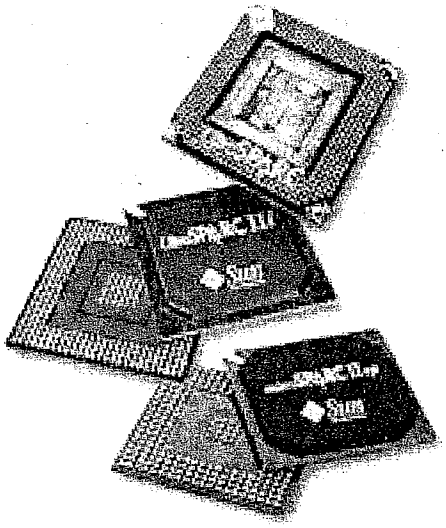
**Технически Университет - Варна**
**катедра "Изчислителна техника"**

Надежда Стефанова Рускова

# МИКРОПРОЦЕСОРНИ СИСТЕМИ



Ръководство за лабораторни упражнения

Варна
1999

ТУ - ВАРНА
KN1999/03204

# Предговор

Ръководството е предназначено за студенти от специалност "Компютърни системи и технологии". Изложеният в него материал разглежда основните архитектурни концепции и способите за изграждане на системи на базата на най-известните микропроцесорни фамилии - Ix86, MC680x0, PowerPC. В ръководството се преследват няколко цели - проследяване на основните направления на развитие и усъвършенстване на системата инструкции и организация на конвейера на инструкциите с цел постигане на висока производителност на микропроцесорите при изпълнение на различен клас приложения, изясняване на вградените възможности за защита на програми и данни и поддържане на основните функции на операционните системи при реализация на мултизадачен и многопотребителски режим, разглеждане на основните компоненти за построяване на ефективни подсистеми за обмен на информация с периферните устройства.

Основен обект на разглеждане в лабораторните упражнения са микропроцесорите Ix86 на фирмата Intel и компонентите за изграждане на системи на тяхна база. Организацията на лабораторните упражнения предполага познания в областта на архитектурата и системата инструкции на базовия микропроцесор от тази фамилия - I8086, а така също и техника за програмиране на асемблер за същия микропроцесор.

Всяко упражнение е придружено с примери и примерни задачи за допълнителна разработка. Част от задачите са с повишена сложност и са адресирани към студенти, които желаят да задълбочат познанията си в областта на проектирането на микропроцесорни системи и организацията на персоналните компютри. По-голямата част от упражненията са базирани на разработени от автора с помощта на дипломанти програмни средства - симулатор за I486, кросасемблер за MC68000, учебна мултизадачна среда, информационни системи за инструкциите и изводите на изучаваните микропроцесори и изследваните периферни интегрални схеми.

Авторът изказва благодарност на рецензентите доц. д-р Валтер Станчев и доц. д-р Анатолий Антонов за полезните забележки и предложения към оформянето на ръководството.

# 1. Архитектурни особености на фамилията микропроцесори Ix86

## 1. Развитие на архитектурата на Ix86

Приемствеността в архитектурата на микропроцесорите на фирмата Intel започва от I8086, инструкциите на който съставят ядрото на системите инструкции на следващите поколения микропроцесори. I8086 е 16 разреден микропроцесор (МП), чиито програмно-достъпни вътрешни регистри са със специално предназначение. В него е реализирана сегментация на паметта, като ефективният адрес (EA) се изчислява по формулата EA = [SR] + displ, където [SR] е съдържанието на един от четирите сегментни регистъра, задаващ 20 разреден базов адрес на текущо активен сегмент в паметта, а displ е отместването в рамките на сегмента. Максималният размер на сегмент е 64 KB, а максималната адресируема памет 1 MB. Архитектурата на I8086 се реализира в т.н. "реален" режим на всички следващи микропроцесори Ix86.

В МП I286 е въведен "защитен" режим на работа, предоставящ средства за поддържане на многозадачни операционни системи. Тези средства включват: 4 привилигировани нива на изпълнение на задачите; права за достъп до сегментите; апаратно превключване на задачи; виртуална памет. Механизмите за защита на програми и данни и виртуализация на паметта се реализират на базата на съответни атрибути на сегментите, задавани посредством дескриптори (описатели), които се съхраняват в дескрипторни таблици. Достъпът до даден дескриптор, а чрез него и до описвания сегмент се осъществява посредством сегментните регистри, чието съдържание се интерпретира като селектор (указател) в дескрипторните таблици.

