, човешките усилия за оптимизиране на програмата, способността на компилатора за оптимизация, ефективността на операционната система, компютърната архитектура и др.



Фиг. 4.1. Производителност на микропроцесорите на фирмата Intel

Съществуват множество подходи за оценяване на производителността, стремежът при които е да се удовлетворят изискванията на всички заинтересовани страни от измерването на производителността (проектанти и потребители). За потребителите е особено важно да знаят по какъв начин системната производителност се влияе от архитектурата, компилатора, операционната система, периферията и др. По този начин те могат да се възползват от спецификата на архитектурните белези и да изберат оптималния стил на програмиране. На проектантите тази оценка ще помогне при оптимизацията на апаратните средства и програмното осигуряване. За реални приложения се определя т.нар. *средна производителност*

на компютъра, която е значително по-ниска от максималната производителност и в общия случай представлява около 40% от максималната производителност.

Системната производителност на компютрите се влияе в значителна степен от приложната област. Затова се дефинира понятието *3D компютър*, при който производителността се оценява като точка в тримерното пространство по три направления: бързодействие на централния процесор, латентност на системата за прекъсване и ефективност на В/И система. Бързодействието на централния процесор е от изключителна важност за инженерни и научни приложения с голям обем изчисления над числа с плаваща точка. Латентността на системата за прекъсване е критична за системи за управление в реално време, тъй като тя определя времето за отговор на компютъра за анализ на контролираните параметри на управлявания обект и задействане на адекватна програма за осъществяването на обратната връзка за въздействие върху управлявания обект. Бързодействието на В/И система е от съществено значение за икономически приложения, които изобилстват с интензивни обръщания към диска за използване на бази данни и др.

Основен метод за определяне на компютърната производителност е изпълнението на еталонни програми (benchmarks). Той осигурява получаването както на качествени, така и на количествени оценки на производителността. *Качествената оценка* засяга характеристиките на разработвания компютър и откриването на недостатъци и евентуални тесни места. Тя се прави още в етапа на проектиране въз основа на симулации с консервативни допускания. Това е т.нар. фаза на „книжния тигър“, в която проектантите обещаваат висока производителност на базата на направените симулации. Всеки нов компютър обаче трябва да бъде разглеждан скептично до окончателното му предаване на клиентите в масови количества. Следва фазата на „белия слон“, при която компютърът се предлага на пазара, но приемането му от клиентите все още не е сигурно. Обикновено в тази фаза за компютъра не е създадено оптимално програмно осигуряване, което трябва да се компенсира от потребителите.

Най-важната *качествена характеристика*, която трябва да бъде определена за даден компютър, е съвместимостта. Обхватът на съвместимостта може да определи програмното осигуряване и периферните устройства, които могат да бъдат използвани, както и взаимстването на програми и файлове. От една страна съвместимостта е желателна, тъй като улеснява прехода към нови апаратни средства. В този случай иновациите в апаратните средства се реализират като надстройка над съще-

ствуващото процесорно ядро. От друга страна съвместимостта е фактор, който в значителна степен може да спъне тяхното развитие. Естествено съществуването на съвместимост при езиците за програмиране, компилаторите и операционните системи е крайно необходимо, тъй като в противен случай потребителят трябва да разработва собствено приложно програмно осигуряване, което води до недопустими разходи на време и труд за постигането на реални резултати.

На основата на качествената оценка на производителността могат да бъдат предпочетени определени класове компютри в зависимост от архитектурата им. Едва след окончателното определяне на качествените характеристики може да се пристъпи към *количествени оценки*. Те включват цената, продуктивността, надеждността и бързодействието. *Цената* включва не само средствата за закупуването на компютъра, но и разходите за инсталирането, поддръжката, обучението на обслужващия персонал и разработката на приложните програми. *Продуктивността* в общия случай се дефинира като отношение цена/производителност, но при нейната оценка задължително се включват диалоговите средства, високоскоростните графични средства, наличните мощни редактори и настрояващи програми, както и средствата за документиране на резултатите.

Високата *надеждност* е от особено значение при програми, изискващи голям обем изчислителна работа и съответно голямо машинно време. Добър показател за надеждността на компютъра е *средното време за отказ*.

Накрая се прави количествена оценка на бързодействието на компютъра. Тя помага да се прецени полезността на компютъра за дадена фирма или организация, стимулира развитието на системното и приложното програмно осигуряване, като дава възможност за ефективно използване на съществуващите архитектури.

Един от най-широко използваните методи за определяне на компютърната производителност е изпълнението на еталонни програми. Получената оценка може да бъде *абсолютна*, когато се оценява ефективността на компютъра за дадено приложение и *относителна*, когато се прави сравнителна оценка с възможностите на други компютри. Проектантите използват тестването с еталонни програми, за да улеснят избора си между алтернативни архитектурни решения и технически реализации. Клиентите разчитат на еталонните програми при закупуването на нов компютър, а програмистите – за избора на оптимален стил за програмиране.

Използването на неудачни еталонни програми обаче може да подведе както проектантите така и клиентите и потребителите. Ето защо оценката на производителността трябва да включва:

- Прецизна характеристика на изчислителния товар;
- Първоначално тестване с еталонни програми;
- Тестване с програми, апроксимиращи ежедневното натоварване

на компютъра.

Изчислителният товар представлява смес от различни типове програми, изпълнявани от един компютър. Основните характеристики на изчислителния товар са изискванията за вход/изход, обемът и видът на изчислителната работа, възможната степен на векторизация.

Поради относително простата си архитектура по-малките компютри могат да бъдат оценявани с прости еталонни програми (от вида на Whetstones). Производителността на тези компютри не се променя значително при различните приложения. Мощните компютри със значителни архитектурни усъвършенствания и разширения са много по-чувствителни към характеристиките на конкретното приложение, тъй като производителността им зависи до голяма степен от степента на използване на архитектурните усъвършенствания на компютъра. С елементарните еталонни програми се определя максималната (апаратната) производителност.

Във всички случаи трябва да се има предвид, че програмите са приспособени в по-голяма или по-малка степен към конкретния компютър, на който ще се изпълняват. Дори когато е използван език за програмиране от високо ниво, изборът на математически модел, числен метод, начинът на трансформиране на физическия проблем в компютърна програма определят специфичната реализация на даден проблем. Основният проблем тук е до каква степен трябва да се модифицира еталонната програма при оценяването на системата. Друг важен проблем е да се определи точно кои компоненти се тестват от дадена еталонна програма, въпреки че е желателно да се оценява цялата система – апаратните средства, системното и приложното програмно осигуряване.

В много случаи е полезно да се направи относителна оценка на работата на машинно ниво спрямо работата на високо ниво – например да се определи средният брой изпълнени инструкции, необходим за всяка операция с плаваща точка или за всеки логически извод. Тези относителни оценки дават представа за ефективността на компютърната архитектура, в частност на компилатора, за дадено приложение. Те са полезни, тъй като дават възможност да се пренебрегнат технологичните особености на компютъра.

При оценяването на производителността на компютрите с успех се прилага *йерархичен подход*, при който информацията, получена на даден етап, подпомага анализа на следващия етап. От особено значение е коректното формиране на работния изчислителен товар.

На най-ниското ниво на йерархията се изпълняват програми, които тестват базовите операции в компютъра – сумиране, умножение и др. Програмите трябва да бъдат достатъчно прости, за да може да се направи изчерпателен анализ. От измерването на тези операции може да се определи времето за запълване на инструкционните конвейери, ефектът от използването на кеша, времето за достъп до паметта, влиянието на конфликтите при обръщането към един и същ модул на паметта, времето за достъп до дисковите памети и др. Програмите, тестващи базовите операции в компютъра, дават възможност да се определи неговата максимална (апаратна) производителност.

На второто ниво в йерархията се използват колекции от програмни ядра, които представляват части от реални програми или са изкуствено синтезирани. Широко приложение са получили следните колекции програмни ядра:

- *Whetstones*. Представляват синтезирани еталонни програми за основна аритметика. Използват се за оценка на производителността на компютри с ниско и средно бързодействие.

- *Dhrystones*. Представляват синтезирани еталонни програми за оценка на операциите в системните програми. Използват се за оценка на производителността при нечислени приложения при малки и средни компютри.

- *Linpack*. Представлява програмна библиотека за линейна алгебра. Използва се за оценка на векторната и скаларната производителност при обработката на числа с плаваща точка. С колекцията програмни ядра Linpack се оценява производителността на широка гама компютри – от персоналните компютри до мощни компютърни платформи. Тази колекция съдържа програма за декомпозиране на матрицата, представяща система линейни уравнения и програма за решаване на системата линейни уравнения на основата на декомпозираната матрица от първата програма. Основните операции тук са умножение на вектор със скалар и сумиране на вектори.

- *Цикли на Livermore*. Това е колекция подпрограми, извлечени от най-често изпълняваните приложни програми в националната лаборатория „Lawrence Livermore“ в САЩ. Тя съдържа апаратно определени подпрограми като подпрограми за вход/изход, управление на паметта или

подпрограми за графика. С тази колекция се оценява скаларната и векторната производителност при обработката на данни с плаваща точка. Тя се използва за оценка на производителността на суперкомпютрите.

Понастоящем за различните приложни области са разработени широка гама еталонни програми като например за приложения с мултимедия, компютърна графика, предпечатна подготовка и т.н.

В общия случай на основата на програмни ядра се оценява само производителността на централния процесор. Много рядко с тях се оценява производителността на интегрираната система, тъй като поради изключително простата си структура те могат да формират погрешна представа за системната производителност при цялостния изчислителен товар. Друг недостатък при използването на такива относително прости програми е, че статистиките при индивидуалните оценки на производителността варират в широки граници и зависят до голяма степен от използвания компилатор.

При подхода с програмните ядра е необходимо да се извърши следното:

1. Съставя се изчерпателна характеристика на работния изчислителен товар, като се определят класове проблеми и техният относителен дял в цялостния товар.

2. Извършва се селекция на множеството програми от работния изчислителен товар с цел средно представяне на отделните приложения, като се включат програми от различните класове приложения и се определят съответните тегловни коефициенти за всяко програмно ядро в зависимост от неговото представяне в товара.

3. Определят се частите от програмите (ядрата), които консумират най-голямата част от времето за изпълнение. Това е критичната стъпка при формирането на програмните ядра.

4. Изчислява се бързодействието при изпълнението на програмните ядра, като се вземат предвид тегловните коефициенти.

На третото ниво при йерархичния подход за оценка на производителността на компютрите се използват базови подпрограми, извлечени от широк спектър приложения. Те тестват както централния процесор, така и входно-изходната система. Така формираната оценка обаче не отразява взаимодействието между подпрограмите в рамките на реалните програми. Ако реалните програми съдържат други подпрограми, които консумират значителна част от времето за изпълнението им, то базовите подпрограми не биха могли да създадат цялостна картина на системната производителност.

На това ниво от йерархията резултатите от изпълнението на еталонните програми вече представляват интерес за крайния потребител, тъй като те дават информация не само за производителността, а също така и за паралелните структури и други оптимизиращи фактори.

На четвъртото ниво в йерархията се използват съкратени версии на основните програми в изчислителния товар. С тяхна помощ се оценяват взаимодействията между базовите подпрограми. От друга страна те съдържат много малка част от предварителната и пост обработка, изпълнявана от цялостните версии на програмите и може да съдържат или да не съдържат взаимодействие между централния процесор и входно-изходната система.

В общия случай съкратените версии на програмите са преносими за разлика от цялостните версии. Недостатък на използването на съкратените версии на програмите е, че са облекчени изискванията за памет или вход-изход, поради което има опасност да се загуби оригиналната програмна структура. Например ако съкратената версия на дадена програма се изпълнява от компютър с ограничен обем на паметта, получената оценка за производителността ще се различава съществено от реалната оценка, получена при изпълнението на пълната версия на програмата.

На петото ниво от йерархията може да се получи най-пълна и точна картина на системната производителност при решаването на типични проблеми, но въпреки това съществуват и редица недостатъци. Поради лошата преносимост на пълните версии на програмите при оценяването на системната производителност съществува тенденция да се отразява стилът на програмиране и вложените усилия за адаптиране на програмата към дадена архитектура. Измерването на системната производителност въз основа на цели програми трябва да се извърши едва след пълна и изчерпателен анализ на резултатите от измерванията с опростени програми и точното дефиниране на изчислителната среда. Особено важно е на това ниво да се използват обработваните данни в пълна обем, както и цялостният код на програмата за дадено приложение. В противен случай получените резултати могат да бъдат подвеждащи.

Целта при най-високото, шестото ниво на йерархията, е да се изследват насоките при формирането на бъдещия изчислителен товар, както и новите подходи при създаването на алгоритми и архитектури. Следователно на това ниво използваните тестове трябва да отразяват алгоритмични техники, които са в процес на разработка и са перспективни. Естествено не е разумно да се отхвърлят обещаващи архитектурни проекти само поради силната взаимосвързаност между създадените про-

грами и съществуващите компютърни системи. При прогресивното развитие от едно поколение компютри към следващото, съвместимостта е важен фактор, но на определен етап от това развитие може да се постигне значително повишаване на производителността, като се обнови радикално съществуващото програмно осигуряване. Резултатите от тестването с такива перспективни програми е трудно да бъдат обобщени. В замяна на това те дават възможност да се оцени потенциалът на нови компютърни системи, за които могат да бъдат създадени нови програми и да бъде формиран нов изчислителен товар.

Следователно за сравнение на производителностите на компютрите и за класифицирането им според производителността засега се използват с успех еталонни програми.

Експериментални техники за оценка на производителността на компютърната система. Целта на проектанта на компютри е да създаде компютърна система, която ще успее на пазара. Това изисква оптимално използване на наличните хардуерни и софтуерни технологии. Основната цел е откриването и отстраняването на тесни места в проекта на компютъра. За тази цел проектантът на компютри разполага със следните алтернативи: изследване чрез аналитични модели като например веригите на Марков или да използва експериментални техники като например симулация по следи (trace-driven simulation), симулация по събития (event-driven simulation) или апаратен мониторинг на съществуващи прототипи.

За изследване на системната производителност последователно по всички йерархични нива от централния процесор до 3-D компютъра е необходимо увеличаване на информацията за поведението на цялостната система, при което неизбежно необходимостта от експерименти нараства. При използване на процесори с опростена структура се усложнява структурата и организацията на цялостната система. Централният процесор, паметта, В/И система, различни компоненти на операционната система и приложното програмно осигуряване влияят върху производителността на цялата система. Съществуват много потенциални причини за тесни места в една система. Необходимо е да се има предвид значителната сложност при трансформирането на програми на езици от високо ниво в сигнали, управляващи апаратните средства. Основен е проблемът за оптимизиране на съотношението между софтуера и хардуера в компютърната система. Възниква въпросът как да се направи изборът. Възможно ли е да се създаде високопроизводителна система от несложен хардуер и несложни компилатори? Дебатът RISC/CISC, воден през последните години, е

показателен за тази алтернатива. Има възможности за избор и алтернативи, които се използват при нужда от измервания и анализи. Особено е актуален проблемът за настройката на индивидуалните компоненти, разделянето на функционални подсистеми и определянето на ефектите, които се загубват при тестването на декомпозираната система.

Експерименталните подходи за оценка на качествените и количествените аспекти на производителността на компютърните системи включват изпълнение на програми както от реални, така и от виртуални (симулирани) апаратни средства. В случаите на симулиран хардуер, поведението на системата се формира на основата на получените статистики, отразяващи динамиката на изчислителния процес.

Симулацията (имитационното моделиране) представлява основна част от експерименталните изследвания на компютърните архитектури и заема важно място в откриването на тесните места и постигането на оптимална производителност. От друга страна голяма част от проблемите на новата компютърна архитектура не се проявяват, докато тя не бъде реализирана и не се проведат експерименти с реални апаратни средства. Освен това дори след конструирането на системата реалните слабости се проявяват само при внимателни измервания на реално работещата система. Всеки от използваните подходи води до различни компромиси между критериите цена и време, за да се осигури гъвкавост на експерименталната система и достоверност на получените данни.

Симулацията по следи дава възможност да се акумулират данни, без да се намалява ефективността на моделираната система. Тя също осигурява възможност лесно да се променят моделираните системни параметри и изискванията за получаваните данни. Методът може да се използва с лекота за получаване на данни при съществуващи продукти или експериментални прототипи. Този вид симулация обаче има два основни недостатъка: ниска скорост на натрупване на статистиките и неперфектността на модела. Фактът, че данните се натрупват няколко пъти по-бавно от бързодействието на изследваната система, е показателен, че не е възможно да се добие пълна представа за реалните събития. Въпреки това този вид симулация е добра първа стъпка за детайлното изследване на системата. По-нататък, когато се обединят и други експериментални техники, симулационният модел може да бъде потвърден, което пък осигурява механизма за даване на бърз отговор в коя посока да се оптимизира системата.

При наличие на реален хардуер могат да се използват три метода: софтуерен мониторинг, фърмуерен мониторинг и апаратен мониторинг.

Програмният мониторинг е приложим както при реални компютри, така и при експериментални прототипи. И в двата случая е полезен за проверка на адекватността на симулационния модел. Така сравняването на данните, получени при програмния мониторинг, със симулационните резултати могат да доведат до разработването на по-добра методология за симулация. Програмният мониторинг е лесен за реализация и често се улеснява от архитектурните характеристики като възможностите за трасиране на програмата или използването на програмни точки на прекъсване. Следователно той е най-лесно приложимият метод. Заедно с това улеснение обаче идва голямото предизвикателство за ефективното използване на данните, получени при програмния мониторинг. Основни недостатъци са ниската скорост на събирането на тези данни, както и липсата на гъвкавост (невъзможността да се променят параметрите на изследваната система) за разлика от симулационните методи.

Апаратният мониторинг както и програмния мониторинг може да се прилага при съществуващи машини и експериментални прототипи. И в двата случая цената е висока и необходимото време е голямо. Предимството е високата достоверност на информацията. Недостатъците са високата цена за конструирането на апаратурата и трудностите при промяната на измерваните параметри. Проблеми са както събирането на експерименталните данни, така и тяхната обработка и интерпретацията на получените резултати.

Фърмуерният мониторинг (измерване чрез изпълнение на микропрограми) заема средно място между програмния и апаратния мониторинг. Както при апаратния мониторинг устройството за мониторинг не влияе на достоверността на резултатите. От друга страна подобно на програмния мониторинг не е скъп и има възможност за лесна промяна.

В резюме може да се обобщи, че при изследването на компютърните архитектури е наложително използването на експериментални техники както в етапа на проектирането така и след създаването на реалната машина. Поради високата сложност на компютърните системи е необходимо да се провери първо фундаменталната концепция чрез екстензивна симулация, след което да се създаде и тества прототип на системата, при което се потвърждава рационалността и работоспособността на концепцията и се събира информация, полезна при следващата итерация от разработката на модела. Полезността на един метод (например симулацията) може да бъде подобрена като се съчетаят изводите, получени при използването на този метод с резултатите, получени при приложението на друг метод (например изпълнението на програмата от прототип).

С цел да се изследва взаимодействието и влиянието на различните фактори в компютърната система, е необходимо експерименталното изследване на компютърната производителност да се извърши на няколко йерархични нива, като се оползотворяват максимално предимствата на всяко ниво и се отчитат съответните ограничения. При това се обобщават резултатите от всички нива за формиране и доказване на общата тенденция. Този подход дава възможност за откриване на „тесни места“ в сложните компютърни системи, които възпрепятстват оптималното използване на наличната технология.

При сравнение на производителностите на два компютъра трябва да се избягват следните похвати: използването на малки по обем еталонни програми, които се побират изцяло в кеша, използването на ограничен брой типове данни, неизползването на В/И операции, надценяването на ролята на конкретни инструкции в програмните ядра, сравняването на резултатите от обработката на експерименталните данни, получени по различни методи като например сравняването на експерименталните данни от симулацията на едната компютърна система с тези от програмния мониторинг на другата система, използването на компилатор с много добри (лоши) оптимизиращи функции. Интерес представлява изследването на влиянието на различните видове високоскоростна памет в процесора (регистрови прозорци, множество регистрови файлове, стандартни регистрови файлове) върху производителността на компютъра при широк спектър компютърни архитектури с различни системи инструкции.

Глава 5

ПАРАЛЕЛНИ КОМПЮТЪРНИ СИСТЕМИ

5.1. ПАРАЛЕЛНИ КОМПЮТЪРНИ АРХИТЕКТУРИ

Съвременните паралелни компютърни архитектури могат да бъдат разделени на два големи класа в зависимост от приемствеността от фон Ноймановата архитектура:

- *Еволюционни паралелни архитектури*, непосредствено развили се от фон Ноймановата архитектура със съхраняване на формата на механизма за управление от инструкции, при които централизираният инструкционен поток се трансформира в децентрализирани инструкционни потоци, изпълнявани под управлението на разпределена операционна система. Тук спадат мултипроцесорите, мултикомпютрите, асоциативните процесори, систоличните процесори, процесорите с разпространение на фронта на вълната, векторните процесори, паралелните компютри с функционална реконфигурация.

- *Радикални паралелни архитектури* с качествено нови механизми за управление (от данни, от заявки, от ситуации). Тук се включват паралелните компютри с управление от потоци данни, редукционните архитектури, ориентирани към езиците за функционално програмиране и паралелните логически компютри.

Предложени са различни дефиниции за паралелни архитектури. Трудността в прецизното дефиниране на термина идва от проблема за изграждането на класификацията на паралелните архитектури, която да отговаря на следните изисквания:

- Поддържане и включване на елементите на класификацията на Флин, основаваща се на потоците от инструкции и данни;
- Обхващане на конвейерните, векторните процесори и други паралелни архитектури, които не могат да се приспособят към схемата на Флин.

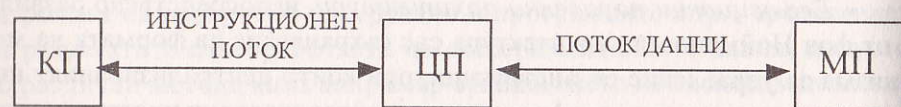
Основен критерий за общата класификация на компютърните архитектури може да бъде организацията на изчислителния процес. Той се изгражда като обработка на потоци от информация – данни и инструк-

ции. В зависимост от това дали тези потоци са единични или множествени Флин е направил една обща класификация на компютърните архитектури, според която съществуват следните четири архитектурни класове:

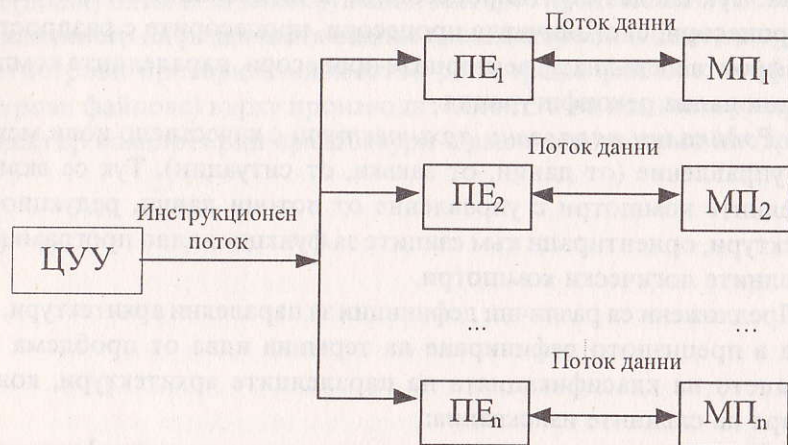
- Класът „Единичен поток Инструкции, Единичен поток Данни“ (ЕИЕД, *SISD*) обхваща компютри с фон Нойманова архитектура, т.е. компютрите с последователна организация на изчисленията.

- В класа „Единичен поток Инструкции, Множествени Потоци данни“ (ЕИМД, *SIMD*) се изпълнява единичен поток от инструкции над множество потоци от данни, които се обработват в множество процесори. Към този клас спадат матричните процесори.

- Към класа „Множествени потоци Инструкции, Множествени потоци Данни“ (МИМД, *MIMD*) спадат мултипроцесорите и мултикомпютрите.

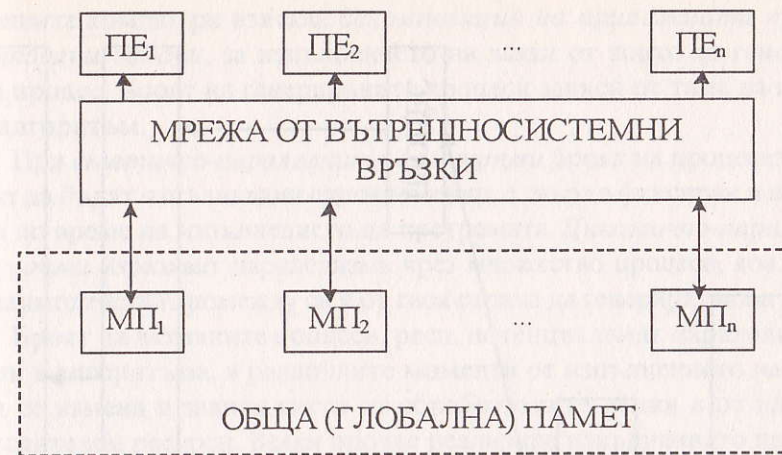


(а) клас „Единичен поток инструкции, единичен поток данни“ (*SISD*)

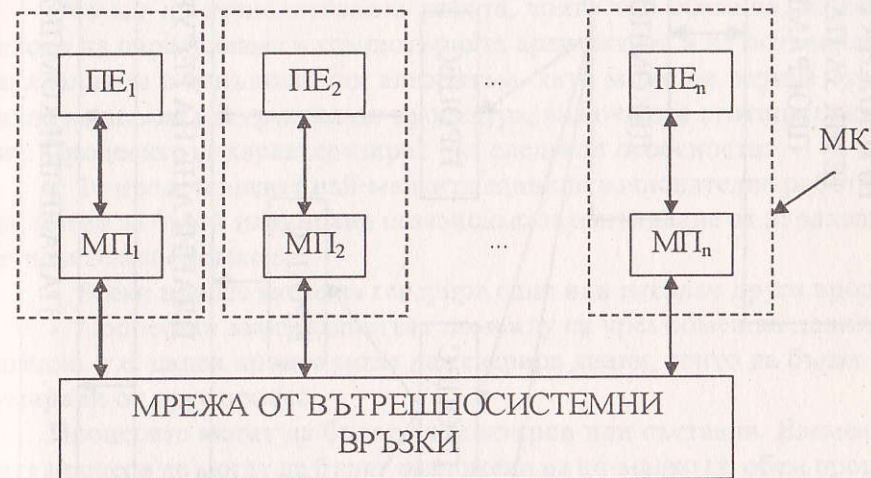


(б) клас „Единичен поток инструкции, множествени потоци данни“ (*SIMD*)

Легенда:
 ЦП – Централен процесор
 КП – Комуникационен процесор
 МП – Модул памет
 ПЕ – Процесорен елемент
 ЦУУ – Централно управляващо устройство



(в) клас „Множествени потоци инструкции, множествени потоци данни“ (*MIMD*) с обща памет (МУЛТИПРОЦЕСОРИ)

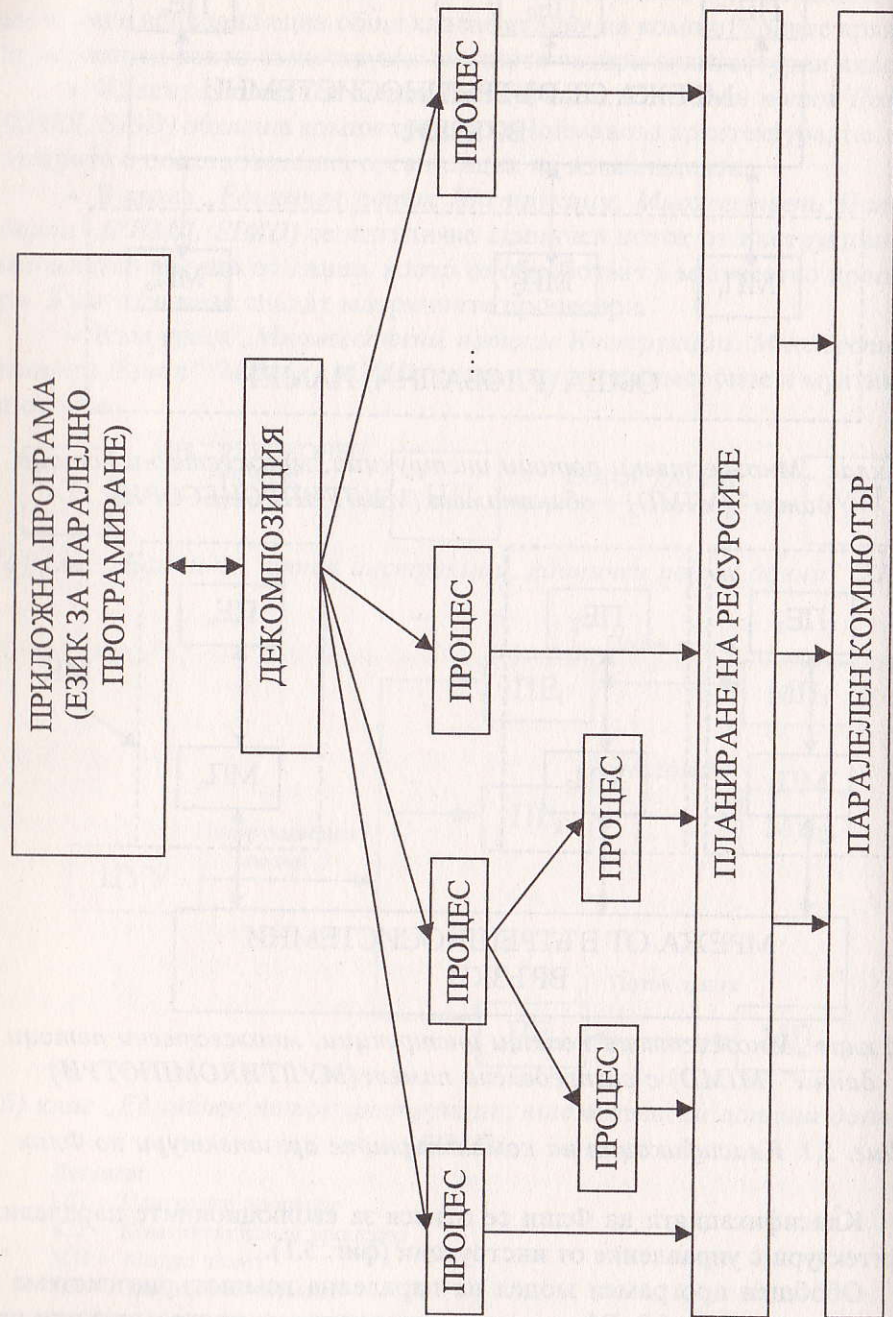


(г) клас „Множествени потоци инструкции, множествени потоци данни“ (*MIMD*) с разпределена памет (МУЛТИКОМПЮТРИ)

Фиг. 5.1. Класификация на компютърните архитектури по Флин

Класификацията на Флин се отнася за еволюционните паралелни архитектури с управление от инструкции (фиг. 5.1).