В архитектурата на I386 е включен нов "виртуален" режим, позволяващ увеличаване на производителността при изпълнение на програми, написани за I8086 и I286. Разширените 32 разредни вътрешни регистри придобиват свойствата на регистри с общо предназначение, т.е. те се използват както за операнди, така и при изчисляване на адреси. Въведено е странициране (4K страница) като допълнителен механизъм за виртуализация на паметта, който за операционните системи е по-ефективен от сегментирането и е невидим за потребителските програми. Системата инструкции е разширена с 32 разредни операнди и адресни компоненти, както и с нови операции (за манипулация с битове, далечни условни преходи, работа със системните регистри и др.).

По-нататъшното увеличаване на производителността се постига чрез реализация в кристала на микропроцесорите на концепциите за паралелна обработка. В I486 паралелно с петстепенния конвейер на инструкциите, изпълняващ по една инструкция за такт на вътрешната синхронизация, работят: устройство за управление на интерфейса, реализиращо обръщенията към външната среда; устройство за управление на паметта (MMU), реализиращо механизмите за сегментиране, странициране и контрол; интегрирано в кристала устройство за плаваща запетая (FPU). Освен това включената в кристала 8K кеш памет (L1 кеш) значително повишава ритмичността на конвейера. Реализирано е пакетно предаване по външната шина.

Pentium е първият частично суперскаларен процесор във фамилията Intel, изпълняващ по две инструкции на такт върху два паралелни конвейера (U и V), с възможност за динамично предсказване на преходите и разделена кеш памет за инструкции и за данни (по 8K). Добавени са средства за работа в мултипроцесорна конфигурация: специален протокол за поддържане консистентността на вътрешните кеш памети (MESI - modified, exclusive, shared, invalid) и интелигентен програмируем контролер на прекъсванията (APIC).

В Pentium Pro е повишена степента на суперскаларност, като максимално се изпълняват по три инструкции на такт. Тъй като постигането на висока степен на суперскаларност е характерно за RISC архитектурите, в Pentium Pro е реализирано RISC ядро. Нормалните CISC инструкции на процесора вътрешно се декодират в последователност от триоперандни RISC микроинструкции (*mops*) с регистрова адресация, имащи формат от вида: ***op reg,reg reg***, където ***op*** - операция, ***reg*** - регистър. Тези инструкции се включват в пул, откъдето се избират за изпълнение не по реда си в програмата, а в зависимост от наличността на необходимите им операнди. Едновременно за изпълнение могат да бъдат стартирани до 5 mops върху пет паралелно работещи изпълнителни устройства - 2 устройства за целочислени операнди (IU), две устройства за плаваща запетая и устройства за запис и четене от паметта. По този начин на потребителско ниво Pentium Pro е CISC процесор, докато вътрешната му архитектура е суперскаларен RISC, управляван от потока операнди, с динамично предсказване и спекулативно изпълнение на преходите. В корпуса на Pentium Pro е включена и 256K кеш памет от второ ниво (L2 кеш), която дава възможност за едновременен

достъп по 4 паралелни адреса. Предаванията по външната шина са организирани като транзакти с разделно управление на адресна и даннова фаза.

Pentium Pro, Pentium II и Pentium III принадлежат на новата фамилия микропроцесори на фирмата Intel, наречена P6. В Pentium II и Pentium III е използвана същата суперскаларна микроархитектура, както в Pentium Pro, разширена с възможност за реализация на нови инструкции за мултимедийни приложения (технология MMX). Въведени са няколко режима с намалена консумация на енергия. Pentium II позволява директно работа в двупроцесорна конфигурация, а Pentium III - четирипроцесорна. Таблица 1.1 илюстрира в синтезиран вид развитието на ключовите характеристики на микропроцесорите Ix86.