Обобщен програмен модел на паралелна компютърна система е представен на фиг. 5.2. Ефективното изпълнение на програмите при па-



Фиг. 5.2. Обобщен програмнен модел на паралелна компютърна система

ралелните компютри изисква *декомпозиция на приложната програма на отделни задачи*, за изпълнението на всяка от които се генерира по един процес. Броят на генерираните процеси зависи от типа на паралелния алгоритъм.

При *статично-паралелните алгоритми* броят на процесите, които могат да бъдат изпълнявани едновременно, е твърдо фиксиран и не се променя по време на изпълнението на програмата. *Динамично-паралелните алгоритми* изразяват паралелизма чрез множество процеси, които могат да взаимодействат помежду си и от своя страна да генерират нови процеси.

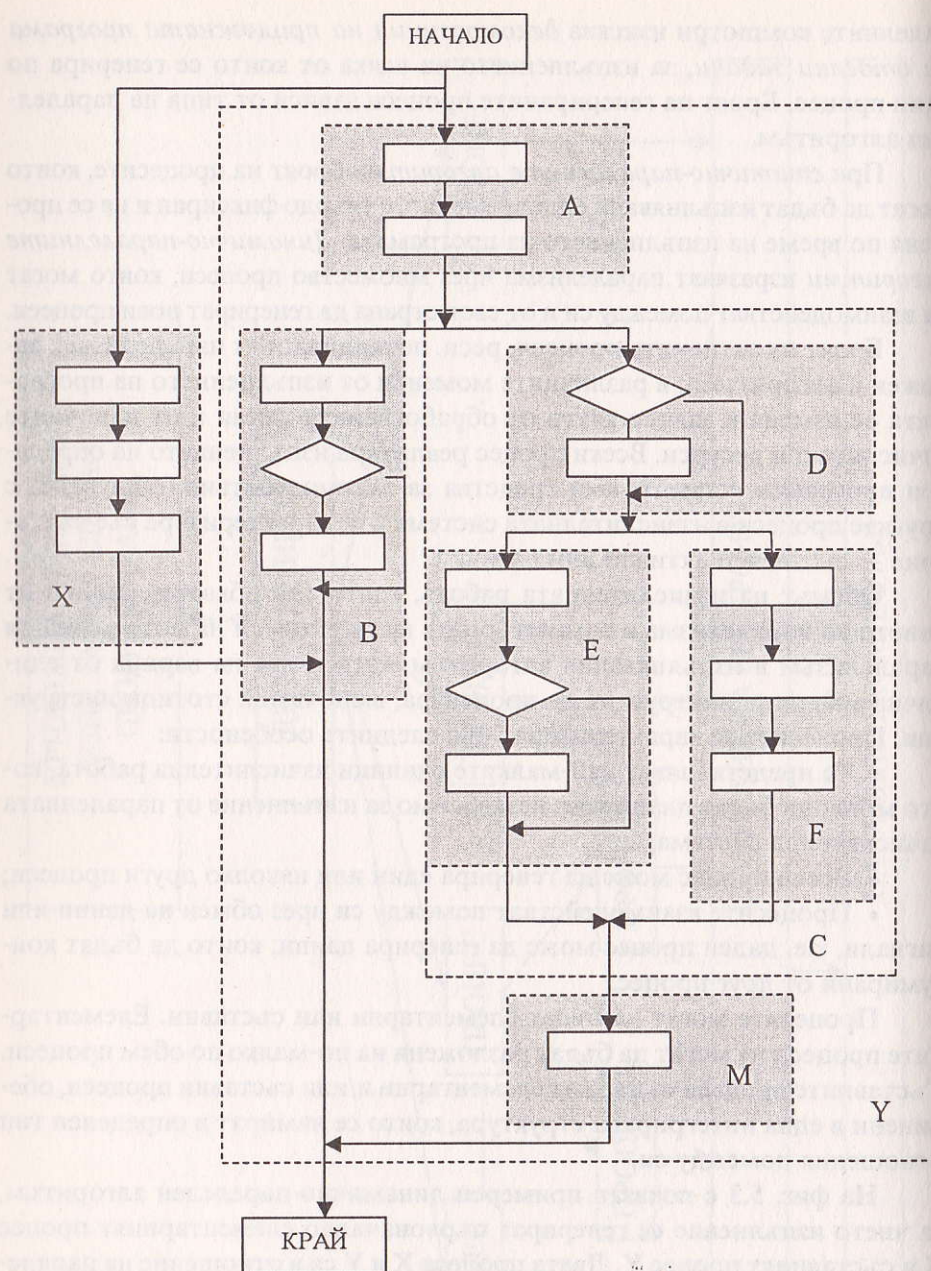
Броят на активните процеси, респ. потенциалният паралелизъм, заложен в алгоритъма, в различните моменти от изпълнението на програмата се изменя и зависи както от обработваните данни и от наличните изчислителни ресурси. Всеки процес реализира изпълнението на определен програмен сегмент, има средства за взаимодействие (общуване) с другите процеси в изчислителната система и се характеризира със състояние – съответно активно или пасивно.

Обемът на изчислителната работа, която той обхваща, зависи от нивото на паралелизма в компютърната архитектура и на потенциалния паралелизъм в изпълнявания алгоритъм, като може да варира от единична машинна инструкция до процедура, включваща стотици инструкции. Процесите се характеризират със следните особености:

- Те представляват най-малките единици изчислителна работа, които могат да бъдат планирани независимо за изпълнение от паралелната изчислителна система;
- Всеки процес може да генерира един или няколко други процеси;
- Процесите взаимодействат помежду си чрез обмен на данни или сигнали, т.е. даден процес може да генерира данни, които да бъдат консумирани от друг процес.

Процесите могат да бъдат елементарни или съставни. Елементарните процеси не могат да бъдат разложени на по-малко по обем процеси. Съставните процеси съдържат елементарни и/или съставни процеси, обединени в една интегрирана структура, които се намират в определен тип отношения помежду си.

На фиг. 5.3 е показан примерен динамично-паралелен алгоритъм, за чисто изпълнение се генерират първоначално елементарният процес X и съставният процес Y. Двата процеса X и Y са в отношение на паралелизъм, тъй като между тях не съществуват никакви причинно-следствени връзки. Съставният процес Y включва елементарните процеси A, M и B и съставния процес C. Процесът A генерира процесите B и C и следовател-



Фиг. 5.3. Генерирани процеси за реализацията на примерен динамично-паралелен алгоритъм

по той е в отношение на следствие с тези два процеса. Процесите В и С са в отношение на паралелизъм поради липсата на причинно-следствени връзки между тях.

Съставният процес С обхваща елементарните процеси D, E и F, като E и F са в отношение на паралелизъм, а процесът D от една страна и процесите E и F от друга, са в отношение на следствие с обратна връзка.

Следователно типовете отношения между процесите, генерирани при изпълнението на даден алгоритъм, отразяват степента на потенциалния му паралелизъм. Очевидно е, че в системата могат да бъдат едновременно реализирани само процесите, които са в отношение на паралелизъм. Наличието на отношение от типа на конкуренция, алтернативност, следствие и следствие с обратна връзка изискват координация между процесите, обуславяща определени форми на взаимодействие между тях, като от своя страна паралелната компютърна архитектура трябва да осигури ефективен механизъм за реализация на тези взаимодействия. Координацията между процесите в паралелната изчислителна система изисква следните форми на взаимодействие между тях:

- Синхронизация;
- Информационен обмен;
- Взаимно изключване.

Понастоящем проблемите по създаване на ефективен механизъм за координация на процесите в паралелните компютърни системи са обект на интензивни изследвания, като са предложени редица решения. Най-голямо признание е получило решението, предложено от Хоор от Оксфордския университет, известно под названието „комуникация между последователни процеси“, което представлява първият успешен опит да се осигурят трите аспекта на координацията между процесите при паралелна обработка. Решението на Хоор обединява синхронизацията и информационния обмен между процесите, като те се реализират на основата на входно-изходни оператори.

Оптимално решение може да бъде постигнато при интегрирането на трите форми на координация между процесите, т.е. тези три аспекта на взаимодействието между тях да бъдат реализирани на основата на комуникация.

Обобщеният структурен модел на паралелна компютърна система е изграден на базата на три основни градивни елемента:

- Процесорни елементи, т.е. модули, които могат да реализират аритметични и логически операции над данните. Тези елементи взаимодействат по между си посредством синхронизиращи сигнали;

- *Модули памет*, които служат за съхраняване на информация;
- *Комуникационни елементи*, които представляват модули за предаване на информация и се характеризират с пропускателната си способност.

Организирането на ефективна система за информационен обмен в рамките на паралелната компютърна архитектура е решаващ фактор за максималното използване на потенциалния паралелизъм в изпълнявания алгоритъм, поради което в значителна степен обуславя системната производителност и ефективност. Основен проблем на структурата и организацията на паралелните компютри е изграждането на ефективна и едновременно с това икономически целесъобразна мрежа от вътрешно-системни връзки.

5.2. ТАКСОНОМИЯ НА ПАРАЛЕЛНИТЕ MIMD КОМПЮТЪРНИ АРХИТЕКТУРИ

Таксономията на съществуващите паралелни компютри се основава на характеристиките на архитектурата на апаратните средства, формираща долния слой на системата и абстрактната машина в горния слой. Абстрактната машина отразява програмния модел на компютърната система.

Основните архитектурни класове на апаратни средства са:

- Системи с физическа обща памет;
- Системи с разпределена памет.

Системите с разпределена памет могат да бъдат архитектури с дистанционен достъп до паметта или архитектури с обмен на съобщения.

Основните форми на абстрактни машини са:

- Системи с обмен на съобщения;
- Архитектури с логическа обща памет.

Съществуват три решения на архитектурите с логическа обща памет:

- Архитектури с разпределена обща памет;
- Многонишкови архитектури;
- Архитектури с обща памет.

Таксономията на MIMD архитектурите е представена на фиг. 5.4. Двете основни физически форми на MIMD архитектурите са: мултипроцесори с обща физическа памет и архитектури с разпределена памет. Архитектурите с разпределена памет могат да бъдат архитектури с дис-

танционен достъп до паметта или архитектури с обмен на съобщения. Както архитектурите с дистанционен достъп до паметта така и архитектурите с обмен на съобщения се разделят от своя страна на подкласове в зависимост от програмния и комуникационния модели, които те поддържат. От едната страна подкласовете са архитектурите с разпределена обща памет и многонишковите архитектури и от другата страна – архитектури с обмен на съобщения с или без виртуална обща памет.

В списъка на качествените белези *работното множество се отнася до оползотворяването на локалността в програмите с цел редуциране на честотата на комуникация с останалите възли*. Други качествени белези са гранулата на паралелната обработка и възможностите за модулно разширяване на системата.

Само архитектурите с обмен на съобщения допускат неограничено модулно разширяване на системата. При класическите системи с обмен на съобщения комуникацията се реализира от ядрото на операционната система. В този случай при паралелните операционни системи латентността е поне с един порядък по-висока от латентността при архитектурите с дистанционен достъп до паметта. Следователно комуникационните обекти трябва да бъдат големи, което води до програмен модел с комуникиращи процеси. Многонишковите архитектури изпълняват голям брой малки процеси (нишки). Въпреки това те са високо ефективни поради ефективната схема за припокриване на латентността. Архитектурите с виртуална памет се разглеждат основно като програмен мост от работните станции с клъстери към паралелните машини.

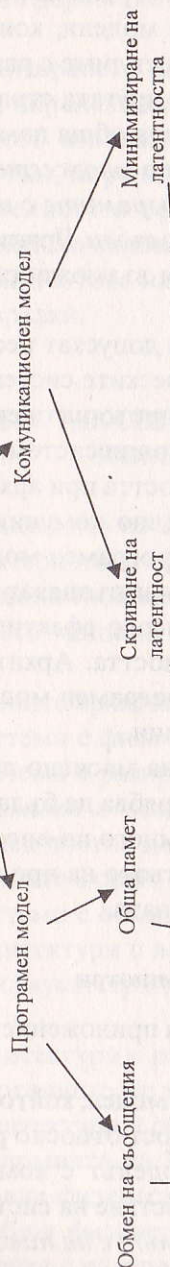
За да се оползотвори пълният потенциал на масивно паралелните компютри, съответните програмни парадигми трябва да бъдат развити с цел представяне на обмена на съобщенията на много по-високо ниво на абстракция, като по този начин отразяват естеството на приложните алгоритми, а не на начина, по който работи машината.

5.2.1. Програмни модели за паралелни компютри

Програмният модел отразява същността на приложението, а не стила на функциониране на паралелния компютър.

Модел с обмен на съобщения. Програмният модел, който е най-ефективен от гледна точка на изчисленията и най-прост относно реализацията за архитектури с разпределена памет, е моделът с комуникиращи процеси, който директно отразява начина на действие на системата. *При парадигмата с обмен на съобщения паралелизмът е на ниво комуники-*

Паралелни MIMD архитектури



Архитектура с разпределена памет	Мултипроцесори с централна памет	Архитектури с разпределена обща памет	Архитектури с виртуална обща памет	Многоонийкови архитектури
Достъп до паметта: само локален	Достъп до паметта: до централна памет	Достъп до паметта: локален и дистанционен	Достъп до паметта: само локален	Достъп до паметта: локален и дистанционен
Работни множества: не се използват на възлите	Работни множества: не се използват	Работни множества: в кеш-паметите	Работни множества: в паметите	Работни множества: не се използват
Гранула: голяма	Гранула: малка	Гранула: малка	Гранула: голяма	Гранула: малка или средна
Възможности за модулно разширяване: неограничени	Възможности за модулно разширяване: малки	Възможности за модулно разширяване: средни	Възможности за модулно разширяване: неограничени	Възможности за модулно разширяване: неограничени

Фиг 5.4 . Таксономия на паралелните MIMD архитектури

раци процеси, т.е. изпълнява се голяма гранула. Всеки възел има отделно адресно пространство. Програмните сегменти могат да бъдат процеси, включващи голям обем изчислителна работа или колекция от малки процеси. Програмистът е отговорен за осъществяването на комуникацията между процесите, като вмъква конструкциите *send*, *receive* и *reply* в програмния код в съответствие с използвания комуникационен протокол. Необходимостта от явна синхронизация на обмена на данни, както и координацията на паралелните процеси прави трудно приложението на модела с обмен на съобщения. Отговорност на програмиста е да дефинира правилно точките на синхронизация, за да осигури коректността на паралелното изпълнение.

Програмирането с парадигмата с обмен на съобщения се опростява значително за приложения, които позволяват използването на модела SPMD (Single Process Multiple Data – един процес, множество данни). В случая моделът SPMD включва една нишка, която се повтаря във всички възли и се прилага към различни множества данни. По такъв начин моделът SPMD комбинира глобалната хомогенност (множество еднакви процеси) с локалната автономност на изпълнението на процеса с различни данни. Следователно паралелизацията се свежда до задачата за разпределение на данните, а координацията на процесите се осъществява чрез схема за синхронизация „заключване на стъпка“. („lock-step“) В този режим всеки процес изпълнява три последователни фази:

- 1) получава ново множество входни данни от други процеси;
- 2) изчислява нови резултати;
- 3) изпраща резултатите към други процеси.

Съществуват важни класове приложения, които допускат използването на прост модел SPMD. Типичен проблем е решаването на частни диференциални уравнения по метода на крайните елементи, при който сегментирането на дефиниционната област на подпространства и декомпозирането на матричния алгоритъм на множество еднакви комуникиращи процеси е просто. Комуникацията също е ориентирана към решетка от процесори. При други класове приложения дефиниционната област е нерегулярна и използването на модела с комуникиращи процеси е трудно приложим. Специфичното е, че при нечислени приложения, при които в общия случай липсва високата степен на паралелизма по данни (присъщ на повечето числени приложения) и зависят силно от структури с указатели, парадигмата с обмен на съобщения може да се окаже доста неподходяща.

Програмен модел за архитектури с обща памет. При числените приложения общият програмен модел (физическа или логическа обща памет) е паралелизацията на цикли. Паралелизмът на ниво данни се представя като цикли от операции над масиви. За простота неитеративните програмни сегменти между циклите обикновено не се паралелизират. Паралелизацията се осъществява от компилатора, който генерира код за паралелно изпълнение.

Програмен модел с обща памет за MIMD архитектури с разпределена памет. Използването на глобални променливи при MIMD архитектурите с разпределена памет налага „частично подреждане“ на съответно множество оператори, при което нивото на паралелизъм е високо на ниво оператор и съответната гранула е фина.

При модела с обща памет координацията между паралелните процеси се осъществява чрез запис или четене на глобални променливи от общата памет. Нарушаването на стриктния последователен ред на четенията и записите променя семантиката на програмата. При изпълнението на програмата от последователна машина, частичното подреждане на зависимостите по данни е осигурено от общото подреждане на операторите в последователната програма. Достъпът до паметта в последователните машини е атомична (неделима) операция, т.е. редът на операциите за достъп до паметта е фиксиран от последователната програма и изискването за „частично подреждане“ автоматически е изпълнено.

Това обаче не е осигурено при паралелните операции в общата памет на паралелните компютри поради две причини:

1) В даден момент могат да се изпълняват няколко операции за достъп до общата памет едновременно;

2) Операциите за достъп до разпределената обща памет не са атомични.

За да се постигне достатъчно висока пропускателна способност на паметта, общата памет обикновено се състои от отделни банки памет, които комуникират заедно с процесорите през вътрешносистемната комуникационна мрежа. При дълги маршрути мрежовото закъснение може да нарастне значително. Това може да доведе до условия за „състезание“ между конкурентните операции за достъп до паметта, които „нарушават“ атомичността на записите и следователно водят до недетерминираност на операциите за достъп до паметта. Този проблем се преодолява чрез архитектурни разширения, осигуряващи „постоянството на последователността“ на операциите за достъп до паметта. При мултипроцесорите

това е осигурено, ако резултатите от изпълнението на паралелната програма са еднакви с тези, получени при последователната програма.

Принципът на „постоянството на последователността“ осигурява коректния ред на достъпите до паметта, но не гарантира коректността на достъпите до общите обекти на паралелните процеси. Тя трябва да бъде осигурена от програмиста чрез адекватни синхронизиращи структури за достъп до данните, т.е. необходимо е да се дефинират критични секции. Удовлетворяването на изискванията на принципа на „постоянството на последователността“ може да доведе до значителни времена за изчакване. На практика неговите предимства се осигуряват за сметка на загубата на производителност. Проблемът може да се облекчи като се приложи „слаб“ модел с „постоянство на последователността“. При този модел принципът на „постоянството на последователността“ се осигурява само в точките на синхронизация, зададени от програмиста. Редът на достъпите до паметта се заместват от реда на точките на синхронизация. Това налага програмите да се съставят от високо квалифицирани програмисти, които могат в явен вид да координират достъпите до глобалните обекти.

5.2.2. Архитектури с обща физическа памет: архитектура PRAM

Паралелната машина с произволен достъп PRAM (Parallel Random Access Machine) може да се разглежда като канонична форма на паралелния компютър, така както фон Ноймановата машина е канонична форма на последователния компютър. Паралелната машина с произволен достъп PRAM съдържа множество от независими последователни процесори (RAM's – Random Access Machines), всеки от които разполага с индивидуална (локална) памет и комуникира с останалите процесори посредством глобална памет, която всички процесори използват общо. Нека да разгледаме паралелен компютър PRAM, включващ p процесора $P_0, P_1, P_2, \dots, P_{p-1}$ и M клетки в общата памет $M_0, M_1, M_2, \dots, M_{M-1}$. Всеки процесор може да прочете съдържанието на клетката M_i от общата памет, като използва инструкцията *READ* (M_i) и да модифицира съдържанието на клетката M_i с инструкцията *WRITE* (M_i). Всеки процесор може да изпълнява атомични (неделими) операции, които изискват една единица време. Тези операции имат следната форма:

Read (M_i)

Compute (M_i)

Write (M_i)

Всички атомични операции се извършват синхронно. С други думи всичките процесори четат, изчисляват и записват едновременно.

Достъпът до паметта може да бъде осъществен по различни начини. Ако в даден момент само един процесор може да чете дадена клетка от паметта, достъпът е за ексклузивно четене (ER – Exclusive Read). Ако в даден момент няколко процесора могат да четат едновременно една и съща клетка от паметта, достъпът е за паралелно четене (CR – Concurrent Read). Аналогична класификация се използва за достъпа за запис. Записът е ексклузивен (EW – Exclusive Write), ако в даден момент само един процесор може да осъществява запис в дадена клетка от общата памет. Записът е паралелен (CW – Concurrent Write) ако в даден момент няколко процесора извършват запис в една и съща клетка от паметта. В случая на паралелен запис процесорите могат да записват различни данни, при което възниква конфликт. Няколко модела са предложени за решаването на тези конфликти:

- Прост модел: всички процесори трябва да записват една и съща стойност в една и съща клетка на паметта;
- Произволен модел: независимо от последователността на записите в общата клетка на паметта алгоритъмът трябва да се изпълни коректно;
- Модел с приоритети: процесорите имат присвоени приоритети и само процесорът с най-висок приоритет ще осъществи записа в общата клетка на паметта.

Алгоритъмът за PRAM представлява последователност от атомични операции, които всеки процесор трябва да изпълни. Началните данни за алгоритъма са разположени в общата памет и резултатите след обработката се записват също в общата памет. Различните модели PRAM формират следната йерархия:

- EREW – един читател, един писател;
- CREW – много читатели, един писател;
- CRCW (common) – много читатели, много писатели, прост модел;
- CRCW (random) – много читатели, много писатели, произволен модел;
- CRCW (priority) – много читатели, много писатели, приоритетен модел;

Тясното място на физическата обща памет – достъпът – може да бъде преодоляно до известна степен чрез придаване на случаен характер на достъпа, като се прилага двуфазна схема. През първата фаза възелът-

източник изпраща заявката си за достъп до произволен междинен възел, избран по случаен признак. Всеки възел в системата може да изпълнява ролята на междинен възел с вероятност $1/N$, където N е броят на възлите. През втората фаза междинният възел придвижва заявката към дестинацията. Това е основната концепция за паралелната машина с произволен достъп PRAM. Схемата за случаен избор минимизира вероятността за конфликти при достъпа за сметка на увеличаването на закъснението за достъп поради двуфазната схема.

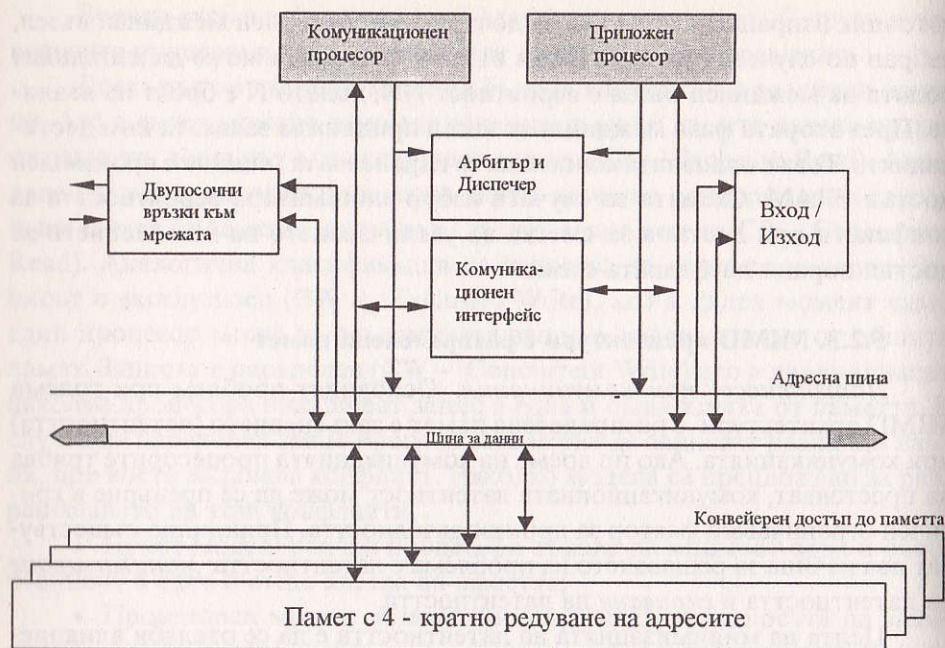
5.2.3. MIMD архитектури с разпределена памет

Латентност при комуникация. Основният проблем при големи MIMD архитектури с разпределена памет е закъснението (латентността) при комуникацията. Ако по време на комуникацията процесорите трябва да престояват, комуникационната латентност може да се превърне в критичен ограничаващ фактор за производителността. Принципно съществуват два начина за решаването на проблема с латентността: *минимизиране* на латентността и *скриване* на латентността.

Целта на минимизацията на латентността е да се отслаби влиянието на комуникацията до пренебрежими стойности или чрез нейното намаляване или чрез намаляването на честотата на комуникационните транзакции. Скриването на латентността е организационна схема, която позволява на процесорите да вършат изчислителна работа по време на комуникацията. Двата метода могат да бъдат комбинирани.

Традиционният подход е да се минимизира честотата на комуникационните транзакции, като за всеки възел се осигури достатъчно голяма и бърза вторична кеш-памет в допълнение на първичната кеш-памет, която е интегрирана в чипа на микропроцесора. По този начин работните множества могат да се поддържат във вторичната кеш-памет. Цената на това решение е значителна. На първо място използването на кеша създава *проблем за кохерентността* относно *глобалните променливи* в изчислителната среда с разпределена памет, който изисква стандартните техники, използвани в архитектури с централна шина към паметта, т.е. подслушване и инвалидация на позицията в кеша.

Комуникационната система включва *апаратни и програмни слоеве*, които *съвместно увеличават латентността*. При мултипроцесорите с обща физическа памет компонентата, свързана с хардуера, е времето за достъп през мрежата, докато софтуерът изисква допълнител-



Фиг. 5.5. Архитектура на възела, осигуряващ скриване на латентността при комуникация

ни разходи за поддържане на „постоянството на последователността“ на достъпите до паметта. При архитектурите с обмен на съобщения хардуерната компонента е времето, необходимо за трансфера на съобщението от източника до дестинацията през мрежата, докато компонентата, свързана със софтуера, е времето за инициализация на предаването, необходимо на ядрото на операционната система да изпълни междупроцесорната комуникация. Следователно минимизирането на латентността изисква комбинираните усилия на специализиран хардуер и оптимизирани програмни средства.

5.2.4. Основни категории архитектури с разпределена памет

При архитектурите с разпределена памет могат да се използват два различни механизма за комуникация между възлите. Всеки възел трябва да може да осъществява достъп не само до неговата локална памет, но и до паметите на другите възли или възелът да може да комуникира чрез изпращане/получаване на съобщения към/от другите възли.

Следователно съществуват две различни архитектурни категории: *архитектури с дистанционен достъп до паметта и архитектури с обмен на съобщения*. Също така в тази категория може да бъде класифицирана поне концептуално многонишковата архитектура. Двете архитектури се различават по начина, по който решават проблема с латентността на комуникацията. Архитектурите с разпределена обща памет използват апаратни средства, минимизиращи латентността, докато многонишковите архитектури използват нейното скриване. И при двете архитектури нивото на паралелизъм е високо, т.е. гранулата е фина.

Архитектурите с обмен на съобщения и архитектурите с дистанционен достъп до паметта се различават по размера на латентността на комуникацията. Латентността при архитектурите с обмен на съобщения включва времето за трансфера на обекта (съобщението) в системата от адресното пространство на единия възел до адресното пространство на другия възел. Латентността може да бъде редуцирана чрез комуникационни апаратни и програмни средства и по-нататък от схема за скриване. Въпреки това, дори в най-добрия случай латентността при системите с обмен на съобщения е относително висока. Следователно обемът на изчислителната работа в процесора трябва да бъде значителен, за да се преодолеят загубите от допълнителните разходи за комуникация. Следователно архитектурите с обмен на съобщения оползотворяват ниско ниво на паралелизъм, т.е. гранулата е голяма.

Естествената парадигма за програмиране за архитектурите с обмен на съобщения е програмният модел с обмен на съобщения. При тях, за да се изпълни парадигмата за програмиране с обща памет, е необходимо да се наложи механизъм за виртуална обща памет над системата с обмен на съобщения. При архитектурите с дистанционен достъп естествената парадигма е общата памет.

Архитектура на възела. Основното решение при високо паралелните архитектури е дали да се използва суперскаларен или векторен процесор във възела. При първото поколение паралелни компютри като Intel iPSC-2, SUPRENUM и Ametek се използваха конвейерни векторни процесори. Положението се промени с появата на суперскаларните процесори, които осигуряват висока производителност за числа с плаваща запетая. При всички положения за предпочитане е да се използва *векторен процесор* (CM-5 на Thinking Machine, Meiko CS-2 с векторен процесор на Fujitsu, допълващ възела SPARC).

Всеки възел в архитектурата с обмен на съобщения има свое собствено локално адресно пространство. Данните се адресират чрез логи-

чески адрес, съдържащ идентификатор на процеса и име на обекта, които вътрешно се трансформират във физически адрес, включващ номера на възела и локален адрес. Обектът се обработва във възела, в който той е резидентен. Единствената причина за изпращане на копия на обекта данни към други възли е за комуникация между процесите.

След установяването на логически канал между процесите от ядрото на операционната система в съответствие с изискванията на комуникационния протокол, копирането се осъществява физически чрез механизма за директен достъп до паметта направо от паметта на възела-източник до паметта на възела-дестинация. На нивото на операционната система може да се изпълни *синхронна комуникация без буфериране*, като по този начин се избягват допълнителните разходи за копиране и буфериране. От друга страна максимизирането на паралелното изпълнение на приложната програма изисква *асинхронно изпращане без чакане (no-wait send)*. Още повече асинхронната комуникация осигурява значителни удобства за програмиста. Естествено тя изисква *буфериране*.

Времето за стартиране на комуникационната транзакция зависи от функционалността на ядрото на операционната система. Най-простият модел съответства на специализирана машина, при която комуникацията се осъществява чрез библиотечна подпрограма, свързана с приложната програма.

Разделянето на ядрото осъществява първото ниво на защита, като ядрото на операционната система се изпълнява в режим супервайзор, докато всичко останало, включително сервизните програми на операционната система на високо ниво, се изпълняват в режим потребител. Следващият етап е многозадачният режим, който добавя необходимостта от планиране и разделяне на задачите. При конвенционален еднопроцесорен възел всяка комуникация води до извикване на операционната система, което превключва средите за задачите. Най-високото ниво на сложност е в режим с много потребители, който изисква допълнително необходимостта от защита на задачите. Друг проблем е дали комуникацията трябва да бъде защитена от протокол за потвърждение. Съществуват множество приложения, които допускат толериране на спорадични грешки при комуникацията в случаите, когато вероятността за такива грешки е много малка.

Нежелателните ефекти на латентността на комуникацията значително могат да се намалят, като се използва *специализиран комуникационен процесор (КП)* във възела като допълнение към приложния процесор (ПП).

При тази симбиоза единствената задача на ПП е да осигурява високоскоростно изпълнение на приложната програма без прекъсване, докато *КП изпълнява задачите за комуникация на ядрото на операционната система*. Двата процесора работят паралелно. В следствие закъснението при стартиране на комуникацията възниква в КП, но то е „скрито“ (прозрачно) за ПП. Тази схема на „скриване“ на латентността е стандартен подход във второто поколение комерсиални системи с обмен на съобщения (Intel Paragon, NEC Cenju-2, Meiko CS-2). За пп съществува само закъснение при изпращането на заявка за комуникация към КП.

Вътрешната архитектура на възел, осигуряващ скриване на латентността при комуникацията, е показан на фиг. 5.5.

ПП и КП могат да бъдат идентични (напр. суперскаларни процесори), които ползват общата памет на възела, която в общия случай е DRAM. Задължително е да се осигури висока пропускателна способност на паметта към възела, като се използва конвейерен достъп и модулна организация на паметта с редуване на адресите. ПП и КП комуникират през общата памет на възела.

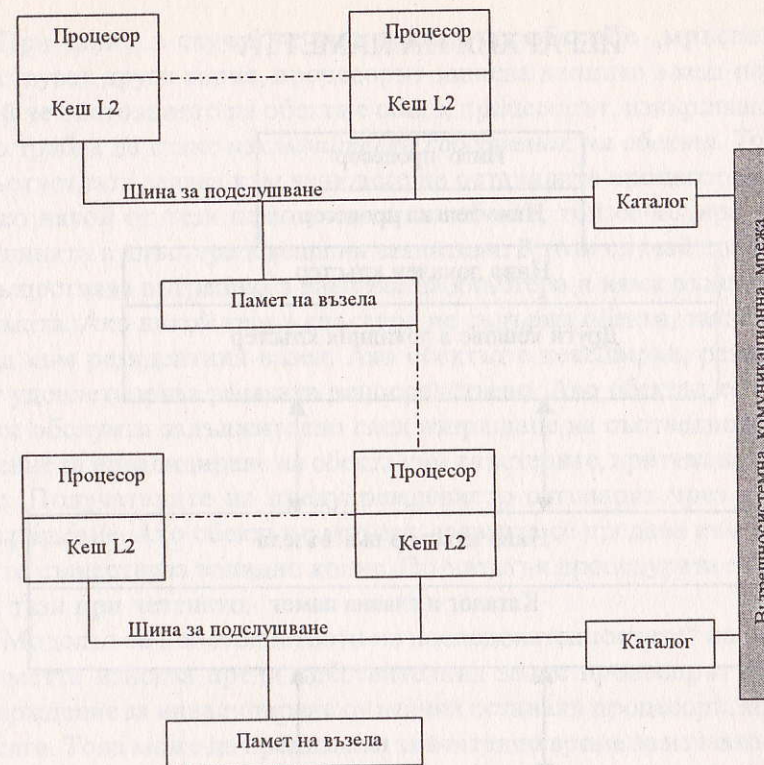
5.2.5. Архитектури с разпределена обща памет

При MIMD системи с разпределена памет по възлите общата системна пропускателна способност на паметта расте пропорционално на броя на възлите. Това прави архитектурата по принцип модулна. Модулното нарастване на системата обаче е възможно в определени граници. За да се поддържа дистанционният достъп, *паметите на възлите са обединени в глобално адресно пространство*. Глобалният логически адрес се преобразува от хеширащия хардуер в съответен физически адрес (номер на възел, адрес в паметта на възела). *Възелът, в който обектът е дефиниран и се намира постоянно, се нарича резидентен възел*. Заявките за дистанционен достъп се маршрутизират в мрежата. Следователно латентността при дистанционния достъп е значително по-голяма от латентността при локален достъп. Тя е функция на дистанцията за комуникация в мрежата между възела-източник и възела-дестинация. Следователно в зависимост от типа на мрежата могат да съществуват значителни вариации на латентностите при дистанционен достъп. Тази организация изисква поддържането на модела с постоянство на последователностите на достъпите до паметта.

При архитектурите с разпределена обща памет проблемът с латентността се облекчава с използването на вторична кеш-памет за всеки

процесор. В зависимост от локалното поведение на приложната програма наличието на тази кеш-памет ще редуцира, в по-голяма или по-малка степен честотата на дистанционните достъпи до паметта. *В този случай общите обекти са конкретни линии в кеша.* Разбира се при този подход възниква проблемът за кохерентност на съдържанието на кеша. В рамките на всеки възел може да се приложи стандартният метод за подслушване на общата шина на паметта. Еквивалентният подход в паралелната среда изисква всяка кеш-памет да информира останалите възли за всеки свой запис. Това означава, че всеки възел трябва да осъществява „разпръскване“ (глобална комуникация) към всички останали възли в системата на адресите, на които е осъществен запис, като в резултат възниква изключително интензивен трафик от съобщения в мрежата.

Значително подобряване на системната производителност се получава при следното алтернативно решение на проблема за кохерентността: всеки възел разполага с каталог, в който е отразена информация за глобалните обекти в паметта на възела. Кохерентността на кеша обикновено се поддържа чрез т.нар. „инвалидиране“ на кеша. При този метод записът в кеш-паметта води автоматично до инвалидирането на всички копия на този обект в останалите възли. Възелът, осъществил запис (т.е. модифицирането на глобалния обект), трябва да предупреди всички останали възли, които имат копие на този обект, за инвалидирането му. Информацията за това, на кои възли да изпрати предупреждението за инвалидиране, се съдържа в каталога. Всеки следващ достъп за четене на този невалиден обект в останалите възли предизвиква грешка в кеша. При това се извършва четене на новото копие на модифицираната позиция в кеша на резидентния възел. Тясното място на централизиран каталог се преодолява чрез разпределена схема на каталогите, при която всеки възел разполага със собствен локален каталог, който за всеки глобален обект в паметта на възела съдържа съответни указатели към останалите възли, разполагащи с копия на тези обекти. На фиг. 5.6 е показана архитектурата на възела при системи с разпределена обща памет. Всеки възел съдържа клъстер процесори, всеки от които разполага с вторична кеш-памет. Процесорите използват общата памет на възела и локалния каталог. Всеки процесор може да подслушва кеш-паметите на останалите процесори в клъстера. Ако процесорът не открие даден обект в първичната си кеш-памет, той може да го открие на едно от следните места: в собствената си вторична кеш-памет, във вторичната кеш-памет на друг процесор в клъстера, в локалната памет на клъстера или в паметта на друг клъстер. Така се формира йерархията на достъпа, представена на фиг. 5.7.



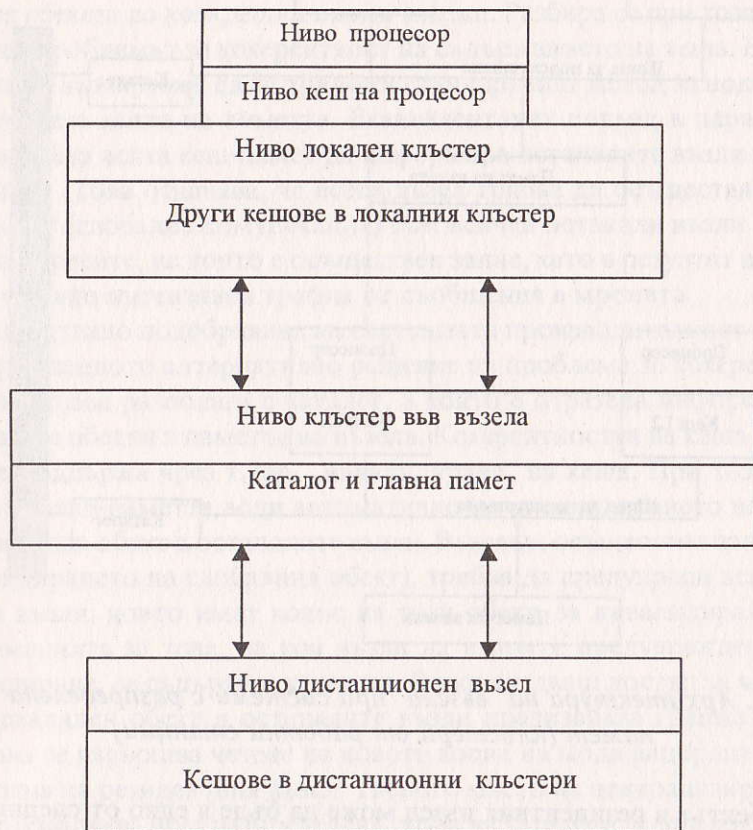
Фиг. 5.6. Архитектура на възела при системи с разпределена обща памет (клъстери от работни станции)

Обектът в резидентния възел може да бъде в едно от следните три състояния:

- **некеширан:** не съществува копие на обекта в системата;
- **общ:** съществуват немодифицирани копия на обекта в кеш-паметите на някои процесори;
- **мръсен:** съществува само едно модифицирано копие в кеша на един процесор (всички останали копия на обекта са невалидни).

Състоянието на всеки обект е отразено в каталога чрез съответен признак. При четенето на обект първо се извършва обръщение към собствената вторична кеш-памет. Ако това обръщение предизвика грешка в кеша (поради липса на обекта в него), процесорът се обръща към останалите процесори в клъстера. При липса на обекта заявката за четене се препраща към резидентния възел на обекта. Резидентният възел чете обек-

ЙЕРАРХИЯ НА ПАМЕТТА



Фиг. 5.7. Йерархия на достъпа до паметта при архитектура с разпределена обща памет

та от паметта на клъстера и едновременно проверява състоянието му в каталога. В случай, че обектът не е кеширан или е общ, той го изпраща към възела-заявител и състоянието на обекта в каталога съответно се актуализира. Ако статусът на обекта е „мръсен“, резидентният възел изпраща заявката заедно с адреса на заявителя към възела, съдържащ единственото валидно копие на обекта. Получателят на заявката изпраща валидно копие на обекта директно към заявителя.

При запис, в случай че състоянието на обекта е „мръсно“, т.е. не съществуват други копия, процесорът записва данните в кеш-паметта. В случай че състоянието на обекта е общо, процесорът, извършващ запис, първо трябва да стане *изключителен собственик на обекта*. Той изпраща съответната заявка към кешовете на останалите процесори в клъстера. Ако някой от тези кешове съдържа обекта, той се копира директно през шината в клъстера в кеша на заявителя. В този случай транзакцията се осъществява вътрешно в рамките на клъстера и няма външен ефект в системата. Ако никой кеш в клъстера не съдържа обекта, заявката се изпраща към резидентния възел. Ако обектът е некеширан, резидентният възел удовлетворява заявката непосредствено. Ако обектът е общ, заявката се обслужва задължително след изпращане на съответното предупреждение за инвалидиране на обекта към клъстерите, притежаващи негово копие. Получателите на предупреждението отговарят чрез позитивно потвърждение. Ако обектът е мръсен, заявката се предава към клъстера, в който съществува валидно копие. По-нататък процедурата е аналогична на тази при четенето.

Моделът за „постоянството на последователностите“ на достъпите до паметта изисква преди действителния запис процесорът да получи потвърждение за инвалидиране от всички останали процесори, които това ги засяга. Това може да предизвика значително време за изчакване, което може да се избегне като се приложи по-малко стриктно изискване за „постоянството на последователностите“ на достъпите до паметта. Този модел се основава на разграничаването на синхронния от асинхронния достъп. При синхронния достъп редът се осигурява от съответния ред за „заклучване“ и „отключване“ на съответните сегменти на паметта. Редът на изпълнение при асинхронния достъп може да бъде произволен. Следователно *последователността на достъпите се осигурява от специфициран ред на точки на синхронизация чрез операции lock и unlock*. Коректното изпълнение на програмата се осигурява от програмиста, който е отговорен за правилния избор на точките на синхронизация. Резултатите от симулациите на този модел показват от 10% до 40% увеличаване на производителността. При този метод не е необходимо заявките за четене да изчакват завършването на предхождащите заявки за запис.

Архитектурите с разпределена обща памет с използване на каталози са по-ефективни от архитектурите с виртуална обща памет. Те обаче не са подходящи за масивно паралелни системи, т.к. капацитетът на паметта, необходима за каталозите, се увеличава квадратично с увеличаването на броя на процесорите в системата $O(N^2)$. За N -процесорна

система с N/P клъстера капацитетът на паметта С се определя по следния начин:

$$C > N^2 * M / (P * L) \text{ (бита)} \quad (5.1)$$

където:

Р – брой на процесорите в един клъстер;

М – капацитет на паметта на клъстера в байтове;

L – брой на байтовете в една позиция на кеша.

Така например, за 256 клъстера, 16 Mbytes клъстерна памет и 32 bytes линията в кеша, 1/8 от паметта се използва за каталога. Това ограничава възможностите за модулно разширяване на архитектурата с разпределена обща памет в най-добрия случай до средно големи инсталации. Масивно паралелна система с 1000 и повече възела не може да бъде реализирана като архитектура с разпределена обща памет.

Друг проблем е 32-битовият адрес на процесорите. Поради глобалното адресно пространство общият капацитет на всички клъстерни памети се ограничава до 4 Mbytes. Така за паралелна система с 256 bytes, максималният капацитет на клъстерната памет е 16 Mbytes, което е недостатъчно при съвременните стандарти.

5.2.6. Многонишковите архитектури

Аналогично на архитектурите с разпределена обща памет многонишковите архитектури имат глобално адресно пространство. Нишката се дефинира като „лек“ или „лек като перце“ процес. *Лекият процес е нишка, в която не се осигурява защита на индивидуалното адресно пространство. Процесът-перце представлява лек процес с минимално контекстно превключване. Нишката може да бъде латентна или готова.* Механизмът за управление е от потока данни (аналогично на архитектурите с управление от потока данни – dataflow). Следователно нишката е в латентно (пасивно) състояние, докато всички нейни операнди са готови, при което тя се активизира за изпълнение в момента на получаване на операндите.

Заявките за дистанционен достъп автоматично суспендират нишката (временен прекратяват нейното изпълнение) до завършването на трансфера. Междувременно се обработва друга нишка. Следователно времето за дистанционен достъп се скрива, т.е. се припокрива с изчислителна работа. Тази схема обаче води до голям брой контекстни превключвания между нишките всеки път, когато се осъществява дистанционен достъп.

От тук следва, че схемата е ефективна, ако допълнителните разходи за контекстното превключване са много малки. Това може да бъде постигнато по два начина:

1) Нишката се простира от един дистанционен достъп до друг. Следователно не се налага съхраняването на съдържанието на регистрите за данни, а само на регистрите на състоянието. Няма да се налага съхраняването на съдържанието на регистрите за състоянието, ако съобщението, носещо операнда за нишката, съдържа и контекста. По този начин времето за превключване на средите може да бъде сведено до 10 машинни цикъла.

2) Компютърът има множество регистрови файлове, така че контекстът на няколко нишки може да бъде съхраняван в специализирани регистрови файлове. Превключването на нишките се отразява само чрез смяна на указателя към актуалния регистров файл. По този начин нишките могат да бъдат по-големи, а следователно техният брой в програмата ще бъде по-малък.

Първият подход предполага голям брой малки нишки. Вторият подход допуска по-големи нишки, но е по-скъп поради необходимостта от множество регистрови файлове.

Управлението от данни при нишките изисква някаква форма на *бариерна синхронизация*, за да се осигури стартирането на изпълнението на нишката едва, след като са налични всички операнди. Синхронизацията на нишките се поддържа от прибавянето на допълнителен бит-признак „пълнен-празен“, който индицира дали клетката може да бъде четена (ако е пълна) или в клетката може да се извърши запис (ако е празна). Тази схема е използвана в първата многонишкова архитектура (с обща памет) – машината НЕР. При многонишковата архитектура на Т машината се използват две инструкции load: load за зареждане от локалната памет и rload за зареждане на регистрите от дистанционна памет. Нека да разгледаме пример за използване на двата начина за зареждането на регистрите при изпълнението на нишка във възел N1:

$vA = rload\ pA$

$vB = rload\ pB$

$C = vA * vB$

Където vA и vB са операнди, pA и pB са указатели към клетки от паметта, а C съдържа резултата. Използването на инструкцията *rload* показва, че операндите са дистанционни, като те се съдържат съответно във възлите N2 и N3 (фиг. 5.8). Както се вижда, нишката може да извърши няколко дистанционни операции *rload*. Тя обаче няма да се блокира, до-

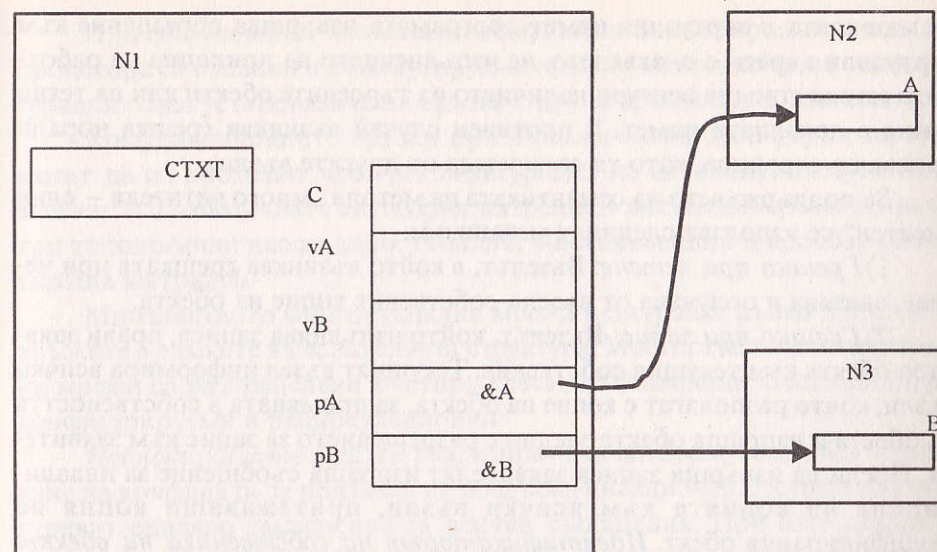
като чака получаването на съобщението. Полученото съобщение носи указател на нишката, който включва контекста за изпълнение. В примера на фиг. 5.8 контекстът на изпълнението на нишката във възел N1 се представя от променливата СТХТ. В зависимост от естеството на езика за програмиране СТХТ може да бъде указател на стек, указател на рамка или текущ указател на обект. Променливите vA, vB и C са валидни в контекста, дефиниран от СТХТ. Всяка операция `load` стартира нова нишка чрез изпълнение на операция `fork`, т.е. разпростират се толкова нишки, колкото операции `load` трябва да бъдат реализирани. Накрая нишките се съединяват чрез операции `join`.

Специфичните програмни модели на многонишковите архитектури зависят от гранулата на нишките.

5.2.7. Архитектури с виртуална памет

При тези архитектури процесорите могат да осъществяват достъп само до паметите на своите възли, а не могат да осъществяват директен достъп до паметите на останалите възли. Следователно ако процесорът иска да осъществи достъп до общ обект, който не е в паметта на възела, обектът или неговото копие трябва да бъде прехвърлен в локалната памет. Това може да бъде реализирано от механизъм, подобен на грешката за достъп до липсваща страница при виртуалната памет. Следователно общите обекти съществуват в глобално адресирана виртуална обща памет.

Съществуването на множество копия на обекти в паметта на системата създава проблем за постоянството на последователността винаги, когато копие се модифицира при запис. Лесно може да се разпознае аналогията с мултипроцесорните системи с обща памет, при които процесорите разполагат с кеш-памет. При тази аналогия паметите на възлите изпълняват ролята на кешовете, а виртуалната обща памет изпълнява ролята на главна памет. Следва, че механизмите за поддържането на постоянството на достъпите до паметта при архитектури с виртуална обща памет са аналогични на тези, използвани при решаването на проблема за кохерентност на кеша. Разбира се, конкретната форма на този механизъм трябва да се базира на съответната система за обмен на съобщения в архитектурата. Допълнителният капацитет на паметта нараства линейно с нарастването на броя на процесорите. Следователно архитектурите с виртуална обща памет се характеризират с възможности за модулно разширение, характерни за архитектурите с обмен на съобщения.



Фиг. 5.8. Възел N1 обработва дистанционни операнди от възли N2 и N3

Следните правила гарантират правилното манипулиране на общи обекти от данни:

1) Общ обект може да бъде прочетен от произволен брой нишки. Във всеки момент от времето само една нишка може да извършва запис в обекта.

2) Преди записа в общ обект всички останали копия на обекта трябва да се маркират като невалидни.

Този метод се нарича „много читатели – един писател“.

Важен аспект е големината на гранулата на общия обект. Възможно решение е общите обекти да бъдат страници, а паметите на възлите да се организират като виртуална памет със странична организация. При съвременното обектно-ориентирано програмиране по-целесъобразно е да се използват общи подходящо дефинирани обекти отколкото страници с фиксиран размер. При този подход се подобрява използването на страниците, тъй като страницата може да съдържа няколко обекта.

За разлика от архитектурите с дистанционен достъп до паметта виртуалният адрес при архитектурите с виртуална разпределена памет не указва къде се намира адресирания обект, така че не съществува резидентен възел. Всеки обект има собственик – това е процесът, който има правото да извършва запис в обекта, т.е. да го модифицира. Аналогично

на машината с виртуална памет програмата извършва обръщение към виртуални адреси с очакването, че изпълнението на принципа на работното множество ще осигури наличието на търсените обекти или на техни копия в локалната памет. В противен случай възниква грешка поради липсваща страница, като тя се зарежда от другите възли.

За поддържането на семантиката на метода „много читатели – един писател“ се използва следният механизъм:

1) *Грешка при четене*: Възелът, в който възниква грешката при четене, заявява и получава от възела-собственик копие на обекта.

2) *Грешка при запис*: Възелът, който изпълнява записа, прави заявка за обекта към текущия собственик. Текущият възел информира всички възли, които разполагат с копие на обекта, за промяната в собствеността на обекта и изпраща обекта заедно с разрешението за запис към заявителя. Преди да извърши записа заявителят изпраща съобщение за инвалидиране на копията към всички възли, притежаващи копия на модифицирания обект. *Идентификаторът на собственика на обекта се намира в таблицата на страниците на мениджъра на виртуалната памет във възела*. Възелът-собственик на обекта трябва да знае кои други възли разполагат с копия на обекта.

Недостатък на този подход е, че в случаите, когато два или повече възела записват алтернативно в една и съща страница, се получава нежелателният ефект на непрекъснато прехвърляне на страницата между тези възли. Това нежелателно явление може да се избегне, като се използва методът „много читатели – много писатели“.

Архитектурите с виртуална разпределена памет могат да се разглеждат като „мост“ между разпределената обработка (кълъстите при работните станции) и паралелната обработка (MIMD компютър с обмен на съобщения). Ако двете системи имат една и съща операционна система, еднакви компилатори и еднакви механизми, поддържащи виртуалната памет, програмите, разработени за първия тип компютърни системи, могат да бъдат изпълнени без проблем на втория тип компютърни системи и обратно.

5.3. ВЪТРЕШНОСИСТЕМНИ КОМУНИКАЦИОННИ МРЕЖИ ЗА ПАРАЛЕН КОМПЮТРИ

Топология. Топологията на мрежата е един от ключовите фактори, определящи производителността, надеждността и възможностите за самовъзстановяване на паралелната компютърна система.

При *статичните топологии* комуникационните връзки между два процесора са пасивни и некомутируемите шини не подлежат на реконфигурация с цел установяване на връзка с други процесори в системата.

Комуникационните връзки при *динамичната категория мрежи* могат да се променят чрез реконфигурация на активните комутаторни елементи. Динамичната категория вътрешно- системни мрежи обхваща три топологични класа: едностъпални, многостъпални и кросбар (комутираща матрица).

Мнозинството многостъпални мрежи осигуряват пълна система на връзките в рамките на паралелната структура. Многостъпалните двустранни мрежи са разпределени в четири класа: неблокиращи, блокиращи, реконфигурируеми и рециркуляционни.

Неблокиращите мрежи реализират всички пермутации без възникване на конфликти за ползване на общ комуникационен ресурс, като осигуряват еднакво закъснение за всички съобщения. При *блокиращите мрежи* реализацията на някои пермутации при едновременни заявки за ползване на общ комуникационен ресурс води до временно локално блокиране в мрежата, в резултат на което закъсненията на различните съобщения варират и се налага буфериране до освобождаване на съответния комуникационен ресурс.

Реконфигурируемите мрежи осигуряват полиморфна структура на връзките чрез динамична промяна на топологията на мрежите. Динамичните промени в топологията на тези мрежи могат да бъдат вътрешнофазови или междофазови и се реализират с цел:

- *Архитектурна реконфигурация*, т.е. преминаване от едно архитектурно състояние в друго за максимално ориентиране на структурата и организацията на паралелната компютърна система към специфичните особености на изпълнявания алгоритъм;

- *Реконфигурация за устойчивост на откази*, т.е. елиминиране на влиянието на отказалите системни ресурси върху функционирането на компютърната система като цяло.

Рециркуляционните мрежи реализират вътрешносистемния информационен обмен посредством рециркуляция на съобщенията по циклични маршрути и се характеризират от една страна с висок коефициент на използване на апаратните ресурси, а от друга – с повишена степен на блокиране на мрежата, т.е. изискват буфериране на съобщенията на няколко нива и сложна приоритетна дисциплина на обслужване.

Мрежи с пълна система на връзките. Мрежите с пълна система на връзките (всеки-с-всеки) по същество представляват единствен кому-

татор, който свързва всички входи към всеки изход. Диаметърът на мрежата, който представлява най-дългия маршрут, е една връзка. Локалната степен на възела е N , където N е броят на свързаните процесори. Отказът на комутатора води до отказ на цялата мрежа. Отказът на една връзка изолира от паралелната система само един процесор. Пример за такава мрежа е общата шина, при която цената е $O(N)$. Основен недостатък на общата шина е, че в даден момент от време може да се осъществи само един информационен трансфер, така че пропускателната способност на мрежата е $O(1)$. На практика пропускателната способност е дори по-ниска от $O(1)$ поради факта, че тактовата честота на общата шина намалява с увеличаването на броя на портовете. Друга мрежа с пълна система на връзките е кросбарът. Осигурява пропускателна способност $O(N)$, но цената му е пропорционална на N^2 , т.е. $O(N^2)$. При всички случаи не е възможно да се осигури модулно разширение на мрежите с пълна система на връзките. Въпреки това те имат важно практическо приложение, тъй като индивидуалните комутатори в общия случай осигуряват вътрешно пълна система на връзките и се използват като базови градивни блокове за изграждането на по-големи мрежи. Показател за технологичното ниво на мрежите е степента на комутатора при приемлива цена. С увеличаването на плътността на компонентите в чипа броят на възлите, между които се осъществява пълна система на връзките от комутатора, се увеличава.

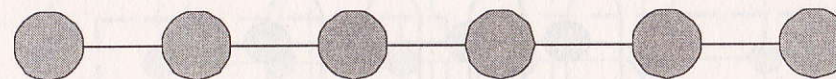
Линейни мрежи и рингове. Най-простата мрежа осигурява линейно свързване с двупосочни връзки на възлите, означени с последователни номера $0, 1, 2, \dots, N-1$ (фиг. 5.8). Диаметърът при линейната мрежа е $N-1$, средната дистанция е приблизително $2/3 \cdot N$, а отказът на единична връзка разделя мрежата на две несвързани части (бисекция). Маршрутизацията в такива мрежи е тривиална, тъй като за всяка двойка възли съществува само един маршрут. Поради липсата на алтернативни маршрути мрежата с линейна структура не осигурява възможности за самовъзстановяване т.е. при наличие на откази да може да продължи да функционира макар и с намалено бързодействие. Мрежата съдържа $N-1$ връзки и техническата ѝ реализация е проста при използването само на къси връзки.

Ринг (торус) от N възела (фиг. 5.9) се формира просто чрез свързване на двата края на мрежата с линейна структура. При реализация с двупосочни връзки, диаметърът на ринга е $N-1$, средната дистанция е $N/2$ и съществува (уникален) единствен маршрут между всяка двойка възли.