**Таблица 1.1**

| Процесор | Година на появяване | Max честота Mhz | Размер на вътр. регистри | Размер на външна даннова шина | Max адресно пр-во | Вътрешна кеш | Вграден FPU | MMX инструкции | Суперскаларна микроархитектура |
|---|---|---|---|---|---|---|---|---|---|
| 8086 | 1978 | 8 | 16 | 16 | 1 MB | - | - | - | - |
| i286 | 1982 | 12 | 16 | 16 | 16 MB | - | - | - | - |
| i386 | 1985 | 20 | 32 | 32 | 4 GB | 8 KB L1 | - | - | - |
| i486DX | 1989 | 50 | 32 | 32 | 4 GB | 16 KB L1 | да | - | - |
| Pentium | 1993 | 166 | 32 | 64 | 4 GB | 16 KB L1 | да | - | да |
| Pentium MMX | 1993 | 200 | 32 | 64 | 4 GB | 16 KB L1 | да | да | да |
| Pentium Pro | 1995 | 200 | 32 | 64 | 64 GB | 2x8 KB L1 512 KB L2 | да | - | да |
| Pentium II | 1997 | 333 | 32 | 64 | 64 GB | 2x16 KB L1 512 KB, 1MB, 2MB L2 | да | да | да |
| Pentium III | 1999 | 500 | 32 | 64 | 64 GB | 2x16 KB L1 512 KB, 1MB, 2MB L2 | да | 70 | да |

## 2. Микроархитектура на PentiumII

RISC ядрото на процесора е реализирано чрез три независими вътрешни устройства, комуникиращи си чрез пул на инструкциите (фиг.1.1). Това позволява инструкциите да се стартират за изпълнение в произволен ред, но да завършват по реда си в програмата.

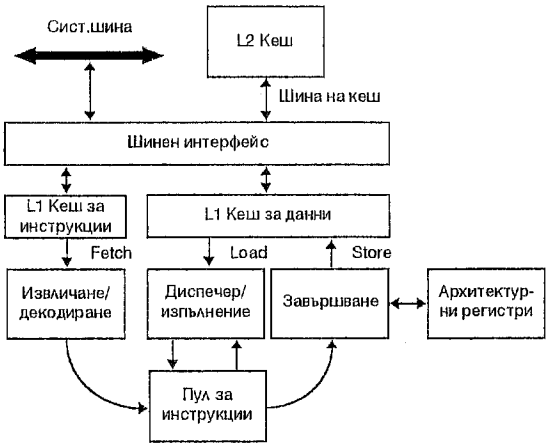
Въвеждането на пул между традиционните фази на конвейера "извличане" и "изпълнение" цели отваряне на широк прозорец от инструкции за осигуряване на по-голяма видимост напред в програмата, което позволява да се реализира по-оптимална стратегия за планиране на изпълнението.

Например нека в пула текущо се съдържа следният програмен фрагмент от псевдо RISC инструкции:

```
r1 <- mem(r0)      /* I1 */
r2 <- r1 + r2      /* I2 */
r5 <- r5 + 1       /* I3 */
r6 <- r6 - r3      /* I4 */
```

Ако при първата инструкция I1, зареждаща регистър r1 с операнд по адрес от регистъра r0, се получи непопадение в кеша, то при традиционното последователно изпълнение процесорът трябва да изчака четенето на операнда от външната памет и едва тогава да премине към изпълнението на следващата инструкция I2. Вместо това в процесорите P6 се проверяват следващите инструкции за готовност за изпълнение. Така, I2 не може да бъде изпълнена, тъй като зависи от резултата на I1. I3 и I4 обаче не зависят от предидущите инструкции и могат да бъдат изпълнени веднага без да чакат реда си. Резултатите от тяхното изпълнение обаче не могат веднага да бъдат отразени в архитектурните регистри, тъй като тяхното съдържание трябва във всеки един момент да отразява

състоянието на изпълнение на програмата. Вместо това резултатите се записват в пула, където изпълнените инструкции очакват реда си на завършване.




Фиг.1.1. Основна вътрешна структура на Pentium II.

Запълването на пула с mops се извършва от устройството "извличане/декодиране", а преглеждането за изпълними инструкции (типично 20 - 30 от текущото значение на програмния брояч) - от устройството "диспечер/изпълнение". В тези рамки устройството "извличане/ декодиране" трябва правилно да предсказва средно 5 прехода, за да се поддържа заетостта на процесора. Прехвърлянето на резултатите в архитектурните регистри и отстраняването на инструкциите от пула по реда им в програмата се изпълнява от завършващото устройство.