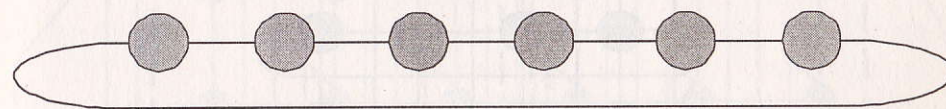
Релативният адрес между възел B и възел A е $(B-A) \bmod N$. При двупосочни връзки, диаметърът е $N/2$, средната дистанция е $N/3$, степента

на възела е 2, а разделящите връзки, при които се получава бисекция на мрежата са две. Съществуват два маршрута (два релативни адреса) между всяка двойка възли, така че мрежата продължава да функционира, макар и с намалено бързодействие, при отказ (прекъсване) на една връзка. Мрежата с рингова структура лесно се реализира с използване на къси връзки. Мрежата може да бъде разделена на подмрежи с линейна структура.

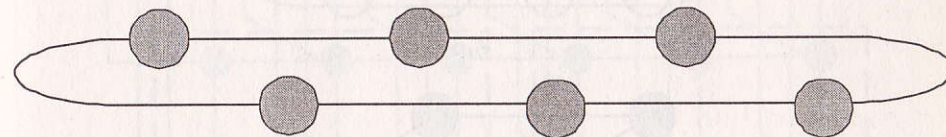
Едномерните мрежи се използват като градивни модули за мрежи с по-сложни конфигурации. Простата маршрутизация и ниската апаратна сложност ги прави изключително популярни при локалните компютърни мрежи. Простата техническа реализация на мрежата с къси връзки дава възможност да се осигури голяма ширина на връзките (на практика от 16 до 128 бита и повече).



Фиг. 5.8. Линейно свързване



а.

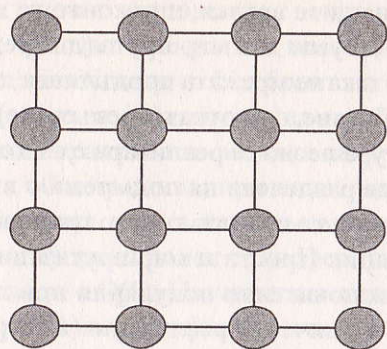


б.

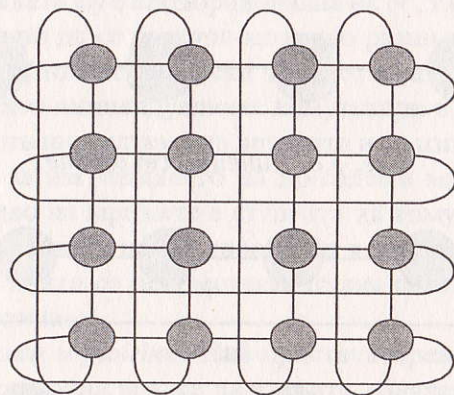
Фиг. 5.9. Рингова топология

а. Ринг (Торус), б. Ринг, ангажиран за реализация с къси връзки

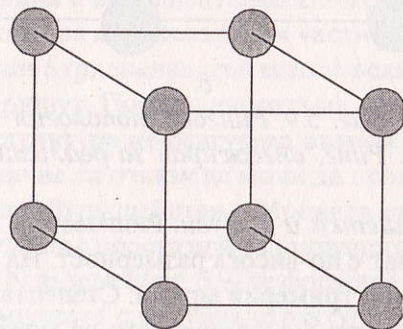
Многомерни решетки и торуси. Ринговете и линейните структури могат да се реализират с по-висока размерност. На фиг. 5.10 са показани примери на двумерни и тримерни мрежи. Степента на комутаторите варира между d и $2d$, където d е измерението на мрежата. Средните възли имат максимална локална степен, докато възлите в ъглите имат минимална степен. Всеки възел се свързва с възли, които се различават от него само по една координата.



а.



б.

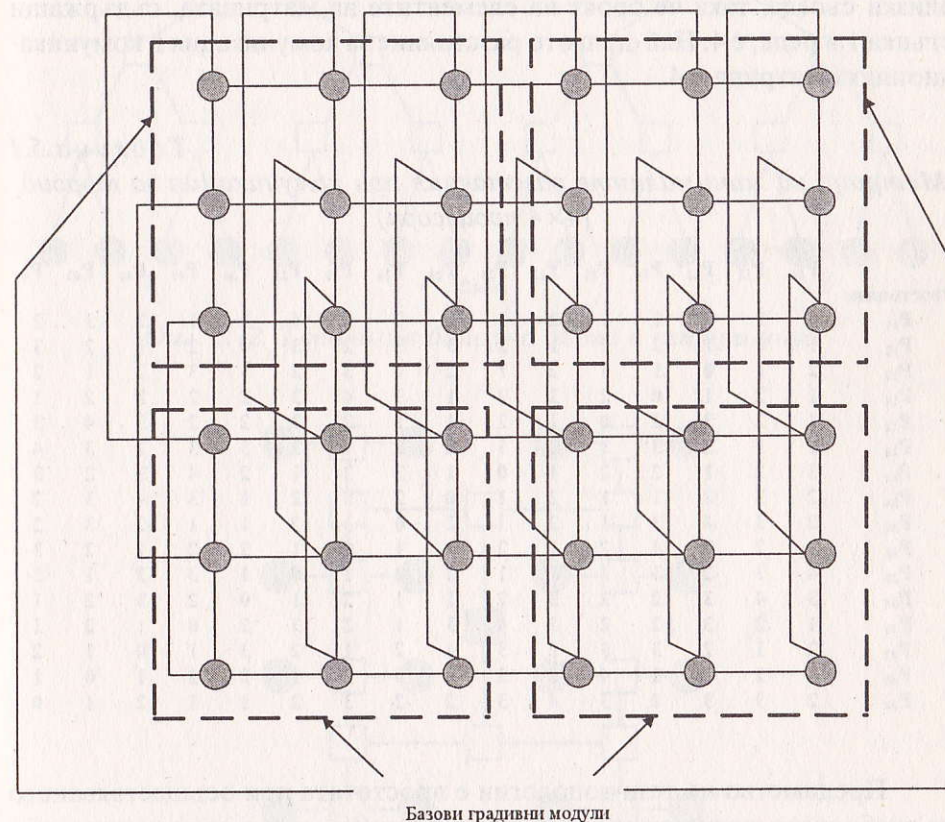


в.

Фиг. 5.10. Решетка, тороид и куб
а. Двумерна решетка, б. Тороид, в. Куб

При d -мерен тороид всеки възел има локална степен $2d$ при двупосочни връзки и е свързан към възлите, различаващи се с 1 в координатата за всяко измерение. Средната комуникационна дистанция е равна на средната дистанция във всяко измерение. Многомерните мрежи се вграждат в планарни структури чрез копиране на базов двумерен слайс и добавяне на допълнителни връзки между слайсовете (фиг. 5.11).

Дървовидни структури. При тях комуникационната дистанция нараства логаритмично с увеличаването на броя на възлите (фиг. 5.12). Възможен маршрут от възел А до възел В е да се достигне до корена на дървото и оттам да се достигне до възел В. Съществува уникален (единствен) маршрут между всяка двойка възли в дървото, като се достигне от



Фиг. 5.11. Композиция на мрежата на модулел принцип с включване на допълнителни връзки между слайсите

единия възел до общ родител и след това до другия възел. Следователно при дървовидните топологии не съществуват алтернативни маршрути.

Означаването на тороида се осъществява, както следва: всеки процесор P_{ij} се означава с два индекса, първият означаващ номера на реда и вторият – номера на колоната на процесорната матрица.

Матрицата на минималните комуникационни разстояния за паралелен компютър с топология тороид (4×4) е представена в табл. 5.1. Комуникационните разстояния се измерват в стъпки на маршрутизация и се отнасят за паралелни системи, включващи до 16 процесора. Очевидно матрицата е симетрична и елементите, включени в главния диагонал, са нулеви. Всеки ред включва минималните комуникационни разстояния до останалите процесори в мултикомпютъра. Всеки процесор има 4 най-близки съседа, така че броят на елементите на матрицата, съдържащи стъпка 1 в реда, е 4. Най-дългото разстояние за комуникация в комуникационната матрица е 4.

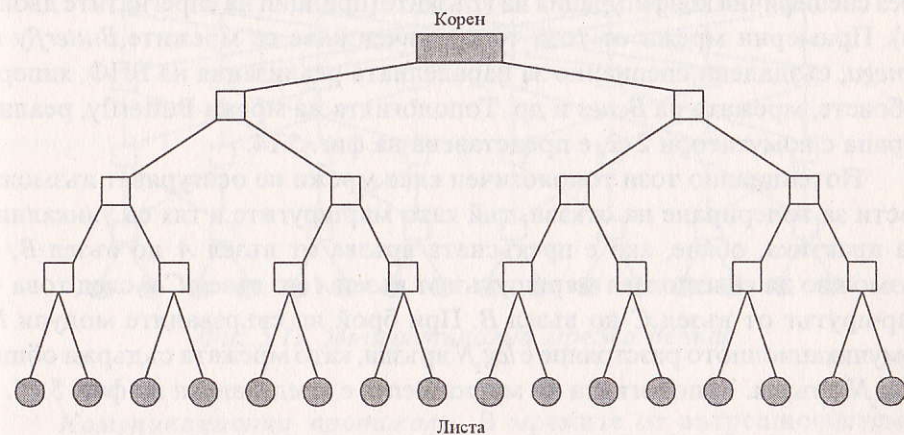
Таблица 5.1

Матрица на минималните разстояния при комуникация за тороид (4×4 процесора)

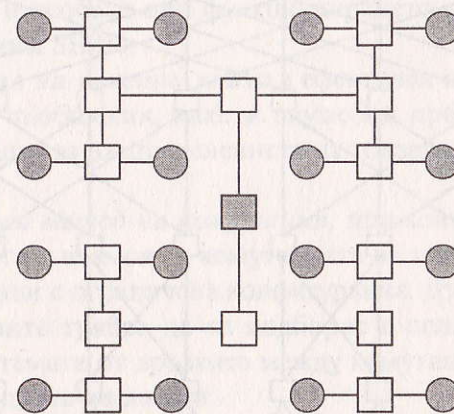
Разстояние	P_{11}	P_{12}	P_{13}	P_{14}	P_{21}	P_{22}	P_{23}	P_{24}	P_{31}	P_{32}	P_{33}	P_{34}	P_{41}	P_{42}	P_{43}	P_{44}
P_{11}	0	1	2	1	1	2	3	2	2	3	4	3	1	2	3	2
P_{12}	1	0	1	2	2	1	2	3	3	2	3	4	2	1	2	3
P_{13}	2	1	0	1	3	2	1	2	4	3	2	3	3	2	1	2
P_{14}	1	2	1	0	2	3	2	1	3	4	3	2	2	3	4	3
P_{21}	1	2	3	2	0	1	2	1	1	2	3	2	2	3	4	3
P_{22}	2	1	2	3	1	0	1	2	2	1	2	3	3	2	3	4
P_{23}	3	2	1	2	2	1	0	1	3	2	1	2	4	3	2	3
P_{24}	2	3	2	1	1	2	1	0	2	3	2	1	3	4	3	2
P_{31}	2	3	4	3	1	2	3	2	0	1	2	1	1	2	3	2
P_{32}	3	2	3	4	2	1	2	3	1	0	1	2	2	1	2	3
P_{33}	4	3	2	3	3	2	1	2	2	1	0	1	3	2	1	2
P_{34}	3	4	3	2	2	3	2	1	1	2	1	0	2	3	2	1
P_{41}	1	2	3	2	2	3	4	3	1	2	3	2	0	1	2	1
P_{42}	2	1	2	3	3	2	3	4	2	1	2	3	1	0	1	2
P_{43}	3	2	1	2	4	3	2	3	3	2	1	2	2	1	0	1
P_{44}	2	3	3	2	3	4	3	2	2	3	2	1	1	2	1	0

Предимство на тези топологии е простотата при осъществяването на глобалната комуникация. Комуникационната мрежа от този тип може да бъде разделена на две изолирани части при прекъсването на една единствена връзка.

Дървовидните структури ефективно се вграждат в планарни структури (фиг. 5.13). При прекъсване на връзка при корена се изолира половината от мрежата. Следователно коренът на дървото е тясно място за надеждността на вътрешносистемната комуникационна мрежа. Пропускателната способност на мрежата варира за различните нива на дървото, като за всяко ниво на родител тя е два пъти по-висока от тази на нивото на децата. Пропускателната способност в рамките на дадено ниво на дървото е постоянна.



Фиг. 5.12. Топология бинарно дърво с четири нива

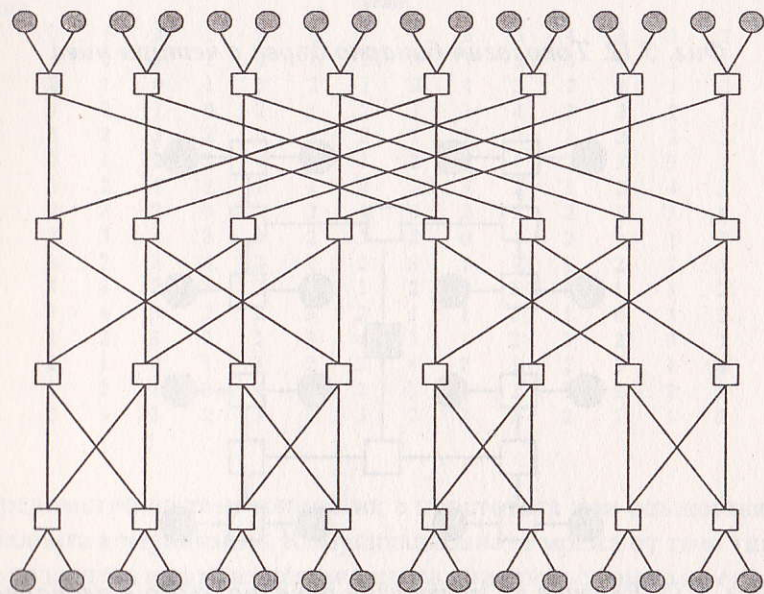


Фиг. 5.13. Влагане на топология бинарно дърво в планарна структура

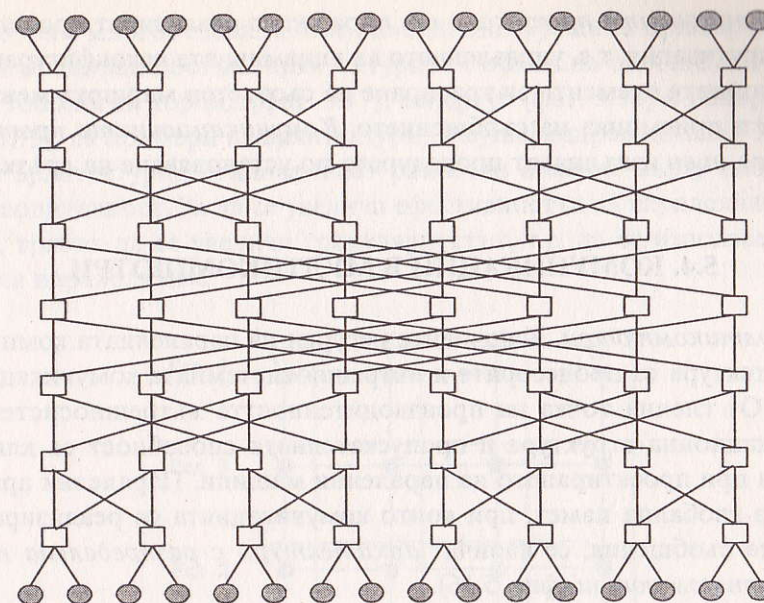
Вътрешносистемни комуникационни мрежи със спрегнати двойки. Тясното място на корена на дървовидните структури може да се преодолее, ако се осигурят множество корени. Това се постига при многостъпалните мрежи със спрегнати двойки.

В топологичен аспект те включват множество припокриващи се дървета. В рамките на всяко стъпало на мрежата комутаторите са разделени на двойки, като всяка двойка комутатори от дадено стъпало се свързва с точно определена двойка комутатори от следващото стъпало чрез специфична конфигурация на връзките (принцип на спрегнатите двойки). Примерни мрежи от този топологичен клас са мрежите *Butterfly* и *Omega*, създадени специално за паралелната реализация на БПФ, хиперкубовете, мрежата на *Benes* и др. Топологията на мрежа *Butterfly*, реализирана с комутатори 2×2 , е представена на фиг. 5.14.

Потенциално този топологичен клас мрежи не осигуряват възможности за толериране на откази, тъй като маршрутите в тях са уникални. На практика, обаче, ако е прекъсната връзка от възел *A* до възел *B*, е възможно да се използва маршрутът от възел *A* до възел *C* и след това – маршрутът от възел *C* до възел *B*. При брой на свързаните модули N комуникационното разстояние е $\log_2 N$ връзки, като мрежата съдържа общо $N \log_2 N$ връзки. Топологията на мрежа *Benes* е представена на фиг. 5.15.



Фиг. 5.14. Мрежова топология *Butterfly*



Фиг. 5.15. Многостъпална мрежа *Benes*

Комуникационни протоколи. В мрежите от вътрешносистемни връзки на паралелните компютърни системи могат да се използват следните методи на комутация:

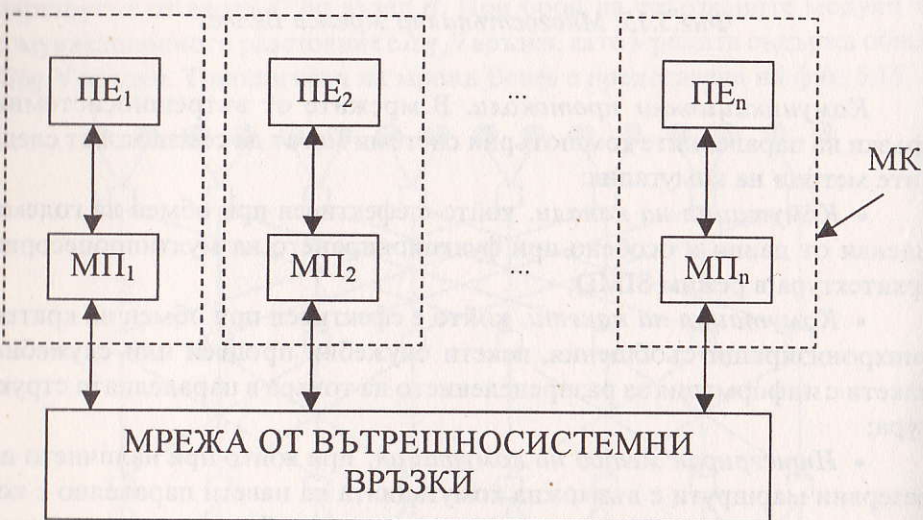
- **Комутация на канали**, който е ефективен при обмен на големи масиви от данни и особено при функционирането на мултипроцесорна архитектура в режим SIMD;
- **Комутация на пакети**, който е ефективен при обмен на кратки синхронизиращи съобщения, пакети служебни процеси или служебни пакети с информация за разпределението на товара в паралелната структура;
- **Интегриран метод на комутация**, при който при наличието на резервни маршрути е възможна комутацията на пакети паралелно с комутацията на канали с ограничена конфигурация. Дължината и конфигурацията на каналите трябва да се подбират с оглед да не са наруши пълнотата на системата от връзките между комутационните възли, реализиращи комутацията на пакети.

Методът на комутация и управляващите функции са вградени в комуникационните възли в съответствие с изискванията за използваните комуникационни протоколи, които могат да се разглеждат на две нива.

Комуникационните протоколи от първо ниво реализират процедурата за маршрутизация, т.е. управлението на динамичната реконфигурация на комутаторните елементи за трасиране на съответен маршрут между източника и приемника на съобщението. Комуникационните протоколи от второ ниво изпълняват процедурата по установяване на връзката.

5.4. КОМУНИКАЦИИ В МУЛТИКОМПЮТРИ

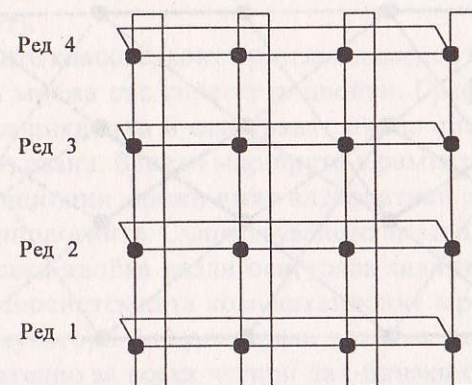
Мултикомпютри. Ключовите ресурси на паралелната компютърна архитектура са процесорите и вътрешносистемната комуникационна мрежа. От гледна точка на производителността вътрешносистемната комуникационна структура и пропускателната способност са ключови фактори при проектирането на паралелни машини. Паралелни архитектури без глобална памет, при които комуникацията се реализира чрез обмен на съобщения, се наричат *архитектури с разпределена памет* или *мултикомпютри* (фиг. 5.16).



Фиг. 5.16. Блокова схема на мултикомпютър

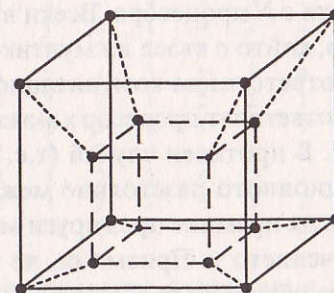
На практика те се считат за паралелни архитектури със средно ниво на паралелизъм. Мултикомпютрите могат да се класифицират като машини с унифициран и неунифициран достъп, при което достъпът се отнася за разстоянието между произволна двойка процесори. Паралелни

компютри с многостъпални комуникационни мрежи са пример на машини с унифициран достъп. Архитектурите с обмен на съобщения, основани на топологии тороид (фиг. 5.17), хиперкуб (фиг. 5.18) и рециркуляция (фиг. 5.19), са примери на архитектури с неунифициран достъп. Тези различни архитектурни стилове имат различно влияние върху системната производителност. За да се увеличи ефективността на паралелната обработка, трябва да се увеличи грануляритетът, т.е. да се използва средно ниво на паралелизъм.



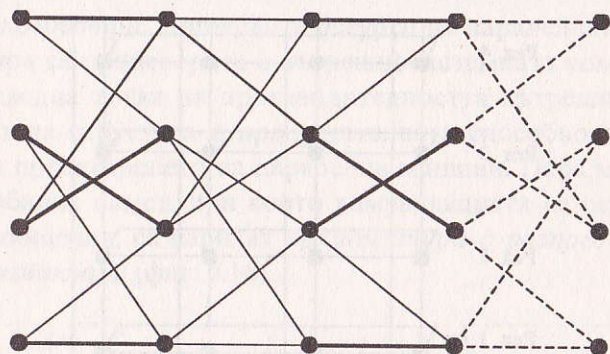
Фиг. 5.17. Граф – теоретичен модел на тороид (16 процесора)

Като се вземе предвид фактът, че всички аспекти на координатата на процесите (информационен обмен, синхронизация и взаимно изключване) при мултикомпютрите се реализират чрез комуникация, очевидно е, че комуникацията между процесорите е основата на всички приложения, изпълнявани на паралелни компютри. Оптималното разпръскване е



Фиг. 5.18. Граф – теоретичен модел на хиперкуб (16 процесора)

един от главните проблеми на комуникацията на мултикомпютри с разпределена памет. При разпръскването един процесор комуникира едно и също съобщение към всички останали. Този процесор се нарича източник на разпръскването. Зареждането на един и същи програмен код от фронт-енд процесора във всички процесори е типичен пример на разпръскване. Режим, при който един процесор може да комуникира едновременно през всичките си портове се нарича режим по „n-порта“ или режим „крещене“.



Фиг. 5.19. Граф – теоретичен модел на мрежа с рециркуляция (16 процесора)

Основен проблем е да се изследва влиянието на системната комуникационна топология върху глобалната системна производителност на най-разпространените мултикомпютри в случаите на интензивна комуникация като разпръскване в режим „n-порта“.

Схеми на комуникациите. Проектирането на комуникационните модели се основава на прякото съответствие между графите $G(V,E)$ от ед N и мултикомпютрите с N процесора. Всеки възел на множеството V съответства на процесор, който е възел на мултикомпютъра и всяко ребро на множеството E съответства на комуникационна връзка. Ако съществува ребро $(x,y) \in E$, съответният процесор x може да изпрати съобщение директно до процесор y . В противен случай (т.е. $d(x,y) > 1$, където $d(x,y)$ представлява комуникационното разстояние между процесорите x и y), съобщението от x трябва да премине през други междинни процесори, за да достигне местоназначението y . Приема се, че комуникацията се реализира чрез процедури за обмен на съобщения.

Типично, комуникационните връзки са двупосочни и се използват в режим на предаване пълен дуплекс. Следователно ние моделираме мултикомпютъра със симетричен насочен граф $G^*(V,E)$, където всяко ребро на $E(G)$ съответства на две симетрични ребра от $E(G^*)$. При повечето от съществуващите мултикомпютри процесорите могат едновременно да комуникират през всичките си портове с най-близките си съседи. Този начин на комуникация се нарича „режим на крещене“ (или режим по n -порта) в противовес на режима „шепот“, при който всеки процесор може да използва само един порт в даден момент от време (нарича се също режим с един порт).

Топологичните класове, които разглеждаме, са тороид, хиперкуб и рециркуляционна мрежа със спрегнати двойки. Графите, представящи тези топологии са циклични и осигуряват пълна система на връзките, т.е. те са силно свързани. Всички маршрути в рамките на вътрешносистемните комуникационни мрежи имат алтернативи поради цикличната структура на топологиите. Съществуването на множество несвързани пътища между всяка двойка възли осигурява значителна отказоустойчивост на вътрешносистемната комуникационна мрежа. Техническото изпълнение с комутатори 2×2 осигурява локална степен 4 за всеки от графите. Следователно за всеки четири най-близки съседни процесори съществува минимално разстояние за комуникация от една стъпка на маршрутизация.

	P ₁₁	P ₁₂	P ₁₃	P ₁₄	P ₂₁	P ₂₂	P ₂₃	P ₂₄	P ₃₁	P ₃₂	P ₃₃	P ₃₄	P ₄₁	P ₄₂	P ₄₃	P ₄₄
Разстояние																
P ₁₁	0	1	2	1	1	2	3	2	2	3	4	3	1	2	3	2
P ₁₂	1	0	1	2	2	1	2	3	3	2	3	4	2	1	2	3
P ₁₃	2	1	0	1	3	2	1	2	4	3	2	3	3	2	1	2
P ₁₄	1	2	1	0	2	3	2	1	3	4	3	2	2	3	2	1
P ₂₁	1	2	3	2	0	1	2	1	1	2	3	2	2	3	4	3
P ₂₂	2	1	2	3	1	0	1	2	2	1	2	3	3	2	3	4
P ₂₃	3	2	1	2	2	1	0	1	3	2	1	2	4	3	2	3
P ₂₄	2	3	2	1	1	2	1	0	2	3	2	1	3	4	3	2
P ₃₁	2	3	4	3	1	2	3	2	0	1	2	1	1	2	3	2
P ₃₂	3	2	3	4	2	1	2	3	1	0	1	2	2	1	2	3
P ₃₃	4	3	2	3	3	2	1	2	2	1	0	1	3	2	1	2
P ₃₄	3	4	3	2	2	3	2	1	1	2	1	0	2	3	2	1
P ₄₁	1	2	3	2	2	3	4	3	1	2	3	2	0	1	2	1
P ₄₂	2	1	2	3	3	2	3	4	2	1	2	3	1	0	1	2
P ₄₃	3	2	1	2	4	3	2	3	3	2	1	2	2	1	0	1
P ₄₄	2	3	3	2	3	4	3	2	2	3	2	1	1	2	1	0

а. Матрица на минималните разстояния при комуникация за тороид (4x4 процесора)

Разстояние	P ₁₁	P ₁₂	P ₁₃	P ₁₄	P ₁₅	P ₁₆	P ₁₇	P ₁₈	P ₂₁	P ₂₂	P ₂₃	P ₂₄	P ₂₅	P ₂₆	P ₂₇	P ₂₈
P ₁₁	0	1	2	1	1	2	3	2	1	2	3	2	2	3	4	3
P ₁₂	1	0	1	2	2	1	2	3	2	1	2	3	3	2	3	4
P ₁₃	2	1	0	1	3	2	1	2	3	2	1	2	4	3	2	3
P ₁₄	1	2	1	0	2	3	2	1	2	3	2	1	3	4	3	2
P ₁₅	1	2	3	2	0	1	2	1	2	3	4	3	1	2	3	2
P ₁₆	2	1	2	3	1	0	1	2	3	2	3	4	2	1	2	3
P ₁₇	3	2	1	2	2	1	0	1	4	3	2	3	3	2	1	2
P ₁₈	2	3	2	1	1	2	1	0	3	4	3	2	2	3	2	1
P ₂₁	1	2	3	2	2	3	4	3	0	1	2	1	1	2	3	2
P ₂₂	2	1	2	3	3	2	3	4	1	0	1	2	2	1	2	3
P ₂₃	3	2	1	2	4	3	2	3	2	1	0	1	3	2	1	2
P ₂₄	2	3	2	1	3	4	3	2	1	2	1	0	2	3	2	1
P ₂₅	2	3	4	3	1	2	3	2	1	2	3	2	0	1	2	1
P ₂₆	3	2	3	4	2	1	2	3	2	1	2	3	1	0	1	2
P ₂₇	4	3	2	3	3	2	1	2	3	1	2	2	2	1	0	1
P ₂₈	3	4	3	2	2	3	2	1	2	3	2	1	1	2	1	0

б. Матрица на минималните разстояния при комуникация за хиперкуб (16 процесора)

Разстояние	P ₁₁	P ₁₂	P ₁₃	P ₁₄	P ₂₁	P ₂₂	P ₂₃	P ₂₄	P ₃₁	P ₃₂	P ₃₃	P ₃₄	P ₄₁	P ₄₂	P ₄₃	P ₄₄
P ₁₁	0	1	2	1	4	1	2	1	2	3	2	3	4	3	2	3
P ₁₂	1	0	1	2	3	2	1	2	1	2	3	2	3	4	3	4
P ₁₃	2	1	0	1	2	3	2	3	2	1	4	1	2	3	4	3
P ₁₄	1	2	1	0	3	2	1	2	1	2	3	2	3	4	3	4
P ₂₁	4	3	2	3	0	3	2	3	4	1	2	1	2	1	2	1
P ₂₂	1	2	3	2	3	0	3	2	1	4	1	4	3	2	1	2
P ₂₃	2	1	2	1	2	3	0	3	2	1	4	1	2	3	4	3
P ₂₄	1	2	3	2	3	2	3	0	1	4	1	4	3	2	1	2
P ₃₁	2	1	2	1	4	1	2	1	0	3	2	3	4	3	2	3
P ₃₂	3	2	1	2	1	4	1	4	3	0	3	2	1	2	3	2
P ₃₃	2	3	4	3	2	1	4	1	2	3	0	3	2	1	2	1
P ₃₄	3	2	1	2	1	4	1	4	3	2	3	0	1	2	3	2
P ₄₁	4	3	2	3	2	3	2	3	4	1	2	1	0	1	2	1
P ₄₂	3	4	3	4	1	2	3	2	3	2	1	2	1	0	1	2
P ₄₃	2	3	4	3	2	1	4	1	2	3	2	3	2	1	0	1
P ₄₄	3	4	3	4	1	2	3	2	3	2	1	2	1	2	1	0

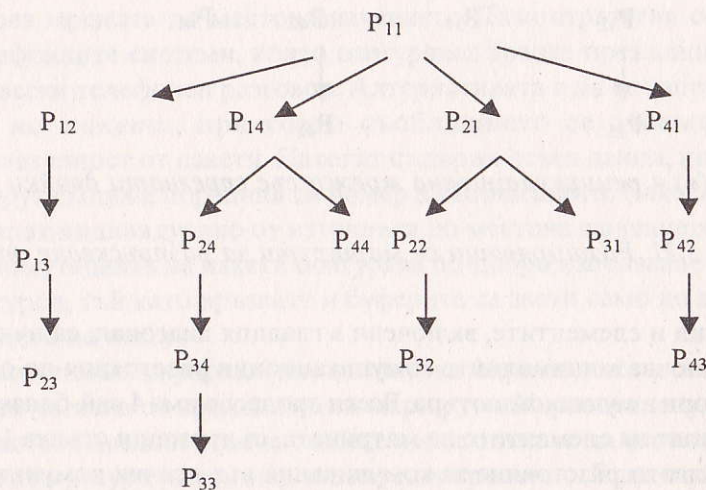
в. Матрица на минималните разстояния при комуникация за рециркуляционна мрежа със спрегнати двойки (16 процесора)

Фиг. 5.20. Матрици на минималните разстояния за комуникация на популярните мултикомпютри

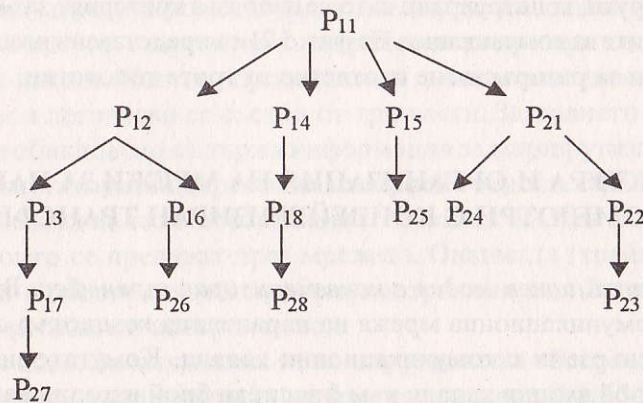
Означаването на тороида се осъществява, както следва: всеки процесор P_{ij} се означава с два индекса, първият означаващ номера на реда и вторият – номера на колоната на процесорната матрица. Процесорите на 4-мерния хиперкуб също се означават с два индекса P_{ij} : първият индекс се отнася за номера на куба, а вторият е последователният номер на

процесора в рамките на този куб. Комуникационните връзки се установяват, като се подчиняват на следното правило: процесор P_{ij} се свързва с процесор P_{2j} . Означаването на рециркуляционната мрежа се подчинява на правилата, приложени към тороида.

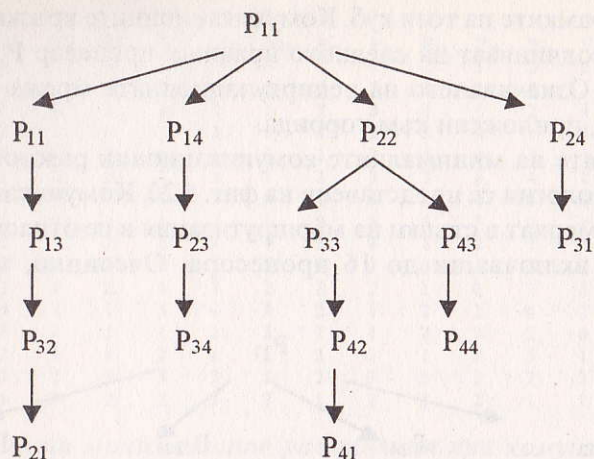
Матриците на минималните комуникационни разстояния за трите циклични топологии са представени на фиг. 5.20. Комуникационните разстояния се измерват в стъпки на маршрутизация и се отнасят за паралелни системи, включващи до 16 процесора. Очевидно, матриците са



(а) в тороид



(б) в хиперкуб



(в) в рециркуляционна мрежа със спрегнати двойки

Фиг. 5.21. Разширяващи се маршрути за разпръскване от P_{11}

симетрични и елементите, включени в главния диагонал, са нулеви. Всеки ред включва минималните комуникационни разстояния до останалите процесори в мултикомпютъра. Всеки процесор има 4 най-близки съседа, така че броят на елементите на матрицата, съдържащи стъпка 1 в реда, е 4. Най-дългото разстояние за комуникация във всички комуникационни матрици е 4.

Разпръскването в режим на „крещение“ се осъществява по разширяващи се маршрути, конструирани като се използва критерият за минимизиране на разходите за комуникация. На фиг. 5.21 са представени разширяващите се маршрути за разпръскване съответно за трите топологии.

5.5. СТРУКТУРА И ОРГАНИЗАЦИЯ НА МРЕЖИ ЗА ПАРАЛЕЛНИ КОМПЮТРИ С КОНВЕЙЕРИЗИРАН ТРАНСФЕР

Комуникационен модел с конвейеризиран трансфер. Вътрешносистемната комуникационна мрежа на паралелния компютър съдържа комутатори, свързани с комуникационни канали. Комутаторите свързват фиксиран брой входни канали към фиксиран брой изходни канали. Броят на комутираните входни (изходни) канали се нарича степен на комутатора. При директните мрежи към всеки комутатор е свързан процесор Про-

цесорите генерират и терминират трафика, докато комутаторите само придвижват трафика.

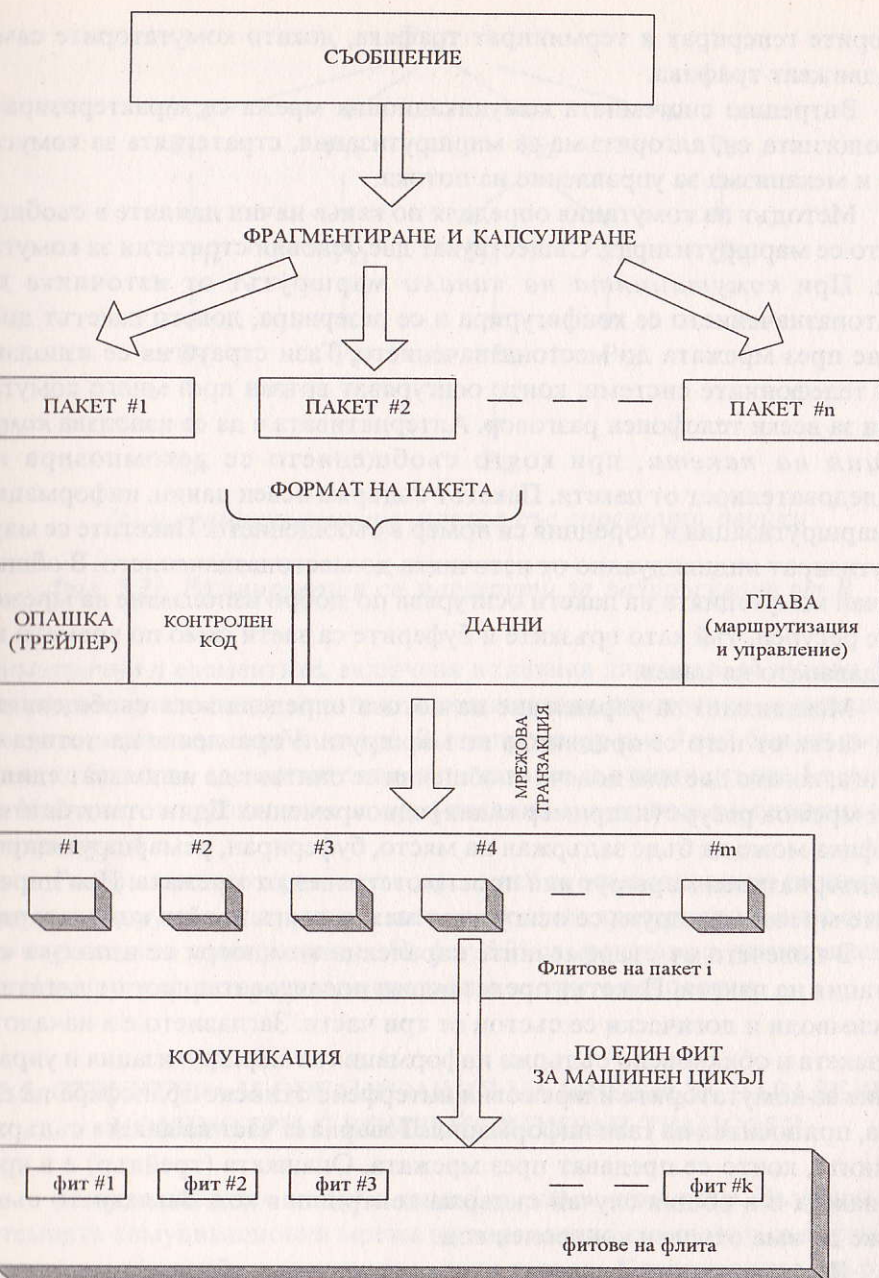
Вътрешно системната комуникационна мрежа се характеризира с топологията си, алгоритъма за маршрутизация, стратегията за комутация и механизма за управление на потока.

Методът на комутация определя по какъв начин данните в съобщението се маршрутизират. Съществуват две основни стратегии за комутация. При *комутацията на канали* маршрутът от източника до местоназначението се конфигурира и се резервира, докато пакетът достигне през мрежата до местоназначението. Тази стратегия се използва при телефонните системи, които осигуряват връзка през много комутатори за всеки телефонен разговор. Алтернативата е да се използва *комутация на пакети*, при която съобщението се декомпозира на последователност от пакети. Пакетът съдържа освен данни, информация за маршрутизация и поредния си номер в съобщението. Пакетите се маршрутизират индивидуално от източника до местоназначението. В общия случай комутацията на пакети осигурява по-добро използване на мрежовите ресурси, тъй като връзките и буферите са заети само по времето на предаването на пакет.

Механизмът за управление на потока определя кога съобщението или части от него се придвижва по маршрута. Управление на потока се налага, когато две или повече съобщения се опитват да използват един и същ мрежов ресурс (например канал) едновременно. Един от потоците в трафика може да бъде задържан на място, буфериран, ремаршрутизиран по алтернативен маршрут или просто отстранен от мрежата. При директните мрежи маршрути се осигуряват между всяка двойка комутатори.

В повечето от съвременните паралелни компютри се използва комутация на пакети. Пакетът представлява последователност от дигитални символи и логически се състои от три части. Заглавието е в началото на пакета и обикновено съдържа информация за маршрутизация и управление за комутаторите и мрежовия интерфейс относно трансфера на пакета, приносител на тази информация. Товарната част на пакета съдържа данните, които се предават през мрежата. Опащката (трейлър) е в края на пакета и в общия случай съдържа контролния код. Заглавието също може да има отделен контролен код.

Двата основни механизма за конструиране на абстракции в контекста на мрежите са капсулирането и фрагментацията (фиг. 5.22). Капсулирането предполага съдържание на информация за протокола от по-високо ниво в неинтерпретирана форма във формата на пакета за дадено ниво.



Фиг. 5.22. Фрагментиране и капсулиране на информацията при комуникация в мрежи за паралелни компютри

Фрагментацията предполага декомпозирането на информацията за протокола от по-високо ниво в последователност от съобщения на дадено ниво. Заглавието и опашката на пакета формират плик, който се интерпретира от комутаторите и капсулира данните. Информацията, свързана с протокола на нивото на текущия възел, се съдържа в данните. Например заявка за четене типично се предава на контролера на паметта в един пакет и отговорът на съответния ред в кеша също се съдържа в един пакет. Контролерът на паметта не се интересува от маршрута на пакета или от формата на заглавието или на опашката. В същото време мрежата не се интересува от формата на заявката за дистанционно четене в полето за данни. Голям масив от данни не може да се предаде с един пакет. Той се фрагментира на няколко пакета. Всеки пакет трябва да съдържа информация, която да указва мястото, където данните трябва да бъдат предадени или кой фрагмент от последователността от данни се съдържа в пакета.

Комутацията на канали традиционно се използва в телекомуникационните мрежи поради факта, че времето за установяване на връзката е кратко в сравнение с времетраенето на телефонния разговор. Този метод на комутация се използва при малка част от паралелните компютри. Важна разлика при паралелните машини е, че каналът се конфигурира чрез маршрутизирането на съобщението през мрежата, като маршрутът се резервира отворен за времетраенето на обмена. Традиционният подход е да се определи маршрутът при източника, след което се конфигурират комутаторите и после се предава информация по канала.

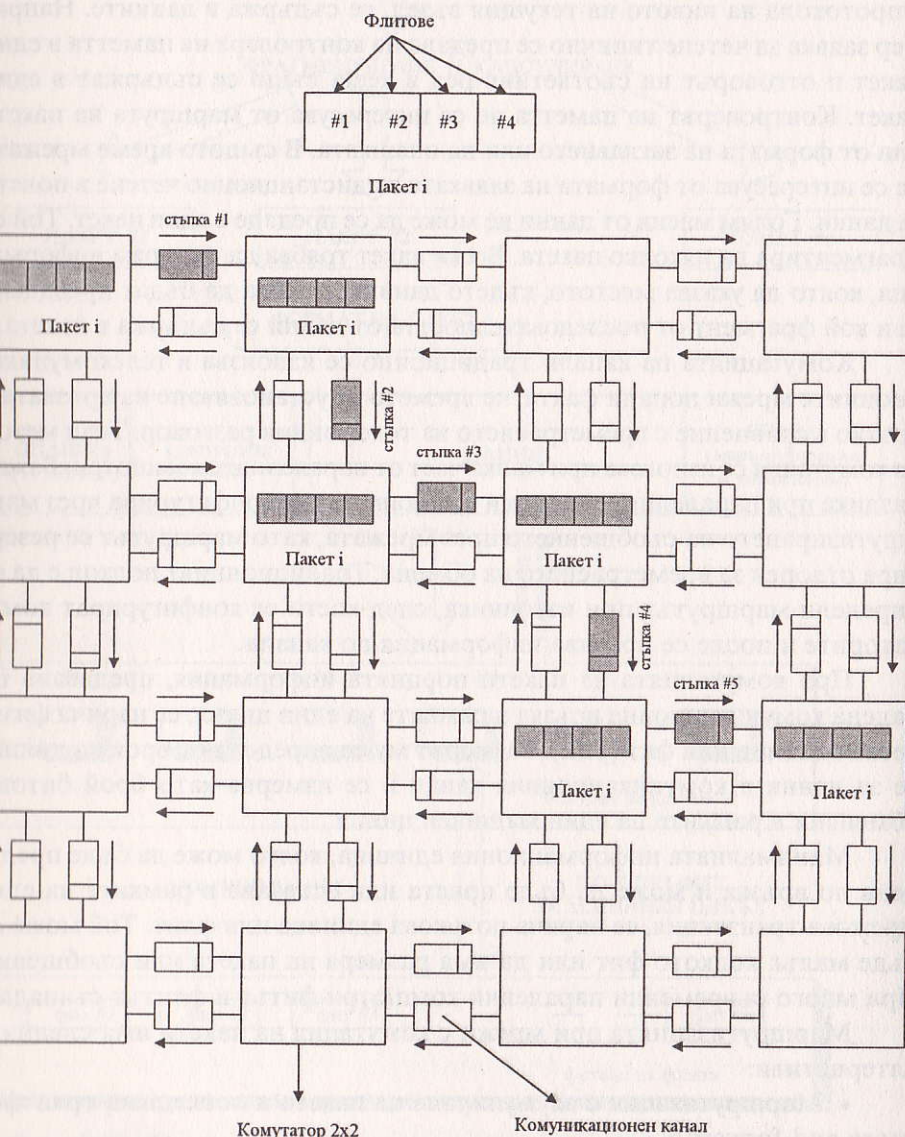
При комутацията на пакети порцията информация, предавана по дадена комуникационна връзка в рамките на един цикъл, се нарича физическа единица или фит (phit). Размерът му се определя от броя на линиите за данни в комуникационния канал и се измерва като брой битове, обменяни в рамките на един машинен цикъл.

Минималната информационна единица, която може да бъде предадена по връзка и може да бъде приета или отказана в рамките на една мрежова транзакция, се нарича потокова единица или флит. Той може да бъде малък колкото фит или да има размера на пакета или съобщение. При много съвременни паралелни компютри фитът и флитът съвпадат.

Маршрутизацията при мрежи с комутация на пакети има следните алтернативи:

- Маршрутизация с акумулиране на пакета и последващ трансфер (store-and-forward);
- Маршрутизация с конвейеризиран трансфер на пакета (cut-through routing).

На фиг. 5.23 е показан моделът на комуникация при мрежи с трансфер „акумулиране и трансфер“. При него пакетът се акумулира в комутатора до приемането на всичките му флитове, като едва след това той се предава към следващия комутатор по маршрута към дестинацията.



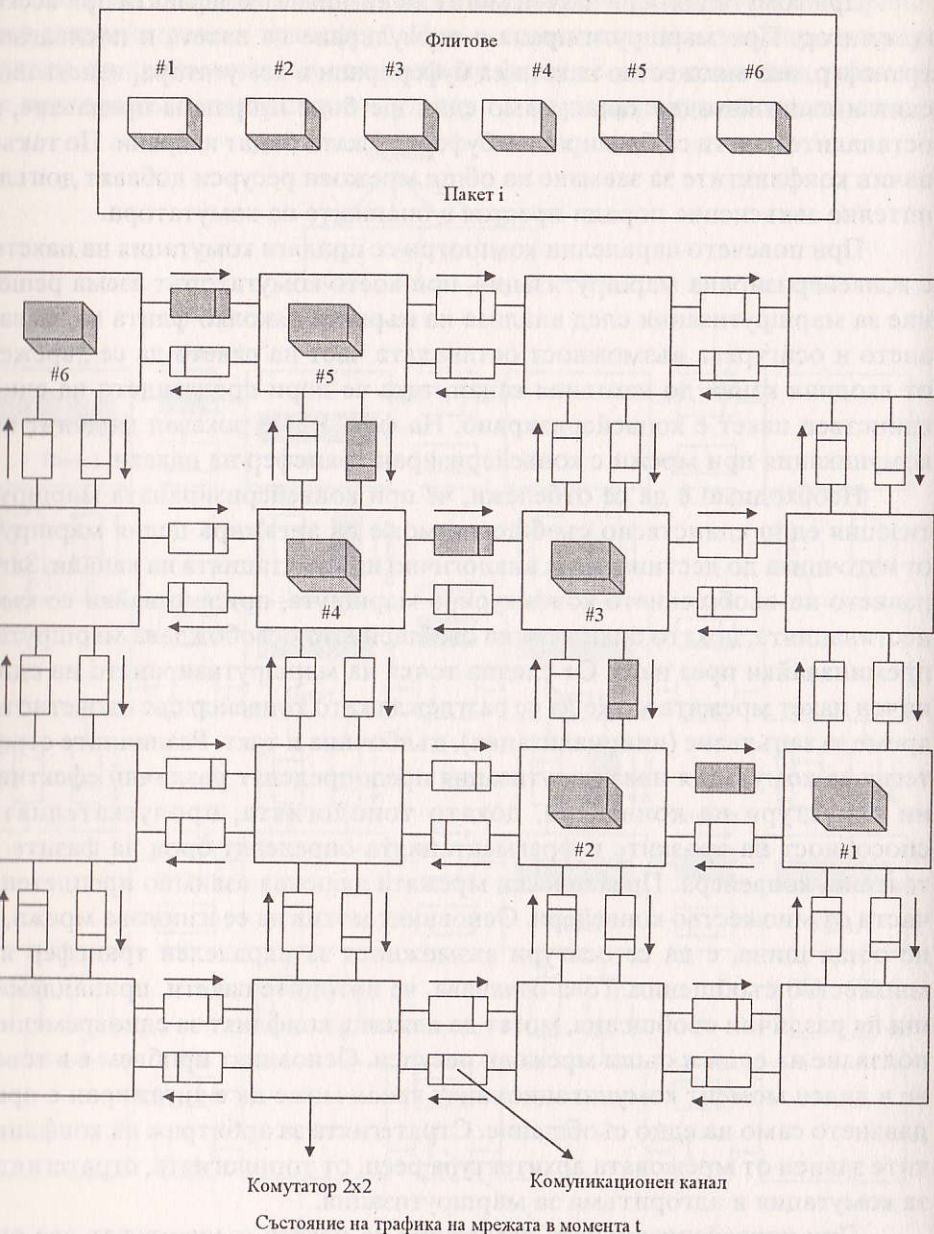
Фиг. 5.23. Маршрутизация с акумулиране на пакета и последващ трансфер (store-and-forward)

При комутацията на пакети могат да възникнат конфликти при всеки комутатор. При маршрутизиране с акумулиране на пакета и последващ трансфер, ако множество пакети са буферирани в комутатора, изискващи един и същи изходен канал, само един ще бъде избран за предаване, а останалите пакети са блокирани в буфера, докато бъдат избрани. По такъв начин конфликтите за заемане на общи мрежови ресурси добавят допълнително закъснение поради престоя в опашките на комутатора.

При повечето паралелни компютри се прилага комутация на пакети с конвейеризирана маршрутизация, при което комутаторът взема решение за маршрутизация след анализа на първите няколко флита на заглавието и осигурява възможност останалата част на пакета да се „вреже“ от входния канал до изходния канал, така че дори предаването на един единствен пакет е конвейеризирано. На фиг. 5.24 е показан моделът на комуникация при мрежи с конвейеризиран трансфер на пакети.

Необходимо е да се отбележи, че при конвейеризираната маршрутизация едно единствено съобщение може да ангажира целия маршрут от източника до дестинацията аналогично на комутацията на канали. Заглавието на съобщението конфигурира маршрута, придвижвайки се към дестинацията, докато опашката на съобщението освобождава маршрута, преминавайки през него. От гледна точка на маршрутизирането на единичен пакет мрежата може да се разглежда като конвейер със съответното време за запълване (инициализация), дълбочина и такт. Различните стратегии за комутация и маршрутизация предопределят различни ефективни структури на конвейера, докато топологията, пропускателната способност на връзките и фрагментацията определят броя на фазите и такта на конвейера. Практически мрежата включва взаимно преплетени части от множество конвейери. Основният мотив да се използва мрежа, а не обща шина, е да се осигури възможност за паралелен трансфер на множество съобщения. Това означава, че потоците пакети, принадлежащи на различни съобщения, могат да влязат в конфликт за едновременно ползване на едни и същи мрежови ресурси. Основният проблем е в това, че в даден момент комуникационният канал може да е ангажиран с предаването само на едно съобщение. Стратегията за арбитраж на конфликтите зависи от мрежовата архитектура респ. от топологията, стратегията за комутация и алгоритъма за маршрутизация.

При конвейеризираната комутация на пакети съществуват две опции за блокиране. При виртуалната конвейеризация се прави спулинг (spooling) на блокирания идващ пакет в буфер, така че положението се влошава в сравнение с акумулативната маршрутизация. При конвейери-



Фиг. 5.24. Маршрутизация с конвейеризиран трансфер на пакета (cut-through routing) от вида „дупка на червея“ (wormhole)

зираната маршрутизация от тип „дупка на червея“ (wormhole) се буферизира само няколко флита на пакета, докато опашката на пакета остава по продължение на маршрута. Блокираната част на пакета наподобява канал, който се поддържа отворен в мрежата.

При вътрешносистемните мрежи на паралелните компютри пакет, който е маршрутизиран към пълен буфер, се блокира, а не се изхвърля от трафика. Това налага „ръкостискане“ (handshake) между изходния и входния порт към връзката, т.е. управление на потока на ниво връзка. При продължително задръстване трафикът „се връща“ от точката в мрежата, в която е възникнал конфликтът за едновременно ползване на общ ресурс, към източниците, които насочват трафика към конфликтната точка.

Тогава тези източници са подложени на „обратно налягане“ от мрежата (когато тя отказва да приеме пакетите) предизвиквайки намаляване на потокната скорост в мрежата, която се определя от потокната скорост през задръстения маршрут. Увеличаването на капацитета на буферите в мрежата дава възможност задръстването да бъде толерирано по-дълго време, без да оказва обратно налягане на източника. От друга страна обаче се увеличава латентността поради потенциалното нарастване на времето за блокиране на пакета в опашките.

Важен проблем, който предизвиква блокирането на съобщения в мрежата, е че обратното налягане може да повлияе на трафика, който не е маршрутизиран за претоварения изходен порт. Нека да разгледаме случая, в който трафикът към дадена дестинация е много интензивен поради факта, че то съдържа често използвана глобална променлива. Този претоварен възел се нарича „гореща точка“. В случаите, когато трафикът, насочен към тази дестинация, превишава пропускателната способност на горещия изходен порт, възниква обратно налягане в мрежата. Ако тази ситуация се задържи по-дълго време, задръстването се разпространява обратно през дървото канали, свързани с проблемната дестинация. Явлението се нарича „дървовидно насищане“. Всеки трафик, пресичащ това дърво, ще бъде забавен. За решаването на този проблем при мрежи с маршрутизация по „дупката на червея“ вместо да се блокира, проблемното съобщение се унищожавя, като задължително се информира източника на колизията.

Възможно е източникът да задържи опашката на съобщението, докато заглавието достигне дестинацията, т.е., докато каналът се конфигурира. Ако по маршрута възникне колизия, „червеят се свива“ обратно до източника. Този подход редуцира до голяма степен влиянието на дървовидното насищане.

Следователно всички аспекти на мрежовия дизайн – пропускателна способност на каналите, топологията, стратегията за комутация, алгоритмът за маршрутизация и управление на потока взаимно определят латентността на съобщението. В случаите, когато пропускателната способност, изисквана от програмата, е ниска в сравнение с максималната пропускателна способност на мрежата, колизиите ще бъдат малко и ще има тенденция буферите да бъдат празни като латентността ще се задържа устойчиво ниска особено при конвейеризираното маршрутизиране. С повишаване на изискванията на програмите към мрежовата пропускателна способност латентността расте поради зачестяване на конфликтите.

Маршрутизация. Основните предимства на ефективния алгоритъм за маршрутизация е генерирането на множество маршрути без рискове за твърдо блокиране, поддържането на ниска латентност при обслужване на заявките, осигуряването на равномерно разпределяне на комуникационния товар и възможности за самовъзстановяване при откази в мрежата.

Основната операция, която изпълнява комутаторът, е за всеки пристигащ пакет на входните портове да избира изходен порт, през който да го изпрати, т.е. в зависимост от дестинацията на пакета да определи следващия комуникационен канал по маршрута. Високоскоростните комутатори използват три механизма за селекция на изходния канал на основата на информацията, съдържаща се в заглавието на пакета: аритметични операции, селекция и таблици. При паралелните компютри комутаторът трябва да взема решения за маршрутизация за всичките си входове на интервали от по няколко цикъла, така че механизмите за маршрутизация трябва да бъдат прости и бързи.

При повечето регулярни топологии прости аритметични операции са достатъчни за селекция на съответния изходен порт. При двумерните решетки всеки пакет носи дистанцията за комуникация за всяка посока $[D_x, D_y]$ със съответния знак в заглавието на пакета. Операцията по маршрутизация в комутатор ij се дефинира по следния начин:

Посока	Условие
WEST (-x)	$\Delta x < 0$
EAST (+x)	$\Delta x > 0$
SOUTH (-y)	$\Delta x = 0, \Delta y < 0$
NORTH (+y)	$\Delta x = 0, \Delta y > 0$
HOST	$\Delta x = 0, \Delta y = 0$

За осъществяването на този тип маршрутизация комутаторът трябва да тества адреса в заглавието и да инкрементира или декрементира съответно едно поле за маршрутизация. В типичния случай маршрутите в двумерна решетка се формират чрез придвижване първо в посока D_x и след това в посока D_y .

По-общ подход представлява задаването на маршрута от подателя на съобщението, при който той включва в заглавието на пакетите номера на изходния порт на всеки комутатор по маршрута $p_0, p_1, \dots, p_{h-1}$. Всеки комутатор прочита номера на изходния порт в началото на съобщението и го премахва от заглавието, като комутира съобщението през зададения канал по-нататък в мрежата. Този подход осигурява проста структура на комутатора с малко управляващи функции, като дори не се изискват аритметични устройства за поддържане на сложни функции по маршрутизация за произволни топологии. В случая цялата интелигентност се съсредоточава в хост-възлите. Недостатък на разглеждания подход е голямата дължина на заглавието, която варира в процеса на придвижването на пакета в мрежата. Ако локалната степен на комутатора е d и максималната дължина на маршрута е h , заглавието на пакета трябва да съдържа $h \log d$ бита за маршрутизация. Този подход се използва в паралелния компютър Meiko CS-2.