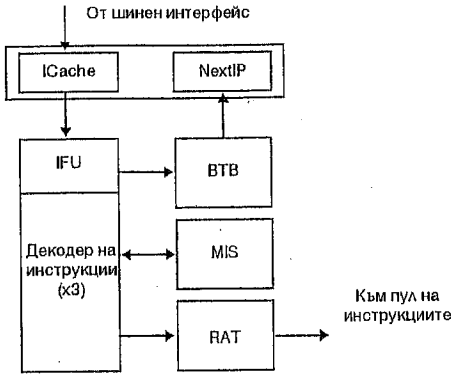
Устройствата "извличане/декодиране", "диспечер/изпълнение" и "завършване" представляват основните части на конвейера и се свързват с външната среда посредством устройството "шинен интерфейс".

### 2.1. Извличане/декодиране

Устройството "извличане/декодиране" (фиг. 1.2) получава от кеша на инструкции (ICache) 32 байта, указвани от указателя към следващата инструкция (NextIP). Устройството за предварително извличане (Instruction Feach Unit -IFU) формира три степени от конвейера на инструкциите, на които предварително извлечените байтове се изравняват на границата на 16 байта и се отбелязват началото и края на съдържащите се в тях архитектурни инструкции. Три паралелни декодера разпознават отделните архитектурни инструкции и ги преобразуват в триоперандни mops. По-голямата част от инструкциите се представят с една mop, някои изискват до 4 mops, а най-сложните - микрокод, представляващ препрограмируема последователност от микрооперации. Микрокодът се извлича от устройството за микропрограмно управление (Microcode Instruction Sequencer - MS). Генерираните и подредени mops се подават към таблицата на синонимните регистри (Register Alias Table - RAT), където обръщенията към архитектурните регистри се преобразуват в обръщения към физически регистри от вътрешен регистров файл (синонимни регистри). Регистровият файл съдържа 40 регистъра с общо предназначение. Те се използват както за операнди с фиксирана, така и за операнди с плаваща запетая. В RAT към микрооперациите се добавя информация за състояние, определяща готовността на операндите. Към mops за преход устройството за предсказване (Branch Target Buffer - BTB) добавя предсказания и алтернативен адреси на прехода. Извличането на следващите инструкции от кеша продължава от предсказания адрес, с чиято стойност се зарежда NextIP. Тези инструкции се отбелязват като спекулативни. След RAT mops се записват в пула на инструкциите (Reorder Buffer - ROB).

BTB съхранява историята на вече срещани и изпълнени преходи и адресите, към които те са осъществени. На базата на тази динамично натрупвана информация при следващо появяване на същия преход BTB директно подава целевия му адрес към устройството за предварително извличане. При действителното изпълнение на инструкцията за преход в изпълнителното устройство

историята на прехода се обновява. Използвайки BTB, срещаните преди преходи могат динамично да се предсказват. BTB е с обем 512 елемента. Процесорите съдържат и стеков буфер за връщане (Return Stack Buffer - RSB), който може да предсказва коректно адреса на връщане от процедури, извиквани от различни програмни обкръжения.




Фиг. 1.2. Устройство "извличане/декодиране"

Устройството "извличане/декодиране" заема 7 степени от цялостния конвейер на инструкциите. По степените на това устройство инструкциите преминават по реда си в програмата.

За един такт трите декодера могат:

- да декодират до три архитектурни инструкции;
- да генерират общо до 6 микрооперации, като при това декодер 0 може да преобразува една архитектурна инструкция до 4 mops, а декодери 1 и 2 по една инструкция до 1 mop (тройка инструкции 4-1-1);
- да декодират архитектурни инструкции с дължина до 7 байта.

Инструкции, преобразуващи се в повече от 4 mops или имащи дължина повече от 7 байта, изискват допълнителни тактове за декодиране. Съблюдаването при програмиране на разпределението на инструкциите по тройки 4-1-1 увеличава производителността на декодерите.