Третият подход за маршрутизация на основата на таблици е универсален и предполага малко заглавие с фиксиран размер. Всеки комутатор съдържа маршрутизираща таблица и всеки пакет съдържа в заглавието си поле за маршрутизация. Определянето на съответния изходен порт става чрез търсене на позиция $R(i)$ в таблицата за маршрутизация, зададена в полето за маршрутизация. Този подход се използва в комутаторите ATM (Asynchronous Transfer Mode). В общия случай в позицията на таблицата се намира и полето за маршрутизация за следващата стъпка на маршрута. Недостатък на този подход е, че е необходимо да се обменят служебни съобщения между комутаторите или друг механизъм, който да поддържа актуалното съдържание на таблицата. Значително големи таблици са необходими за поддържането дори на прости алгоритми за маршрутизация. Този подход се използва главно при компютърните мрежи (глобални и локални), при които в даден интервал от време се използват само някои от маршрутите между възлите за разлика от паралелните компютри, при които в общия случай съществува трафик между всички възли.

Алгоритмът за маршрутизация е *детерминиран (неадаптивен)*, ако маршрутът на съобщението се определя единствено от взаимното разпо-

ожение на подателя и дестинацията независимо от трафика в мрежата. пакетът ще продължи да следва маршрута независимо от това дали по го има блокирана връзка. При *адаптивните* алгоритми за маршрутизация трафикът на мрежата оказва влияние върху маршрута на пакета. Например в решетка маршрутът може да следва зиг-заг по пътя си към дестинацията, заобикаляйки блокирани или отказали връзки. Ако алгоритъмът за маршрутизация избира най-кратките маршрути до дестинацията, той се нарича *минимален*. В противен случай той е *неминимален*. Наличието на алтернативни маршрути от източника до местоназначение е необходимо условие за адаптивната маршрутизация и толерантност към откази, като също така осигурява възможности за разпределянето на комуникационния товар между повече връзки. Изборът на конкретен маршрут се прави динамично, след като пакетът се присъедини към трафика в мрежата. При адаптивната маршрутизация структурата на комутаторите е по-висока сложност, но тя осигурява потенциалната възможност за по-добро използване на комуникационните канали.

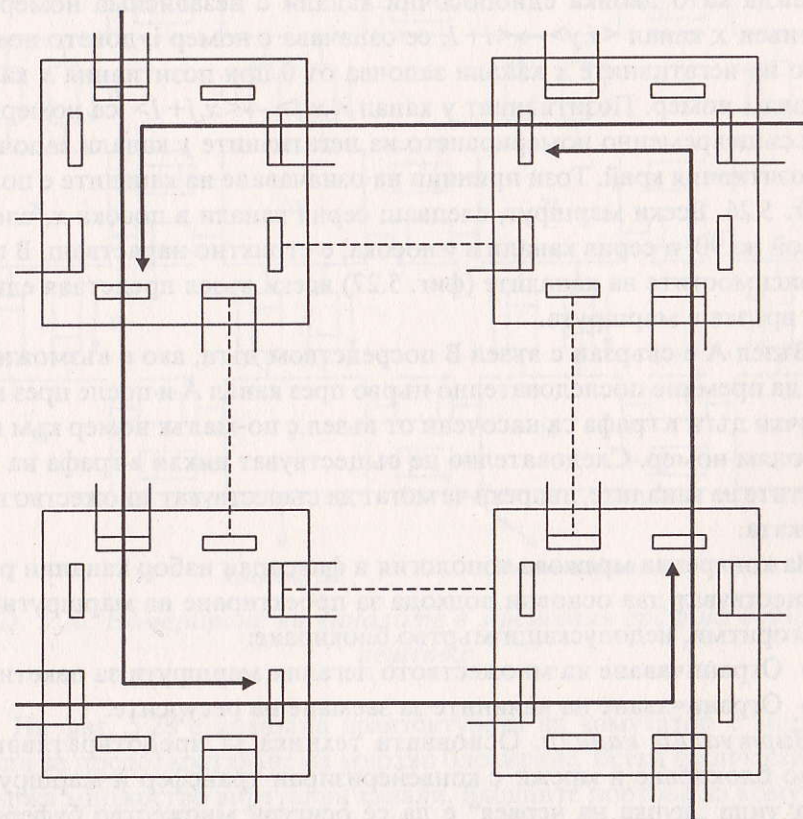
Мъртво блокиране възниква в случаите, когато пакетът чака събиене, което не може да възникне, например ако никой от пакетите не може да напредне към дестинацията си, тъй като опашките на мрежата са пълни и пакетите взаимно се изчакват да освободят комуникационните ресурси.

Мъртво блокиране може да възникне в различни ситуации. Например в случаите, когато два възела си изпращат взаимно пакети и всеки от тях започва да предава преди никой от тях да започне да приема. При мрежите тази ситуация може да възникне при полудуплексни канали или когато управляващото устройство на комутатора не е успяло едновременно да осъществи приемане и предаване по дуплексен канал. Надеждната мрежа може да бъде защитена от мъртво блокиране, ако всичките ѝ възли са в състояние да отстраняват пакети от мрежата, дори когато не могат да изпращат пакети. Мъртво блокиране може да възникне, когато няколко пакета заявяват заемане на мрежови ресурси (фиг. 5.25).

На фиг. 5.25 комутаторите имат по 4 входни и 4 изходни порта. Всеки от четирите пакета държи входен порт, изходен порт и входен порт и се стреми да заеме следващия изходен порт. Никой от тях няма да освободи изходния буфер, докато не се придвижи напред, така че никой от тях няма да напредне в мрежата.

Този тип мъртво блокиране може да възникне както при маршрутизацията с акумулиране и придвижване, така и при конвейеризираната маршрутизация с врязване. При конвейеризираната маршрутизация с

врязване съществува по-голяма вероятност за мъртво блокиране, тъй като пакетът се разпростира върху множество флит-буфери. Само заглавният флит на пакета съдържа информация за маршрутизиране, така че след като веднъж главата на пакета се предаде по канала, всички останали флитове трябва да я последват по същия канал. По този начин един единствен пакет може да заеме каналните ресурси през няколко комутатора. Важното в случая е, че ресурсите логически са свързани с каналите и съобщенията, придвижвайки се през мрежата, въвеждат зависимости между тези ресурси.



Фиг. 5.25. Мъртво блокиране при маршрутизация в мрежата

Основен подход за предотвратяване на мъртво блокиране в мрежата е да се анализират зависимостите между каналите в резултат на придвижването на съобщението в мрежата и да се осигури отсъствието на цикли в графа на зависимостите на каналите. Това е гаранция, че

какви конфигурации на мрежовия трафик няма да предизвикат мъртво блокиране.

Най-простият начин за предотвратяване на мъртво блокиране е да номерират каналните ресурси по такъв начин, че всеки локален маршрут да следва монотонна нарастваща (или намаляваща) последователност от номера на канали, като по този начин се гарантира, че не могат възникнат никакви цикли на зависимост в рамките на маршрутите.

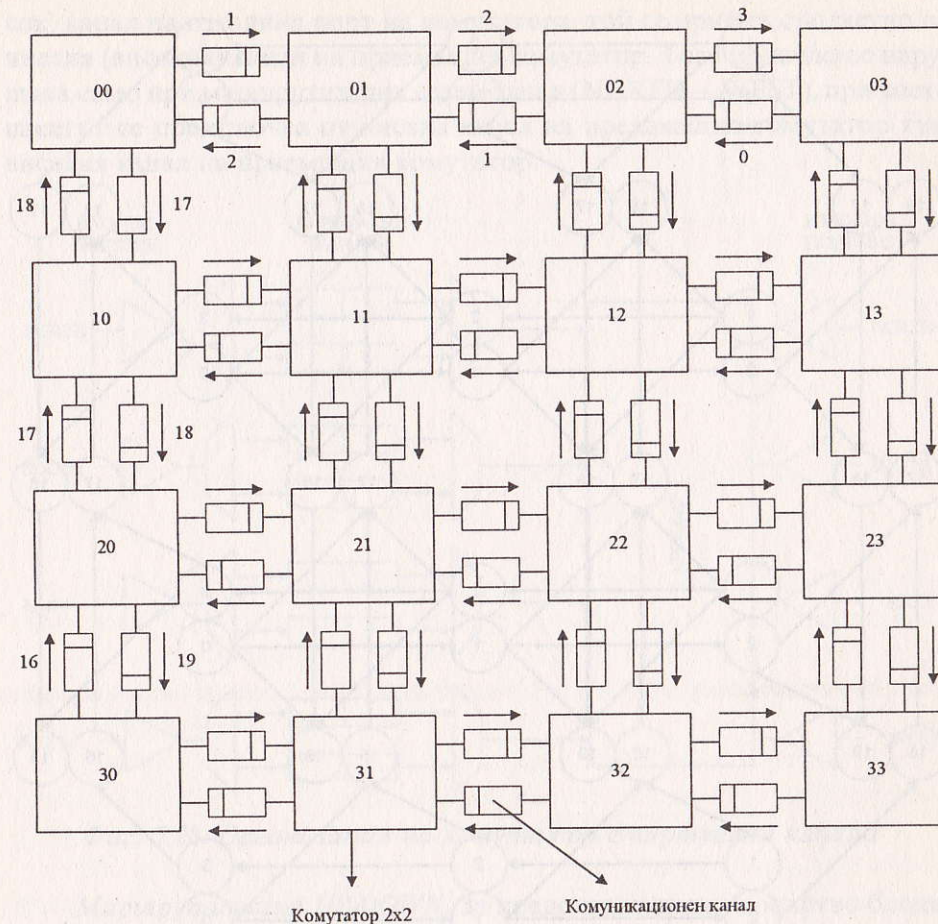
Разглеждаме възможностите за мъртво блокиране при $\Delta x, \Delta y$ маршрутизацията в двумерна решетка. Всеки двупосочен канал в мрежата се разглежда като двойка еднопосочни канали с независима номерация. Позитивен x канал $\langle i, y \rangle \rightarrow \langle i+1, y \rangle$ се означава с номер i , докато номерирането на негативните x канали започва от 0 при позитивния x канал с най-голям номер. Позитивният y канал $\langle x, j \rangle \rightarrow \langle x, j+1 \rangle$ се номерира с j и същевременно номерирането на негативните y канали започва от най-позитивния край. Този принцип на означаване на каналите е показан на фиг. 5.26. Всеки маршрут, следващ серия канали в посока x , следван завой на 90° и серия канали в y посока, е стриктно нарастващ. В графа на зависимостите на каналите (фиг. 5.27) всеки възел представя еднопосочна връзка в маршрута.

Възел А е свързан с възел В посредством дъга, ако е възможно пътът да премине последователно първо през канал А и после през канал В. Всички дъги в графа са насочени от възел с по-малък номер към възел с по-голям номер. Следователно не съществуват цикли в графа на зависимостите на каналите, въпреки че могат да съществуват множество цикли в мрежата.

За конкретна мрежова топология и фиксиран набор канални ресурси съществуват два основни подхода за проектиране на маршрутизиращи алгоритми, недопускащи мъртво блокиране:

- Ограничаване на множеството легални маршрути за пакетите;
- Ограничаване на начините за заемане на ресурсите.

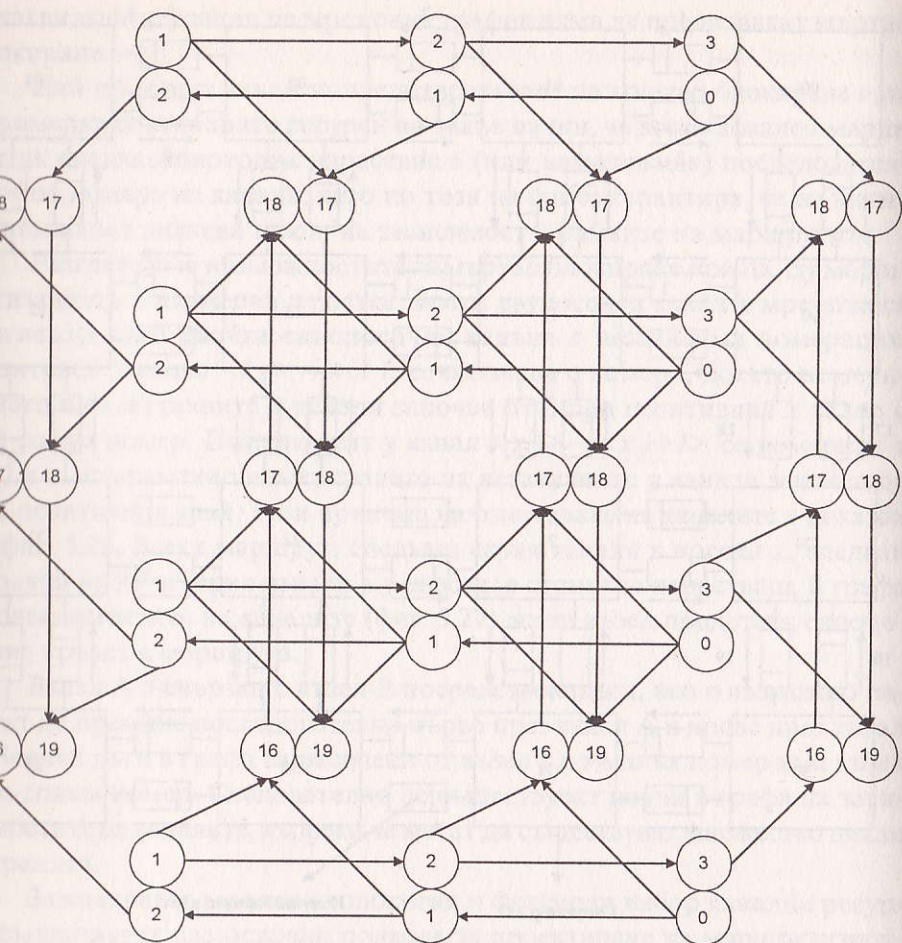
Виртуални канали. Основната техника за предотвратяване на мъртво блокиране в мрежи с конвейеризиран трансфер и маршрутизация от типа „дупка на червея“ е да се осигури множество буфери във всеки физически канал и всеки буфер да се раздели на група виртуални канали. Тази техника не увеличава броя на връзките и комутаторите в мрежата. Също така не се увеличава размерът на вътрешния кросбар във всеки комутатор, тъй като в даден момент само един флит се придвижва от комутатора към съответния изходен порт.



Фиг. 5.26. Номериране на каналите в мрежовия граф на двумерна решетка

На фиг. 5.28 е показана архитектурата на комутатор с виртуални канали за предотвратяване на мъртво блокиране. Всеки физически канал съдържа множество виртуални канали. Входните портове на комутатора разпределят потока пакети във виртуалните канали в отделните буфери. Виртуалните канали се мултиплексират допълнително, за да се избегне разширяването на кросбара на комутатора.

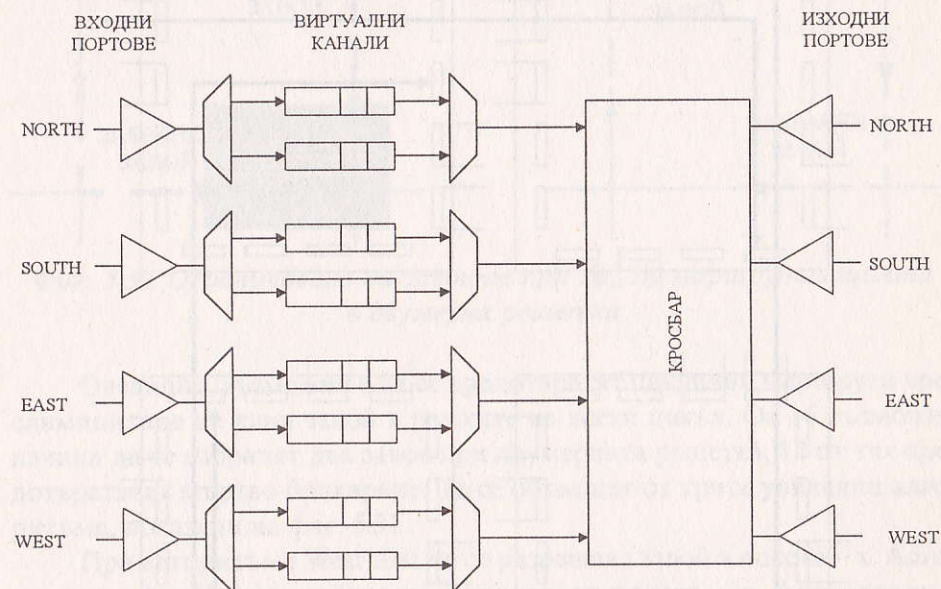
Предназначението на виртуалните канали е да предотвратят мъртво блокиране чрез систематично прекъсване на циклите в графа на зависимостите на каналите.



Фиг. 5.27. Граф на зависимостите на каналите на двумерна решетка

Нека да разгледаме 4-стъпковия цикличен маршрут, предизвикващ мъртво блокиране на фиг. 5.25. С цел предотвратяване на мъртво блокиране нека да организираме два виртуални канала в рамките на всеки физически канал, означени като нисък (с по-малък номер) и висок (с по-голям номер). Освен това съобщенията във възли с номера по-големи от номера на дестинацията си се маршрутизират по високите канали (с по-големи номера). Както се вижда от фиг. 5.29 цикълът, предизвикващ мъртво блокиране, е разрушен. Когато пакетът се предава от нисък (ви-

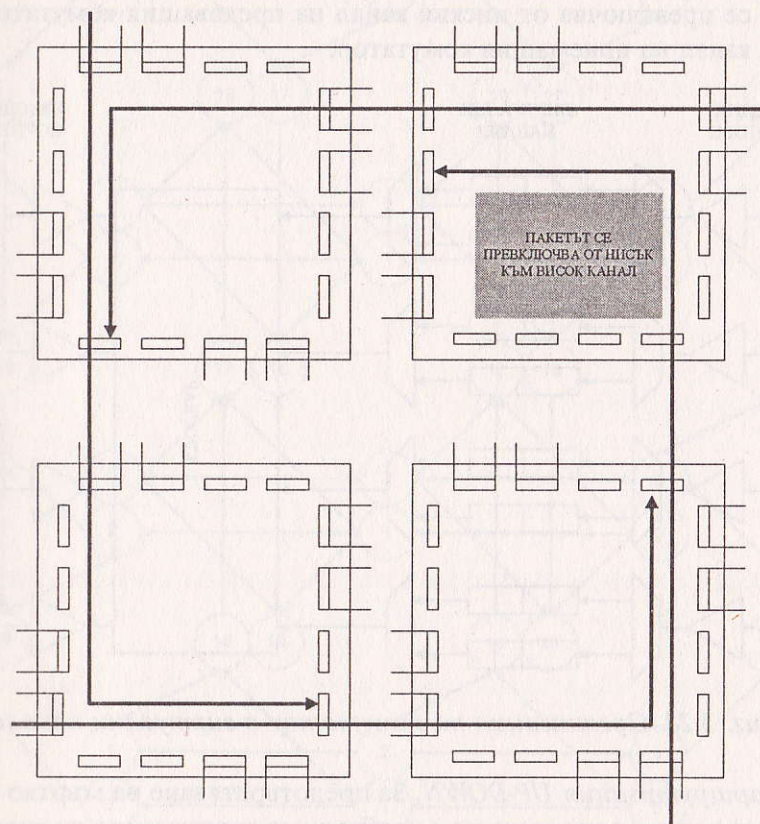
сок) канал на изходния порт на комутатора, той се приема съответно от ниския (високия) канал на приемащия комутатор. Това правило се нарушава само при маршрутизация север-запад (NORTH – WEST), при което пакетът се превключва от ниския канал на предаващия комутатор към високия канал на приемащия комутатор.



Фиг. 5.28. Организация на комутатор с виртуални канали

Маршрутизация UP-DOWN. За предотвратяване на мъртво блокиране не за всички топологии не е необходимо въвеждането на виртуални канали. При положение, че каналите са двупосочни, съществува прост алгоритъм за конструиране на маршрути, непредизвикващи мъртво блокиране в произволна топология, който ограничава множеството легални маршрути. Основен подход е да се конструира дървовиден маршрут до листа-дестинация с възходящо номериране на възлите към корена. За всяка дестинация се формира UP-DOWN маршрут, включващ последователност от канали към възли с по-големи номера, т.е. към корена на дървото, последващ завой и последователност от канали към възли с по-малки номера в посока на листата на дървото. Множеството маршрути, следващи тази ориентация, са свободни от мъртво блокиране. Мрежовият граф може да съдържа цикли, но графът на зависимостите на канали-

при UP-DOWN маршрутизацията не съдържа цикли. Множеството
соки канали формира насочен ацикличен граф, както и множеството
ски канали формира насочен ацикличен граф.



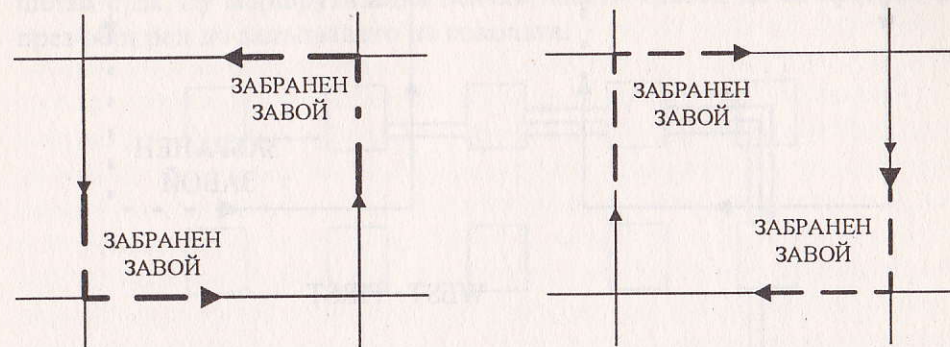
Фиг. 5.29. Предотвратяване на цикли на мъртво блокиране с виртуални канали

Комуникационният процесор на възела, изпращащ съобщението, анализира мрежовата топология и конструира дървовиден маршрут. След това той изчислява съответните UP-DOWN маршрути като проблем за разделяне на ограничено множество най-кратки маршрути. След това конструират съответни таблици за маршрутизация в комутаторите.

Маршрутизация с ограничаване на завоите. При двумерната решетка съществуват 8 завои, които формират два прости цикъла (фиг. 5.30).

При $\Delta x, \Delta y$ маршрутизацията в двумерна решетка се забраняват 4 завои: при придвижване в посока $-x$ ($+x$) се допуска завои в посока $-y$ ($+y$), но след като пакетът се придвижва в измерение y , не се допускат повече завои. Забранените завои са маркирани с пунктир на фиг. 5.30.

у ($+y$), но след като пакетът се придвижва в измерение y , не се допускат повече завои. Забранените завои са маркирани с пунктир на фиг. 5.30.



Фиг. 5.30. Ограничаване на завоите при $\Delta x, \Delta y$ маршрутизацията в двумерна решетка

Очевидно, възможно е да се предотвратят циклични маршрути чрез елиминиране на един завои в рамките на всеки цикъл. От 16 възможни начина да се забранят два завои при двумерната решетка, 12 от тях предотвратяват мъртво блокиране. Те се обхващат от трите уникални алгоритъма, показани на фиг. 5.31.

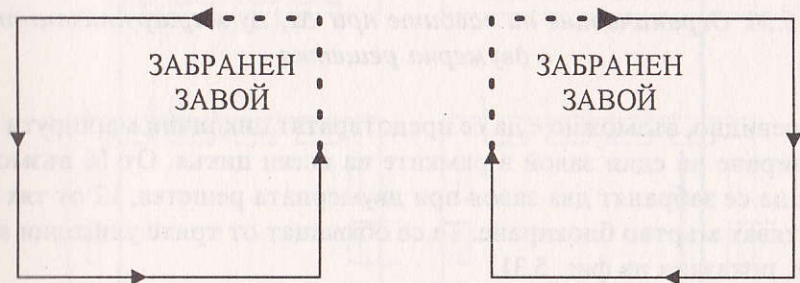
При алгоритъма West-first не се разрешава завои в посока $-x$. Алгоритъмът North-last забранява завоите от посока $+y$. Алгоритъмът Negative-first забранява завои от позитивна в негативна посока.

За произволна топология може да се приложи комбиниран алгоритъм, обединяващ концепциите на виртуалните канали и ограничаването на завоите.

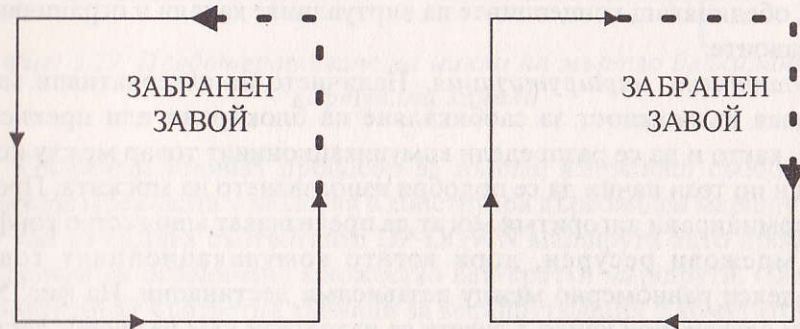
Адаптивна маршрутизация. Наличието на алтернативни маршрути дава възможност за заобикаляне на блокирани или прекъснати връзки, както и да се разпредели комуникационният товар между повече канали и по този начин да се подобри използването на мрежата. Простите детерминирани алгоритми могат да предизвикат множество конфликти за мрежови ресурси, дори когато комуникационният товар е разпределен равномерно между независими дестинации. На фиг. 5.32 е показан случай, при който 4 пакета са изпратени към различни дестинации в двумерна решетка. При $\Delta x, \Delta y$ маршрутизацията всички пакети са принудени да използват общи маршрути. Комуникацията е напълно сериализирана през тясно място при положение, че множество други алтернативни връзки са свободни.



WEST - FIRST



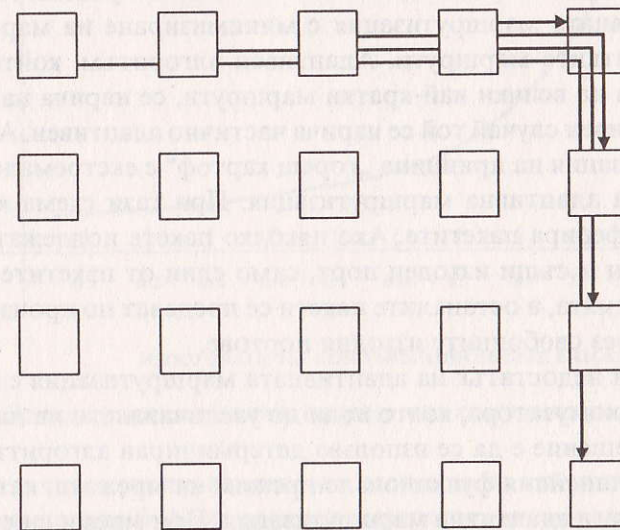
NORTH-LAST



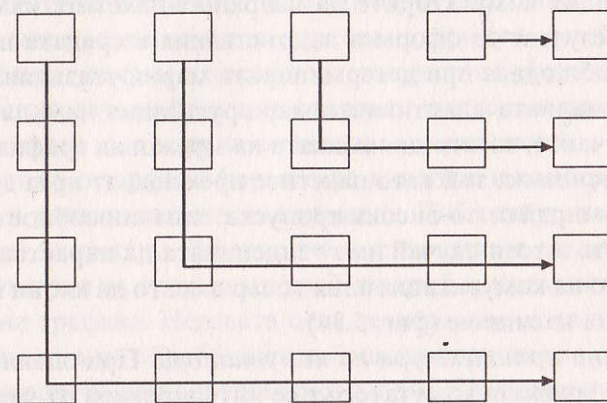
NEGATIVE-FIRST

Фиг. 5.31. Алгоритми за минимална маршрутизация с ограничаване на завоите при двумерна решетка

Алгоритъм с адаптивна маршрутизация би могъл да използва алтернативни маршрути, както е показано на фиг. 5.33. При двумерна решетка с Δx , Δy маршрутизация всички пакети трябва да се придвижват през общ ред до запълването на колоната.



Фиг. 5.32. Конфликти за мрежови ресурси в двумерна решетка при детерминиран алгоритъм за маршрутизация



Фиг. 5.33. Избягване на конфликти за мрежови ресурси в двумерна решетка при адаптивен алгоритъм за маршрутизация

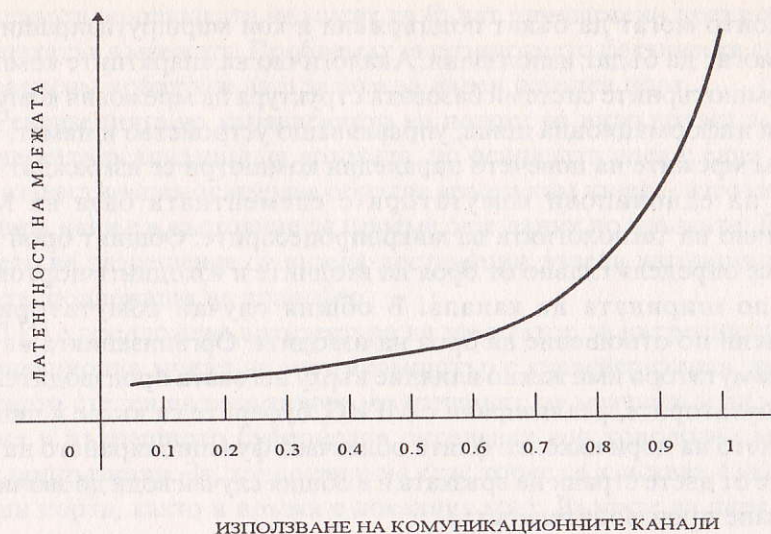
При адаптивната маршрутизация се използват алтернативни маршрути, като изборът на маршрутите се прави динамично от комутатора след анализ на динамиката на трафика по алтернативните пътища. При положение, че изискваният изходен порт е блокиран или отказал, комутаторът може да избере алтернативен канал за изпращането на пакета.

Адаптивната маршрутизация с минимизиране на маршрутите изира най-кратките маршрути. Адаптивен алгоритъм, който позволява ползването на всички най-кратки маршрути, се нарича напълно адаптивен, в противен случай той се нарича частично адаптивен. Алгоритъмът на маршрутизация на принципа „горещ картоф“ е екстремален случай на минимална адаптивна маршрутизация. При тази схема комутаторът кога не буферизира пакетите. Ако няколко пакета подлежат на комутация през един и същи изходен порт, само един от пакетите се изпраща към дестинацията, а останалите пакети се предават по произволни други маршрути през свободните изходни портове.

Основен недостатък на адаптивната маршрутизация е повишената сложност на комутатора, която води до увеличаването на латентността. Естественото решение е да се използва детерминиран алгоритъм за маршрутизация в линейния функционален режим на мрежата, като при насищане се прилага адаптивна маршрутизация. При мрежи с неунифициран състав, например двумерни решетки и тороиди, адаптивната маршрутизация намалява бързодействието при случаен трафик с равномерно разпределение. Стохастичните вариации в комуникационния товар предизвикват временно блокиране в мрежата. В двумерна решетка съществува тенденция комутаторите да изпращат пакетите към центъра на мрежата. В резултат се оформят задръствания в средата на решетката, което не се наблюдава при детерминирата маршрутизация.

Неминималната адаптивната маршрутизация намалява бързодействието в случаите, когато динамиката на мрежовия трафик приближава мрежата на насищане, тъй като пакетите преминават през допълнителни скопки, консумирайки по-висока пропускателна способност. Латентността на мрежата в този случай има тенденцията да нараства драстично с увеличаването на комуникационния товар вместо да клони към постоянна стойност на насищане (фиг. 5.34).

Вътрешна архитектура на комутатора. При оценяването на параметрите на мрежата комутаторът се интерпретира от гледна точка на своята степен. Цената на някои компоненти на комутатора нарастват линейно с увеличаването на неговата степен респ. на предавателите, приемниците и буферите. Цената на вътрешното свързване (кросбара) нара-



Фиг. 5.34. Зависимост на закъснението от комуникационния товар

ства квадратично с увеличаването на степента на комутатора. Капацитетът на вътрешните буфери, както и сложността на маршрутизиращата логика също могат да нарастват повече от линейно в зависимост от степента на комутатора. При съвременните комутатори, реализирани с МГИС (много големи интегрални схеми), доминиращото ограничение е броят на изводите, който е пропорционален на броя на портовете, умножен по ширината на всеки порт. Комутаторът включва множество входни портове, множество изходни портове, вътрешен кросбар, свързващ всеки входен порт с всеки изходен порт, вътрешни буфери и управляваща логика. В общия случай броят на входните портове е равен на броя на изходните портове и се нарича *степен на комутатора*.

Мрежовият интерфейс форматира пакетите и конструира маршрутизиращата и управляващата информация. Той осигурява много по-голямо по обем вътрешно буферизиране в сравнение с комутаторите. Той реализира контрол за грешки от край-до-край и изпълнява функции по управление на трафика. Неговата цена се определя от капацитета на паметта, сложността на обработката и броя на портовете.

Проектирането на вътрешносистемната комуникационна мрежа за паралелни компютри се свежда до проектирането на комутаторите и начина на тяхното свързване. Степента на комутатора, неговият механизъм за вътрешна маршрутизация и вътрешно буферизиране определят тополо-

ите, които могат да бъдат поддържани и кои маршрутизиращи алгоритми могат да бъдат използвани. Аналогично на апаратните компоненти на компютърните системи базовата структура на мрежовия комутатор включва информационна шина, управляващо устройство и памет. Понастоящем мрежите на повечето паралелни компютри се изграждат на основата на едночипови комутатори с елементната база на МГИС. Аналогично на технологията на микропроцесорите. Общият брой на изходите се определя главно от броя на входните и изходните портове, умножен по ширината на канала. В общия случай комутаторите са ограничени по отношение на броя на изходите. Организацията на буферите в комутатора има важно влияние върху неговата производителност. При комутаторите, реализирани с МГИС, буферите са вътре в чипа. Буферизирането на дори няколко флита облекчава функционирането на комутаторите от двете страни на връзката и в общия случай води до значително растване производителността.

Атрактивен подход е да се осигурят независими буфери FIFO във всеки входен порт. Всеки буфер трябва да приема по един фит на всеки такт и да предава по един фит към изхода, така че вътрешната пропускателна способност на комутатора се адаптира лесно към входния поток от пакети. Функционирането на комутатора е относително просто. Той извършва мониторинг на главата на всяка входна структура FIFO, определя желан изходен порт и планира съответно пакетите да се придвижат през кросбара. С всеки входен порт е асоциирана маршрутизираща таблица за определяне на желан изходен порт. За тази цел е необходима таблица за маршрутизиране.

При стратегията с входно буфериране съществува проблемът с блокиране от „главата“ на опашката. В случаите, когато два или повече портове имат пакети с дестинации, изискващи комутиране през един и същи изходен порт, само един от пакетите ще бъде избран, докато останалите ще бъдат блокирани. Пакетите в опашката след главата може да изискват комутиране през някой от свободните портове, но няма да имат възможност за придвижване, т.е. те също ще бъдат блокирани. Проблемът с блокирането от главата на опашката се влошава при наличие на вериги от редуващи се лавинообразни потоци към различни изходни портове. При равномерно разпределяне на трафика всеки лавинообразен поток ще доведе до блокиране на всички входове. Дори когато няма задължаване за даден изход в комутатора, пакетът в началото на входния буфер може да изисква комутиране през изходен порт, който е блокиран поради задържане в друга точка на мрежата. В този случай пакетите

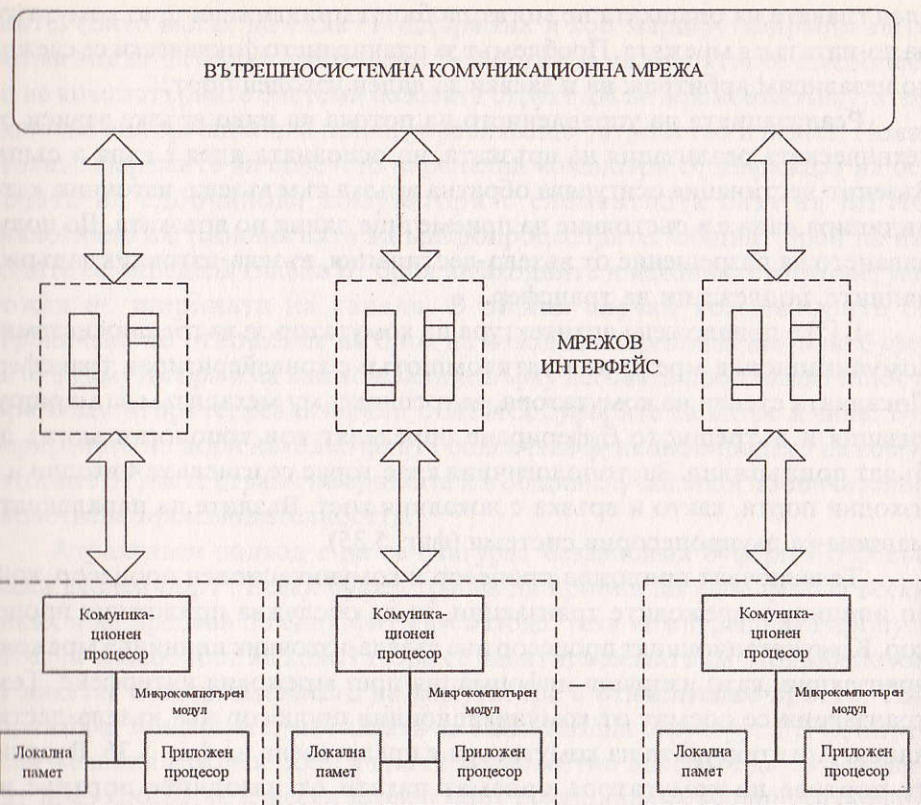
след главата на опашката не могат да бъдат придвижени през комутатора по-нататък в мрежата. Проблемът за планирането фактически се свежда до независим арбитраж на n заявки за даден изходен порт.

Реализацията на управлението на потока на ниво връзка зависи от техническата реализация на връзката, но основната идея е една и съща. Възелът-дестинация осигурява обратна връзка към възела-източник, като индицира дали е в състояние да приеме още данни по връзката. До получаването на разрешение от възела-дестинация, възела-източник задържа данните, подлежащи на трансфер.

В [5] е предложена архитектура на комутатор за вътрешносистемна комуникационна мрежа на мултикомпютър с конвейеризиран трансфер. Локалната степен на комутатора, вътрешният му механизъм за маршрутизация и вътрешното буфериране определят кои топологии могат да бъдат поддържани. За топологичния клас торус се изискват 4 входни и 4 изходни порта, както и връзка с локалния хост. Възлите на паралелната машина са двупроцесорни системи (фиг. 5.35).

Те включват приложен процесор и комуникационен процесор, който иницира мрежовите транзакции, за да обслужва приложния процесор. Комуникационният процесор във възела-източник иницира мрежови транзакции, като изпраща информация през мрежовия интерфейс. Тези транзакции се поемат от комуникационния процесор във възела-дестинация. Архитектурата на комутатора е представена на фиг. 5.36. Входните портове на комутатора приемат пакети от изходните портове на съседните комутатори (висок приоритет) или новогенерирани пакети от комуникационния процесор на локалния хост (нисък приоритет). В случай, че идващите пакети от мрежата са адресирани за локалния хост, е осигурен кратък маршрут за директното им предаване към локалния комуникационен процесор. Това решение дава възможност да се избегне добавянето на допълнителен входен и допълнителен изходен порт за трафика, генериран от и адресиран за локалния хост.

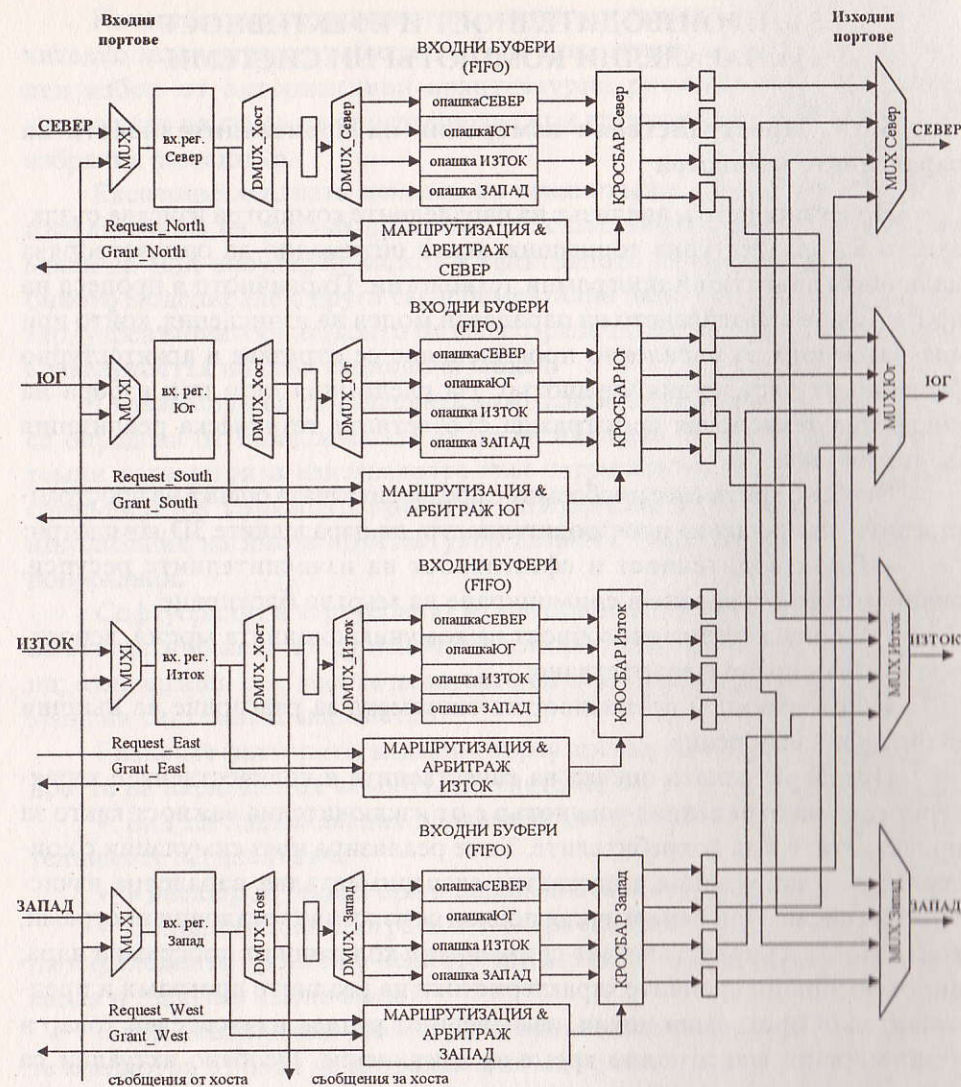
Входният порт има 4 буфера, всеки от които селектира пакетите за даден изходен порт. Така при пристигането си пакетите се сортират по дестинации и по този начин се елиминира блокирането от началото на опашката. Всеки входен порт има буфери FIFO, които синхронизират приеманите данни с тактовата честота на комутатора. Всеки буфер приема по един фит на всеки машинен цикъл и предава по един фит към изходния порт, така че вътрешната пропускателна способност на комутатора осигурява приемането на предаваните данни. Комутаторът съдържа 4 неблокиращи кросбара, всеки от които е свързан с определен



Фиг. 5.35. Архитектура на хост-възлите в мултикомпютъра

ходен порт. В логически аспект, неблокиращите кросбари представляват 4-входови мултиплексори, реализирани в МГИС като обща шина с райвери с три състояния. Управлението на потока се осъществява с допълнителни линии, осигуряващи сигналите заявка/разрешение“ при ръркостискане“.

Комутаторът функционира по следния начин: той проследява дали даден изходен порт е свободен и съответно планира пакетите за предаване през кросбара. Всеки входен порт има маршрутизираща логика, която определя съответния изходен порт. При маршрутизацията с врязване независимо решение се взема за всеки пакет. Логиката за маршрутизация изчислява необходимия изходен порт и активира сигнала за заявка за изходен порт. Планиращата логика на изходния порт прави арбитраж на заявките, избира една и активизира съответния сигнал за



Фиг. 5.36. Архитектура на комутатора 4x4

разрешение. Когато даден входен буфер подаде заявка за даден изходен буфер, заявката се заддържа и се нарежда в опашката. Обслужва се най-старата заявка. Следователно по своята същност логиката за маршрутизация представлява краен автомат, който изпраща всички флитове на пакета по един и същ изходен канал.

5.6. ПРОИЗВОДИТЕЛНОСТ И ЕФЕКТИВНОСТ НА ПАРАЛЕЛНИ КОМПЮТЪРНИ СИСТЕМИ

5.6.1. Пространствени измерения на производителността на паралелните компютри

Проектирането и анализът на паралелните компютри изисква създаването на архитектурна концепция, която оптимално да оползотворява наличните апаратни и програмни технологии. Първичното в процеса на проектиране е създаването на паралелен модел на изчисления, който при еквивалентен език за паралелно програмиране се отразява в архитектурно решение на паралелния компютър. На следващия етап при избора на конкретна технология се изгражда съответната техническа реализация на компютъра.

Особено критичен е проблемът с предварителната оценка на пространствените измерения на производителността на паралелните 3D-компютри:

- Производителност и ефективност на изчислителните ресурси, максимално откриване и елиминиране на мъртво блокиране;
- Пропускателна способност на комуникационната мрежа, вероятност за блокиране и задръстване;
- Латентност и прецизност на системата за реагиране на външни събития и изключения.

Предварителната оценка на качествените и количествените характеристики на паралелния компютър е от изключителна важност както за проектантите и за потребителите. Тя се реализира чрез симулации с консервативни допускания в адекватна експериментална паралелна изчислителна среда. При йерархичния подход се използват еталонни програми, които могат да представляват синтезирани колекции от програмни ядра, оксимиращи средните характеристики на реалните програми и представителни програмни цикли, извлечени от реален изчислителен товар и сумиращи максимално време за изпълнение. Особено актуални са симулациите на перспективни приложения, на основата на резултатите от които компютърният архитект трябва да избере една от възможните алтернативи:

- Еволюционно усъвършенстване на съществуващо архитектурно решение с цел съхраняване на съвместимост със съществуващи модели;
- Създаване на радикални нови архитектурни концепции и реализации, които съвместно с използването на авангардни технологии да доведат до качествен скок в системната производителност.

Качествените характеристики на паралелния компютър са изключително важни и за потребителите, тъй като служат за основа за оптимален избор от алтернативни архитектурни решения и след това за адаптиране на стила на програмиране към архитектурната специфика на избрания компютър.

Експерименталната дейност по компютърна архитектура обхваща изпълнението на реални програми от съществуващи или симулирани машини. Най-популярен подход е ефективното съчетаване на имитационното моделиране с други експериментални подходи, като софтуерен и хардуерен мониторинг, които от своя страна проверяват и потвърждават адекватността на симулационния модел.

Гъвкавостта на симулацията като основен експериментален подход се определя от възможността за лесна промяна на моделираните системни параметри и изискванията към натрупваните статистики. Недостатъците са голямото време за симулации, както и неизбежната идеализация на някои архитектурни детайли, въпреки консервативните допускания.

Софтуерният и хардуерният мониторинг осигуряват висока скорост на статистическите натрупвания, но изискват експериментални прототипи, отличаващи се с висока цена, като не осигуряват възможност за промяна на системните параметри.

Главните факторите, влияещи върху производителността и ефективността на паралелните компютърни системи, са:

- Вид на изпълнявания паралелен алгоритъм респ. заложения потенциален паралелелъзм;
- Краен брой процесори в паралелната компютърна система;
- Допълнителни програмни и апаратни разходи за координацията на паралелните процеси в нейните три аспекта (синхронизация, комуникация и взаимно изключване).

В най-общ вид зависимостта на производителността на паралелните компютри от вида на изпълнявания алгоритъм се представя от закона на Амдал (фиг. 5.37). За оценка на ефективността на паралелните компютри относно даден алгоритъм се въвежда параметърът *степен на паралелизъм L*, отразяващ потенциалния паралелизъм на алгоритъма:

$$L = T_{seq} / (T_{seq}' + T_{par, id}) \quad (5.2)$$

където:

T_{seq} – време за последователно изпълнение на инструкциите на изпълняваната програма;

$T_{par,id}$ – време за изпълнение на паралелните процеси на програмата от идеална паралелна компютърна система (при наличие на безкраен брой процесори и без отчитане на допълнителните разходи за координацията на процесите);

$T_{seq'}$ – време за изпълнение на последователните процеси в програмата.

За времето за последователно изпълнение на програмата е в сила отношението:

(5.3)

ето:

T_i – времетраене на изпълнението на i -тата инструкция;

C – общ брой на инструкциите в програмата.

При статично-паралелните алгоритми броят на паралелните процеси е известен преди изпълнението на програмата, така че времето за негово изпълнение може да се определи по следния начин:

$$T_{par,id} = \text{MAX}_{r \in [1, Q]} \left\{ \left(\sum_{j=1}^{S_r} T_j \right) \right\} + \sum_{k=1}^S \sum_{j=1}^{S_k} T_{kj} \quad (5.4)$$

ето:

Q – брой на паралелните процеси, генерирани при декомпозицията на програмата на независими задачи;

S_r – брой на инструкциите в r -тия паралелен процес при изпълнението на програмата;

T_j – времетраене на изпълнението на j -тата инструкция в рамките на паралелния процес на програмата;

S – брой на последователните процеси при изпълнението на програмата;

S_k – брой на инструкциите в k -тия последователен процес при изпълнението на програмата;

T_{kj} – времетраене на изпълнение на j -тата инструкция в рамките на последователен процес при изпълнението на програмата.

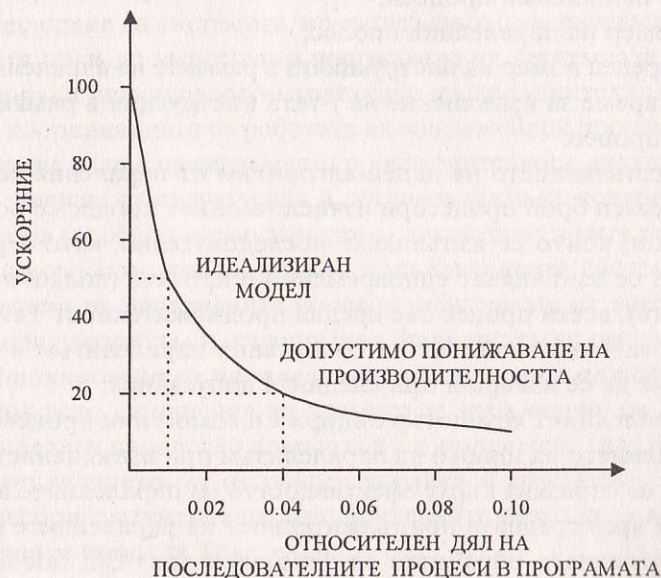
Очевидно времето $T_{par,id}$ се определя от времетраенето на най-дължителния паралелен процес на алгоритъма, тъй като поради безкраен брой процесори всички паралелни процеси в идеалния паралелен компютър се изпълняват едновременно.

Следователно за степента на паралелизъм L при изпълнението на статично-паралелни алгоритми се получава:

$$L = \frac{\sum_{i=1}^C T_i}{\text{MAX}_{r \in [1, Q]} \left\{ \left(\sum_{j=1}^{S_r} T_j \right) \right\} + \sum_{k=1}^S \sum_{j=1}^{S_k} T_{kj}} \quad (5.5)$$

За да се оцени влиянието на крайния брой процесори върху производителността на паралелната компютърна система, разглеждаме идеален паралелен компютър с фиксиран брой процесори, като пренебрегваме допълнителните системни разходи за координацията на процесите в системата. **Ефективният паралелизъм E_p** се дефинира като отношение на времето за паралелно изпълнение на дадена програма от идеален паралелен компютър (с безкраен брой процесори) $T_{par,id}$ към времето, необходимо за изпълнението ѝ от паралелен компютър с краен брой n процесори $T_{par,n}$:

$$E_p = T_{par,id} / T_{par,n} \quad (5.6)$$



Фиг. 5.37. Закон на Амдал

Ефективното използване на потенциалните възможности на паралелните компютърни архитектури изисква оптимална декомпозиция на работата на паралелни процеси, при което изпълнението на алгоритъма ще се реализира поетапно с междинно комбиниране на частичните резултати, което в значителна степен намалява ефективния паралелизъм и паралелната обработка.

За оценката на ефективния паралелизъм при реализацията на паралелни асинхронни изчисления с комуникаращи процеси в мултикомпютърна система се въвежда параметърът средно време за реализацията на паралелен процес, тъй като нивото на паралелизма е променливо и варира от единична инструкция до процедура, обхващаща хиляди инструкции. Следователно за средната продължителност на паралелния процес T_{av} в асинхронна паралелна компютърна система, отразяваща непосредствено нивото на паралелизъм в изчислителната система, се получава:

$$T_{av} = \left(\sum_{i=1}^Q \sum_{j=1}^{S_i} T_{ij} \right) / Q \quad (5.7)$$

където:

Q – броят на процесите, формирани при декомпозицията на работата на независими процеси;

i – номер на паралелния процес;

j – пореден номер на инструкцията в рамките на паралелния процес;

T_{ij} – време за изпълнение на j -тата инструкция в рамките на i -тия паралелен процес.

При изпълнението на даден алгоритъм от паралелна компютърна система с краен брой процесори изчислителният процес се осъществява в Q/n групи, които се изпълняват последователно, като в рамките на всяка група се изпълняват едновременно n процеса (колкото е броят на процесорите), всеки процес със средна продължителност T_{av} .

Следователно анализът на ефективния паралелизъм в мултикомпютърна система може да се извърши при следните допускания:

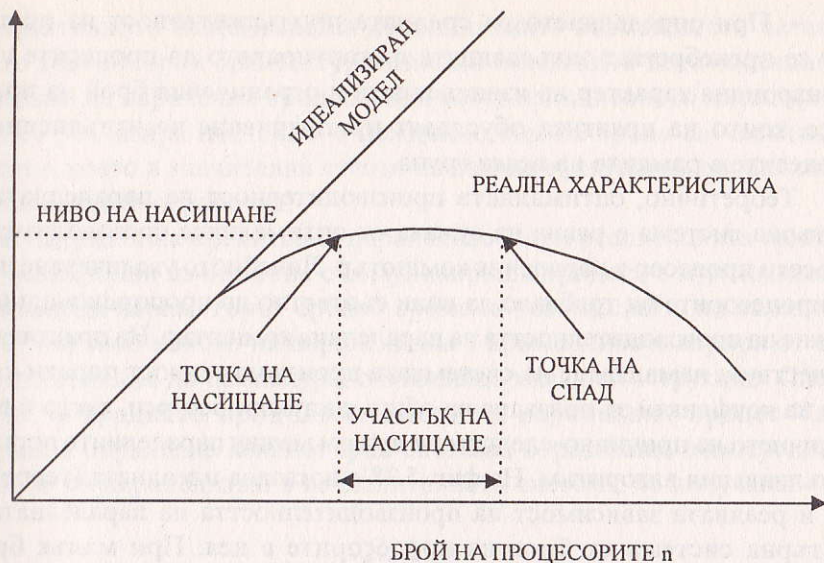
- Паралелният компютър съдържа n хомогенни процесори;
- Влиянието на нивото на паралелизъм при изпълнението на даден алгоритъм се отразява върху ефективността на паралелните асинхронни изчисления чрез средната продължителност на паралелните процеси;
- Паралелната обработка се реализира чрез Q/n последователни групи, като в рамките на всяка група се реализират паралелно n процеса и наличните n процесора;

• При определянето на средната продължителност на процесите T_{av} се пренебрегват закъсненията в стартирането на процесите поради асинхронния характер на изчисленията и ограничения брой на процесорите, които на практика обуславят припокриване на изпълнението на процесите в рамките на всяка група.

Теоретично, оптималната производителност на паралелната компютърна система е равна на сумата от оптималните производителности на всеки процесор в паралелния компютър. Линеиното увеличаване на броя на процесорите би трябвало да води съответно до пропорционално нарастване на производителността на паралелния компютър. На практика обаче съществува намаляване на системната производителност поради наличието на конфликти за ползване на общи системни ресурси, както и поради наличието на причинно-следствени връзки между паралелните процеси на изпълнявания алгоритъм. На фиг. 5.38 е показана идеалната (теоретичната) и реалната зависимост на производителността на паралелната компютърна система от броя на процесорите в нея. При малък брой на процесорите увеличаването на броя им води до пропорционално нарастване на системната производителност. При достигане на така наречения прагов брой на процесорите по-нататъшното увеличаване на броя на процесорите не води до повишаване на системната производителност (участък на насищане). След преминаване на горната граница на броя на процесорите при насищане на системата, по-нататъшното увеличаване на броя на процесорите води до значително понижаване на системната производителност, поради значителното нарастване на допълнителните системни разходи за координацията на работата на големия брой процесори. Точките на насищане и спад на системната производителност, както и нивото на насищане, зависят от структурата и организацията на конкретната паралелна система (особено силно зависят от характеристиките на вътрешно-системната комуникационна мрежа), допълнителните системни разходи за координация на паралелните процеси, обусловени от спецификата на изпълнявания алгоритъм и ограничения брой системни ресурси.

Ефективността на паралелната компютърна архитектура E_{pa} се дефинира като отношение на времето за изпълнение на даден алгоритъм от идеален паралелен компютър с n процесора $T_{par,id,n}$ към времето за изпълнението му от реален паралелен компютър с ограничен брой процесори с отчитане на допълнителните разходи за координация на паралелните процеси $T_{par,real,n}$:

$$E_{pa} = T_{par,id,n} / T_{par,real,n} \quad (5.8)$$



Фиг. 5.38. Зависимост на производителността на паралелните компютри от броя на процесорите

В контекста на мултикомпютрите ефективността на паралелната компютърна архитектура се определя предимно от удължаването на средната продължителност на паралелния процес поради допълнителното време за координация на паралелните процеси в системата.

Ускорението Sup при паралелната обработка се дефинира като отношение на времето за последователно изпълнение на даден алгоритъм към времето за паралелното изпълнение на същия алгоритъм от паралелна компютърна система $T_{par,real,n}$:

$$Sup = T_{seq} / T_{par,real,n} \quad (5.9)$$

Ефективността на използването на процесорите E_{pe} се определя като отношение на ускорението към броя на процесорите в паралелния компютър:

$$E_{pe} = Sup / n \quad (5.10)$$

Натоварването на ресурсите (изчислителни и комуникационни) $U_{resource}$ се определя като отношението на времето на тяхната заемотост към времето за изпълнение на алгоритъма от паралелния компютър $T_{par,real,n}$:

$$U_{resource} = T_{active} / T_{par,real,n} \quad (5.11)$$

5.6.2. Производителност на вътрешносистемни комуникационни мрежи за паралелни компютри

ЛАТЕНТНОСТ. Времето за трансфера на n байта информация от източника до дестинацията има 4 компоненти, както следва:

$$Time(n)_{s-d} = OV + RD + CO + CD \quad (5.12)$$

където:

OV (OVERHEAD) – допълнителни разходи за присъединяване на съобщението в трафика на мрежата и излизането му от трафика на мрежата при действителното му предаване от край до край, свързан с интерфейса възел-мрежа;

RD (ROUTING DELAY) – закъснение за маршрутизация;

CO (CHANNEL OCCUPANCY) – заетост на комуникационните канали;

CD (CONTENTION DELAY) – закъснение поради блокиране, което отразява престоя на съобщенията при пълен буфер на приемния комутатор по маршрута;

S-D (SOURCE-DESTINATION) – източник-дестинация.

Заетостта на комуникационните канали осигурява удобна долна граница за латентността при комуникация. Заетостта на комуникационната линия зависи от ширината на канала, скоростта на предаване на сигналите и количеството на управляващата информация, която от своя страна зависи от топологията на мрежата и алгоритъма за маршрутизация. Разглеждана „извън мрежата“, заетостта на комуникационните канали представлява времето за трансфер на съобщението през най-натоварения канал в рамките на маршрута. Разглеждана „вътре в мрежата“, тази заетост, се определя от заетостта на каналите на всяка стъпка на маршрутизация. За определен период от време комуникационният асистент е ангажиран за обслужване на заявката от контролера на паметта или процесора да спулира пакета в мрежата. Всеки комуникационен канал е зает определено време с предаване на съобщението по маршрута от източника до местоназначението, както и комуникационния асистент на възела-дестинация. В този аспект е необходимо да се изтъкне важността на ефективността на кодирането на пакета. Пликът на пакета също увеличава заетостта на каналите, защото в източника се добавят символи за заглавието и опащката, които се отстраняват в дестинацията. Така например за товар с размер n заетостта на канала е $(n + n_e)/b$, където n_e е размера на плика и b е физическата пропускателна способност на канала. Въвежда се **ефективна пропускателна способност** на комуникационната връзка,

се формулира от гледната точка „извън мрежата“ и се базира на техническата пропускателна способност на комуникационната връзка поне пакети с фиксиран размер: $(n/n+n_e)$.

Закъснението за маршрутизация от гледната точка „извън мрежата“ включва времето за трансфер на даден символ – например първия символ на съобщението от източника до дестинацията. Разглеждана в рамките на мрежата, всяка стъпка по маршрута добавя закъснение за маршрутизация, което се акумулира в тотално закъснение, виждано „извън мрежата“. Закъснението за маршрутизация зависи от броя на каналите по маршрута, формиращи т.нар. дистанция за маршрутизация h и закъснението D , възникващо във всеки комутатор, необходимо за селекция на изходящия изходен порт. При анализа на производителността при комуникация е удобно закъснението при маршрутизация, възникващо в интерфейса възел-мрежа, да се разглежда аналогично на закъснението в комутаторите. Дистанцията за маршрутизация зависи от мрежовата топология, алгоритъма за маршрутизация и конкретната двойка източник-дестинация. Общото закъснение силно се влияе от стратегиите за маршрутизация и маршрутизация.

При маршрутизацията с акумулиране и придвижване на пакета целият пакет се получава в даден комутатор преди да се придвижи по следващата връзка. Тази стратегия се използва при повечето глобални компютърни мрежи и беше популярна при по-старите модели паралелни компютри. В този случай, мрежовата латентност за пакет от n байта, очакващо пакета е:

$$T_{sf}(n, h) = h(n/b + \Delta) \quad (5.13)$$

където Δ е допълнително време за маршрутизация на стъпка, h – дистанция за комуникация и n – размер на данните.

От уравнение (5.13) се вижда, че мрежовата топология е от първостепенно значение за мрежовата латентност, тъй като топологията определя дистанцията за маршрутизация h . На практика определящите фактори са повече.

Нека да разгледаме влиянието на стратегията за комутация. При маршрутизацията на канали се очаква закъснение, пропорционално на h за придвижване на канала, реконфигурация на всички комутатори по маршрута и информирането на източника за заемане на маршрута. След това пакетите се придвижват в канала за време n/b плюс допълнително малко закъснение, пропорционално на h . Следователно за мрежовата латент-

ност, измерена в мрежови цикли, t , за n -байтово съобщение, предавано по дистанция h при мрежа с комутация на канали, е:

$$T_{cs}(n, h) = n/b + h \Delta \quad (5.14)$$

В уравнение (5.14) закъснението за инициализация и закъснението за маршрутизация са допълнителни членове, независими от размера на съобщението. Така с увеличаването на дължината на съобщението влиянието на дистанцията за маршрутизация, а следователно и на топологията, относително намаляват. Комутацията на канали се използва традиционно в телекомуникационните мрежи, тъй като времето за инициализация е кратко в сравнение с продължителността на информационния обмен. Комутацията на канали се използва в малко на брой комуникационни мрежи на паралелни компютри като Meiko CS-2 и BBN Butterfly. Важна разлика е, че при паралелните компютри каналът се конфигурира от съобщението, което се придвижва през мрежата и поддържа отворен маршрута като канал. По-традиционен подход е да се определи маршрутът в източника, да се конфигурират комутаторите и след това да се предаде информацията по канала.

Възможно е също така да се използва комутация на пакети, като се намали латентността при маршрутизацията с акумулиране и придвижване на съобщението. Основният извод от уравнение (5.13) е, че закъснението е произведението от дистанцията за маршрутизация и заостотта за цялото съобщение. Дългото съобщение може да се фрагментира на няколко малки пакети, които да се предават конвейеризирано през мрежата. В този случай закъснението е:

$$T_{sf}(n, h, n_p) = (n - n_p)/b + h(n_p/b + \Delta) \quad (5.15)$$

където n_p е размер на фрагментите.

Ефективното закъснение при маршрутизация е пропорционално по-скоро на размера на пакета, отколкото на размера на съобщението. В основни линии това е основният подход, възприет в традиционните комуникационни мрежи като Internet.

При вътрешносистемните комуникационни мрежи на паралелните компютри идеята за конвейеризация на маршрутизацията и комуникацията е развита в по-висока степен. Повечето паралелни компютри използват комутация на пакети с маршрутизация с връзване (cut-through), при която комутаторът взема решение за маршрутизация след проверка на първите фитовете на заглавието и осигурява възможност останалата част на пакета да се „вреже“ от входния към изходния канал, така че предава-

на дори един единствен пакет е конвейеризирано. В този случай управлението за латентността е аналогично на това за комутацията на канали. Разлика, че коефициентът на маршрутизацията D ще бъде различен, тъй като механизмите на процесите са доста различни:

$$T_{ct}(n, h) = n/b + h \Delta \quad (5.16)$$

Трябва да се отбележи, че при маршрутизацията с връзване едно съобщение може да заеме целия маршрут от източника до дестинацията аналогично на комутацията на канали. Главата на съобщението конфигурира маршрута, придвижвайки се към дестинацията, маршрутът се освобождава с придвижването на опашката на съобщението.

До тук направеният анализ на производителността разглежда съобщенията, придвижващо се от източника до дестинацията без наличието на буфер в мрежата. В случая без трафик мрежата може да се разглежда като конвейер с време за инициализирането му, дължина и такт. Различните стратегии за комутация и маршрутизация променят ефективната структура на конвейера, докато топологията, пропускателната способност на връзките и фрагментацията определят дължината на конвейера и такта. Практически мрежите представляват плетеница от пресичащи се части от множество конвейери.

Целта на използването на мрежа вместо обща шина е да се осигури възможност за реализацията на множествен трансфер на данни едновременно. Следователно мрежата трябва да поддържа механизъм за уреждане на конфликтите. Поведението на мрежата при наличие на конфликти зависи от няколко аспекта на мрежовия дизайн – топологията, стратегията за комутация, алгоритъма за маршрутизация. Основният проблем е, че в даден момент каналът може да обслужва само едно съобщение. В този случай, че две съобщения едновременно се опитват да използват един и същ канал, едното съобщение трябва да бъде блокирано. В общия случай комутатор осигурява арбитраж за ползване на изходните канали. В този начин комутаторът ще избере един от входящите пакети, съществени са за един и същи изходен порт, а останалите ще бъдат блокирани. Очевидно, конфликтите увеличават латентността при комуникация в мрежовите възли. Размерът на това увеличение зависи от конкретния механизъм за арбитраж в мрежата, който от своя страна зависи от основната стратегия на мрежовия дизайн. При комутацията на пакети. Напротив, при маршрутизацията с акумулиране и придвижване, ако множество буфери-

рани пакети в комутатора изискват един и същ изходен канал, само един ще бъде избран и комутиран, докато останалите ще бъдат блокирани в буферите, докато бъдат избрани. По този начин конфликтите увеличават закъснението поради престой в опашките в буферите на комутаторите. При комутацията на канали ефектът от конфликтите възниква, когато се прави опит да се конфигурира каналът. В общия случай се прави проба за конфигуриране на канала до дестинацията и ако бъде срещнат резервиран канал, пробата за конфигуриране се преустановява. След известен интервал от време мрежовият интерфейс прави нов опит за конфигуриране на канала. Така времето за конфигуриране на канала се увеличава при наличие на конфликти, но след като веднъж каналът е конфигуриран, предаването се осъществява при пълна скорост за цялото съобщение.

При маршрутизацията с връзване са възможни два варианта за блокиране на пакетите. При подхода с „виртуално връзване“ блокираните входящи пакети се спулirat в буфер, така че мрежовото поведение при конфликти деградира в сравнение с маршрутизацията с акумулиране и натрупване. При маршрутизация от типа „дупка на червея“ в комутатора се буферират само няколко флита на съобщението, докато опашката се оставя по маршрута. Блокираната част на съобщението е аналогична на канал, поддържан отворен през част от мрежата.

Комутаторите съдържат буфери с ограничен размер за пакетите, така че при множество продължителни конфликти буферите на комутаторите могат да се запълнят. Очевидният проблем е какво се случва на входящите пакети при пълни буфери в комутатора. В традиционните комуникационни мрежи връзките са дълги и типичният подход е да се загуби пакетът. Следователно при наличието на конфликти мрежите стават силно ненадеждни и протоколите като TCP/IP се използват да адаптират заявения комуникационен товар във възлите към текущите възможности на мрежата за трансфер без загуби на високата скорост. Аналогично е поведението на АТМ комутаторите дори когато те използват тясно свързани клъстери. Източникът не се уведомява, че неговият пакет е загубен, така че трябва да се използва механизъм с time-out за установяването на възникване на проблем.

В общия случай във вътрешносистемните комуникационни мрежи на паралелните компютри пакет, който се предава към пълен буфер, се блокира на място, а не се загубва. Това изисква „ръкостискане“ между изходния порт и входния порт в краищата на комуникационната връзка, т.е. управление на потока на ниво връзка. При продължителни конфликти трафикът се отблъсква от точката в мрежата, в която възникват конф-

ките към източниците, генериращи трафик в тази точка. Тези източници са подложени на обратно налягане в мрежата (тъй като тя отказва да приеме пакетите), което предизвиква трансфера на потока данни в мрежата да се забави до скорост, която позволява тясното място (bottleneck). Увеличаването на размера на буферите в комутаторите в мрежата осигурява възможност за по-дълго време конфликтите да не предизвикат обратно налягане към източника, но това също така води до забавяне на времето за престой на пакетите в опашките на буферите.

Един от основните проблеми при блокирането на съобщения в мрежата е, че обратното налягане може да окаже влияние върху трафика, което не е маршрутизиран за конфликтния изходен порт, наричан горещата точка. Ако трафикът, маршрутизиран за конфликтния изходен порт, повишава пропускателната му способност, то този трафик ще предизвика обратно налягане в мрежата. При продължителни конфликти обикновеното налягане ще се разпространи през дърво от канали по маршрутите до дестинацията. Това явление се нарича *дървовидно насищане*. Всеки изходен порт, пресичащ това дърво, също ще бъде забавен. При мрежи с маршрутизация от типа „дупка на червея“ алтернативата е да се изостави съобщението, като се информира източникът на колизията. Например в паралелния компютър BBN Butterfly източникът задържа опашката за съобщението, докато главата му достигне дестинацията (т.е. се концентрира канал) и ако възникне конфликт по маршрута, червеят се приключва обратно към източника. Този подход редуцира до голяма степен забавянето на дървовидното насищане.

Като се обобщи гореизложеното, може да се стигне до заключение, че във всички аспекти на мрежовия дизайн – пропускателна способност, връзките, топологията, стратегията за комутация, алгоритъма за маршрутизация и управлението на потока – комбинирано определят латентността на съобщението. Също така съществува тясна връзка между латентността и пропускателната способност на мрежата. В случаите, когато пропускателната способност при комуникация, изисквана от програмата, е ниска в сравнение с наличната пропускателна способност на мрежата, конфликтите ще бъдат малко, тенденцията за буферите е да са празни и латентността ще се поддържа ниска особено при маршрутизация с врязване. С повишаване на изискванията за пропускателната способност при комуникация, изисквана от програмата, латентността ще се увеличава поради зачестяване на конфликтите.

Важно е да се отбележи, че вътрешносистемните комуникационни връзки и на паралелните компютри са затворени системи с обратна връзка

от мрежата към източниците, генериращи трафика. Натоварването на мрежата зависи от скоростта на подаването на заявки от процесорните възли за комуникация, която от своя страна зависи от скоростта, с която мрежата реализира тази комуникация. Бързодействието на програмата зависи най-силно от латентността в случаите, когато възникват определени зависимости: програмата трябва да изчака завършването на четене или получаването на съобщение, за да продължи изпълнението си; докато се изчаква, натоварването на мрежата намалява и латентността също намалява. В паралелния компютър програмата се конкурира сама със себе си за комуникационните ресурси. Ако паралелният компютър работи в мултипрограмен режим, паралелните програми могат също да се конкурират помежду си. В този случай скоростта на подаване на заявки ще намалее с намаляването на скоростта на мрежата поради конфликти.

В заключение можем да обобщим, че комуникацията с ниска латентност е от критично значение за бързодействието на паралелните програми.

ПРОПУСКАТЕЛНА СПОСОБНОСТ. Пропускателната способност на вътрешносистемната комуникационна мрежа на паралелния компютър е критичен фактор за производителността на паралелните програми поради следните причини:

- Високата пропускателна способност на мрежата намалява заетостта на комуникационните канали;
- Високата пропускателна способност на мрежата намалява вероятността за конфликти;
- Отделните фази на паралелната програма могат да генерират комуникационен товар с голям обем данни, без да е необходимо да се изчака завършването на техния трансфер;
- При мрежите с комутация на пакети и маршрутизация с врязване е възможно да се осигури висока пропускателна способност дори при големи стойности на латентността поради факта, че поведението на мрежата се определя от функционирането на множество конвейери за трансфер.

Пропускателната способност на вътрешносистемната комуникационна мрежа на паралелния компютър може да се разглежда от две гледни точки:

- **Глобална (обща) пропускателната способност**, налична за всички възли в мрежата;
- **Локална (индивидуална) пропускателната способност**, налична за даден възел в мрежата.

При обем на глобалния комуникационен товар на паралелната про-
ма M байта и глобална (обща) пропускателната способност на мре-
а B байта за секунда, очевидно времето за комуникация е поне M/B
унди. От друга страна, ако цялата комуникация е от или към един и
взел, то тази оценка е твърде оптимистична. В този екстремален
най времето за комуникация ще се определя от локалната пропуска-
на способност на критичния възел.

Нека да разгледаме влиянието на спецификата на мрежовия ди-
върху локалната пропускателна способност на възела. Комуника-
ният канал представлява физическата връзка между комутаторите.
има ширина w и скорост на предаване на сигналите $f=1/\tau$, където τ е
мстраенето на мрежовия цикъл. Количеството данни, предадени през
ена връзка в рамките на мрежовия цикъл, е един фит (физическа
ница).

Ефективната локална пропускателна способност се дефинира като
от физическата пропускателна способност на комуникационната
зка в зависимост от ефективността на кодирането на пакета: $b \cdot n / (n + n_e)$,
ето b е физическата пропускателна способност на комуникационната
зка, n е дължината на съобщението, а n_e са допълнителните символи
лика на съобщението. При положение, че комутаторът блокира паке-
ри закъснение за вземане на решение при маршрутизация за Δ цикъ-
ефективната локална пропускателна способност на възела още се
алява до $b \cdot n / (n + n_e + w \cdot \Delta)$. В случая $w \cdot \Delta$ е времето за престой на
бщението поради блокирането на връзката. Следователно аспекти-
а мрежовия дизайн като форматът на пакета и алгоритъм за марш-
зация оказват влияние върху локалната пропускателна способност
възела.

При възникване на конфликти, когато множество възли комуники-
локалната пропускателна способност ще намалее още повече, а ла-
ността ще се увеличи. В случаите, когато множество възли изпращат
бщения към един и същ възел, във всички мрежи възникват конфлик-
между крайните точки на трансфера. Конфликти възникват обаче и
вътрешността на мрежата. Типът на мрежовата топология и алго-
ъмът за маршрутизация оказват влияние върху вероятността за възник-
на конфликти в мрежата.

В случаите на едновременна комуникация между множество възли
лезно да се изследва глобалната пропускателна способност на мре-
а, а не локалната пропускателна способност на индивидуалния възел.
популярният подход при определянето на глобалната пропускател-

на способност на мрежата е да се изследва *мрежова* ~~и~~ *пропускателна*
способност при бисекция, която се дефинира като сумата от пропуска-
телните способности на минималното множество канали, чието отстра-
няване води до разделянето на мрежата на две несвързани множества от
възли. При тази концепция разпределението на комуникационния товар
е напълно равномерно, тъй като половината от съобщенията трябва да
прекосят бисекцията във всяка посока. Пропускателна способност при
бисекция за индивидуалния възел варира значително при различните
мрежови топологии. Мрежовата пропускателна способност при бисек-
ция обаче не е изцяло задоволителен показател за глобалната мрежова
пропускателна способност, тъй като не във всички случаи комуникаци-
онният товар е равномерно разпределен в цялата паралелна машина. В
случаите, когато комуникацията е локализирана, а не равномерно раз-
пределена, мрежовата пропускателна способност при бисекция ще дове-
де до песимистична оценка на времето за комуникация. При локализирана
комуникация се използва алтернативният подход за оценка на глобална-
та пропускателна способност на мрежата, при която тя се определя като
сумарната пропускателна способност на връзките от възлите към мре-
жата. Подтекстът на този подход е, че съществува опасност вътрешната
структура на мрежата да не може да поддържа глобалната пропускател-
на способност на мрежата. Очевидно наличната глобална пропускател-
на способност на мрежата зависи от разпределението на комуникационния
товар и в частност от дължината на комуникационната дистанция на
пакета.

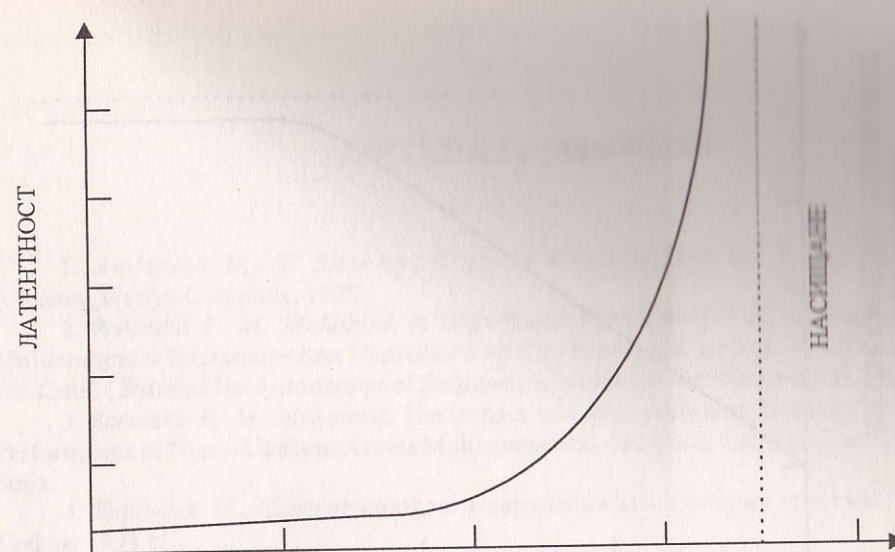
Глобалната пропускателна способност на всички канали (или връзки)
в мрежата се определя от броя на каналите C , умножен по пропускател-
ната способност на един канал, т.е. $C \cdot b$ байта за секунда, $C \cdot w$ бита за
цикъл, или C фита за цикъл. Ако всичките N хоста генерират по един
пакет на всеки M цикъла със средна дистанция за комуникация h , то то-
гава всеки пакет заема средно h канала за $l=n/w$ цикъла и глобалният
товар в мрежата е $N \cdot l \cdot h / M$ фита за цикъл. Средното използване на ко-
муникационните връзки е очевидно по-малко от 1 и е най-малко:

$$\rho = M \cdot C / N \cdot l \cdot h \quad (5.17)$$

Един начин за третиране на този параметър е, че броят на връзките
за даден възел C/N отразява средната пропускателна способност при
комуникация към всеки възел (брой фитове за един цикъл за един възел).
Тази пропускателна способност се консумира право пропорционално на
дистанцията за маршрутизация и размера на съобщението. Броят на

ките за всеки възел е статично свойство на мрежовата топология. Дадената дистанция за маршрутизация зависи от топологията, алгоритъмът за маршрутизация, модела на разпределението на комуникационния ресурс, наложен от паралелната програма и стратегията за планиране на ресурсите на паралелния компютър за изпълнение на паралелната програма. Висока степен на локалност на комуникациите може да доведе до ниски стойности на h , докато случайното разпределение на комуникационния товар ще използва средната дистанция за комуникация. Съобщенията с максимална дистанция ще преминават през целия диаметър на мрежата. Размерът на съобщението се определя от поведението на програмата, както и от нивото на абстракция при комуникация.

На практика няколко фактора ограничават използването на канали. Възможно е комуникационният товар да не бъде балансиран спрямо всички връзки. Дори той да е балансиран, алгоритъмът за маршрутизация може да осигурява равномерно натоварване на всички връзки за даден набор комуникационни транзакции, изискван от паралелната програма. Даже в случаите, при които всички връзки се използват и комуникационният товар е балансиран във времето, е възможно да възникнат статистически вариации в комуникационния товар и конфликти за ресурси на ниско ниво. Всички тези фактори влияят върху точката на насищане на мрежата, която представлява общата пропускателна способност на мрежата, която мрежата може надеждно да осигури. Изискванията за пропускателна способност при комуникация, предявявана от процесорите, се нарича *предлагана пропускателна способност*. Когато тя е умерена, латентността остава ниска, а *предоставяната пропускателна способност* на мрежата се увеличава с нарастването на предлаганата пропускателна способност от паралелната програма. В дадена точка, обаче (фиг. 5.39), изискванията за по-висока пропускателна способност от предлаганата програма води само до увеличаване на конфликтите, което от своя страна значително увеличава латентността. В този режим мрежата функционира при максимално натоварване на комуникационните си ресурси и допълнителните заявки само увеличават дължината на опашките в буферите на комутаторите. В режима на насищане увеличаването на предлаганата пропускателна способност от паралелната програма не води до съответното увеличаване на предоставяната пропускателна способност на мрежата, т.е. тя остава постоянна. Целта при проектирането на паралелни компютри е да се осигури работен режим на вътрешносистемната мрежа далеч от точката на насищане или чрез осигуряване на много висока пропускателна способност при комуникация или чрез въвеждане на



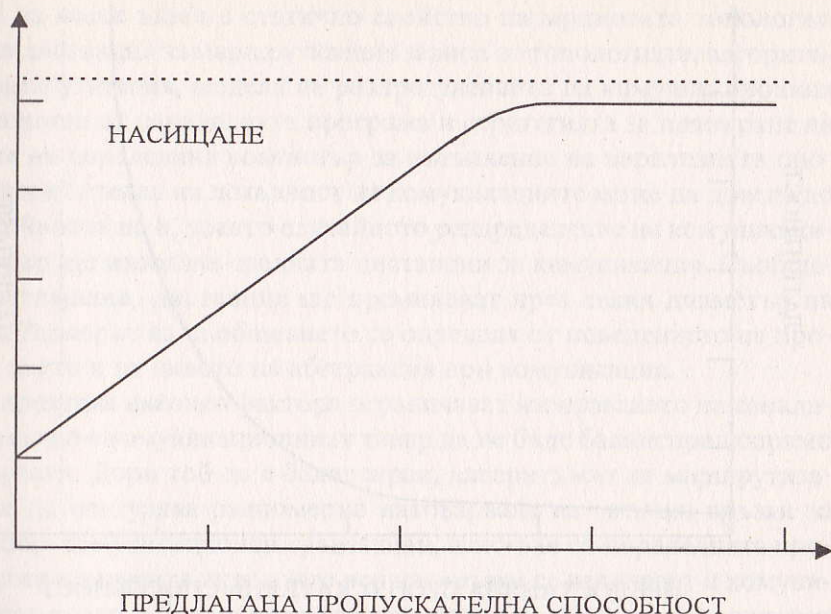
ПРЕДОСТАВЯНА ПРОПУСКАТЕЛНА СПОСОБНОСТ

Фиг. 5.39. Ефект на насищане върху латентността на вътрешносистемната комуникационна мрежа

механизми за ограничаване на изискванията за предлагана пропускателна способност на процесорите.

Поведението на мрежата, представено на фиг. 5.40, е характерно за всички системи, поддържащи опашки, но при допускането, че товарът не зависи от времето за отговор. Източниците генерират съобщения в системата с по-висока скорост, отколкото мрежата може да ги обслужи, така че се формират опашки с произволна дължина, предизвиквайки съответното нарастване на мрежовата латентност. С други думи направеният анализ се отнася за отворени системи, докато реално паралелните компютри представляват затворени системи. В мрежата и в мрежовите интерфейси са осигурени ограничени възможности за буфериране. Така в случаите, когато опашките се напълнят, източниците ще намалят интензивността на подаване на заявки до предоставяната скорост на обслужване на мрежата, тъй като няма да има възможност за поемане на нов пакет, докато не бъде отстранен от трафика поне един пакет.

Механизмите за управление на потока осъществяват това взаимодействие между източниците на трафик и възможностите на мрежата за обслужване на този трафик. От друга страна зависимостите в рамките на



Фиг. 5.40. Типично поведение на вътрешносистемната комуникационна мрежа при насищане

елната програма сами по себе си предопределят в някаква степен ение на потока от край до край, тъй като процесорът трябва да и дистанционната информация, за да може да продължи с по-на- ната обработка и едва тогава може да генерира нов комуникацио- афик. Важно е да се отбележи, че общ ресурс като мрежова връзка ке да се очаква да бъде използвана 100% дори за най-благоприят- риложения.

ЛИТЕРАТУРА

1. Anderson D., T. Shanley, Pentium Processor System Architecture, Addison_Wesley Company, 1995.
2. Borovska P., M. Mohamed, A High-Speed Packet Switch Architecture For Multicomputer Interconnection Networks With Cut-Through Wormhole Routing, 17th Int. Conf. „Systems for Automation of Engineering and Research“ SAER-2003, Varna.
3. Borovska P., M. Mohamed, The impact of Interconnection Topology on the Performance of Non – Uniform Access Multicomputers, Int. Conf. CompSysTech'2002, Sofia.
4. Боровска П., Суперкомпютри и паралелни компютърни системи, ТУ-София, 1993 г.
5. Боровски Б., П. Боровска, Архитектура на ЕИМ и микрокомпютри, „Техника“, София, 1992 г.
6. Denning P., „The Working Set Model For Program Behaviour“, Communications of the ACM, May 1968.
7. Denning P., „Working Sets past and present“, IEEE Trans. Soft. Eng., 1980, № 1, 684.
8. Dijkstra E., „Cooperating Sequential Processes“, Programming Languages, Academic Press, New York, 1968.
9. Dubey P., M. Flynn, „Branch Strategies: Modeling and Optimization“, IEEE Trans. On Computers, October 1991.
10. Flynn M., Computer Architecture Pipelined and Parallel Processor Design, Jones and Bartlett Publishers: Boston, Massachusetts.
11. Handy J., The Cache Memory Book, San Diego: Academic Press, 1993.
12. Hoare C., „Communicating Sequential Processors“, Conn. ACM, № 8, 1978.
13. Intel IA-32 Architecture Optimization, Reference Manual, 2003, <http://developer.intel.com>.
14. Kilburn T., D. Edwards, M. Lanigan, F. Sumner, „One-level Storage System“, Computer Structure Principles and Examples, 1968.
15. Leighton F., Introduction to Parallel Algorithms and Architectures, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1992.
16. Lilja D., „Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons“, ACM Computing Surveys, Sept. 1993.
17. Mak P., „Shared-Cache Cluster in a System with a Fully Shared Memory“, IBM Journal of Research and Development, July/Sept. 1997.
18. Mashey J., „CISC vs. RISC or what is RISC really“, USENET comp.arch.newsgroup, article 46782, Febr. 1995.

9. Motorola Inc., „Power PC Microprocessor Family: The Programming Elements for 32-bit Microprocessors“, Denver, 1997.
10. Patterson D., J. Hennessy, „Computer Organization and Design: The Hardware/Software Interface“, San Mateo, CA: Morgan Kaufmann, 1998.
11. Perleberg C., A. Smith, „Branch Target Buffer Design and Optimization“, Trans. On Computers, № 4, 1993, pp.396-412.
12. Романски Р., Компютърно моделиране, ТУ-София, 2002 г.
13. SCSI: Understanding the Small Computer Systems Interface, Englewood NJ: Prentice Hall, 1990.
14. Shanley T., D. Anderson, „PCI Systems Architecture“, Richardson, Mindshare 1995.
15. Shantley T., Pentium Pro and Pentium II System Architecture, Addison Wesley, 1997.
16. Shen J., High-Performance Microarchitecture Techniques, Intel labs, Processor Research Forum, Oct. 2002.
17. Teller P., „Translation-lookaside Buffer Consistency“, IEEE Computer, 1990, pp.26-36.
18. Walker W., H. Cragon, „Interrupt Processing in Concurrent Processors“, IEEE Computer, 1995, № 6, pp.36-46.
19. www.intel.com/info/hyperthreading
20. www.intel.com/products/processors/pentium4
21. www.tomshardware.com

ИНТЕРНЕТ КНИЖАРНИЦИ

www.mobilis.bg
www.bgbook.dir.bg



12,00 лв.

43 587.
Хеликон
12 - 12.00