В зависимост от броя генерирани mops архитектурните инструкции могат да се групират по следния начин:

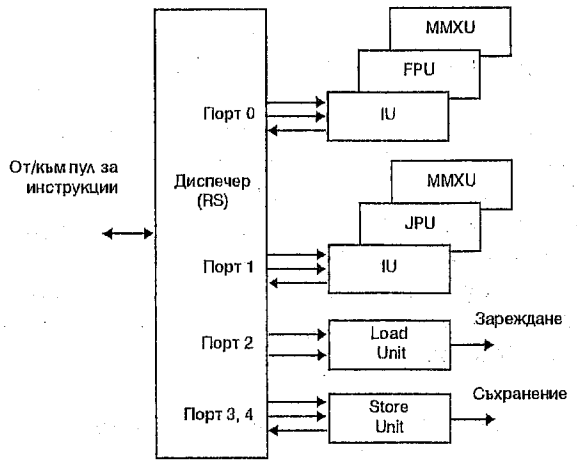
- обикновени инструкции от тип "регистър-регистър" - 1 mop;
- инструкции за зареждане - 1 mop;
- инструкции за съхраняване - 2 mops;
- обикновени инструкции от тип "четене-модификация" - 2 mops;
- обикновени инструкции от тип "регистър-памет" - 2 или 3 mops;
- обикновени инструкции от тип "четене-модификация-запис" - 4 mops;
- сложни инструкции - повече от 4 mops.

### 2.2. Диспечер/изпълнение

Диспечеризиращото устройство (Reservation Station - RS) избира от пула изпълнимите mops с достъпни операнди и ги стартира за изпълнение не по реда им в програмата, ако необходимото изпълнително устройство е свободно. Изпълнителните устройства са достъпни през 5 порта (фиг. 1.3), така че RS може да разпределя максимум по пет инструкции на такт, по една към всеки порт. Резултатът от изпълнението отново се записва в пула, където микрооперацията изчаква реда си на завършване. От RS изпълнителната фаза се разпаралелва на няколко конвейера, образувани от различни изпълнителни устройства.

Устройството за обработка на преходите (JPU) проверява правилността на предсказване на текущия преход. При правилно предсказан преход JPU отбелязва микрооперацията като завършила и я връща в пула. Извлечените след нея спекулативни mops се отбелязват като редовно изпълними. В противен случай, при несъвпадение на предсказания с изчисления преход, се сигнализира BTB, което

рестартира конвейера от алтернативния адрес. Спекулативно извлечените инструкции се изхвърлят от пула.




Фиг. 1.3. Устройство "диспечер/изпълнение"

### 2.3. Завършващо устройство

Завършващото устройство (RRF) непрекъснато проверява състоянието на микрооперациите в пула, открива изпълнените mops и ги отстранява според реда им в програмата. RRF завършва по 3 mops на такт, отчитайки появата на прекъсване, изключения, трасировка и неправилно предсказаните преходи. При отстраняване на завършилите mops RRF записва резултатите в архитектурните регистри или в паметта.

### 2.4. Шинен интерфейс

Устройството шинен интерфейс осъществява достъпа до паметта посредством външната системна шина. Системната шина е 64 разредна, организирана на базата на транзакции, т.е. с разделно управление на фазите адресация и предаване на данни. Това дава възможност за конвейерна организация на предаванията по нея. От друга страна устройството е тясно свързано посредством 64 разредна шина с L2 кеш. Тази шина също е организирана на базата на транзакции, работи със скоростта на МП и позволява до 4 паралелни достъпа до L2 кеш. Заявките към L2 кеш и паметта се буферират и планират на базата на подреждащ буфер, който може да преподреди обръщенията към паметта с цел повишаване на производителността. Възможно е спекулативно изпълнение на операциите четене, докато операциите запис се изпълняват по реда на възникването си, тъй като няма транспарентен механизъм за тяхната отмяна.

**Въпроси и задачи:**

1. Какви са предимствата на динамичната микроархитектура на семейството P6? Какво се печели от гледна точка на производителността на МП от динамичното предсказване на преходите?
2. Обяснете ползата от устройството за динамично предсказване на преходите за повишаване ефективността на процесорите.
3. Какви са особеностите на спекулативното извличане на инструкциите в динамичната архитектура на P6?
4. В какви примерни mops ще се декодира следната последователност от инструкции:

```
mov     eax, mem32
and     edx, 10
inul    eax, edx
mov     mem32, eax
```

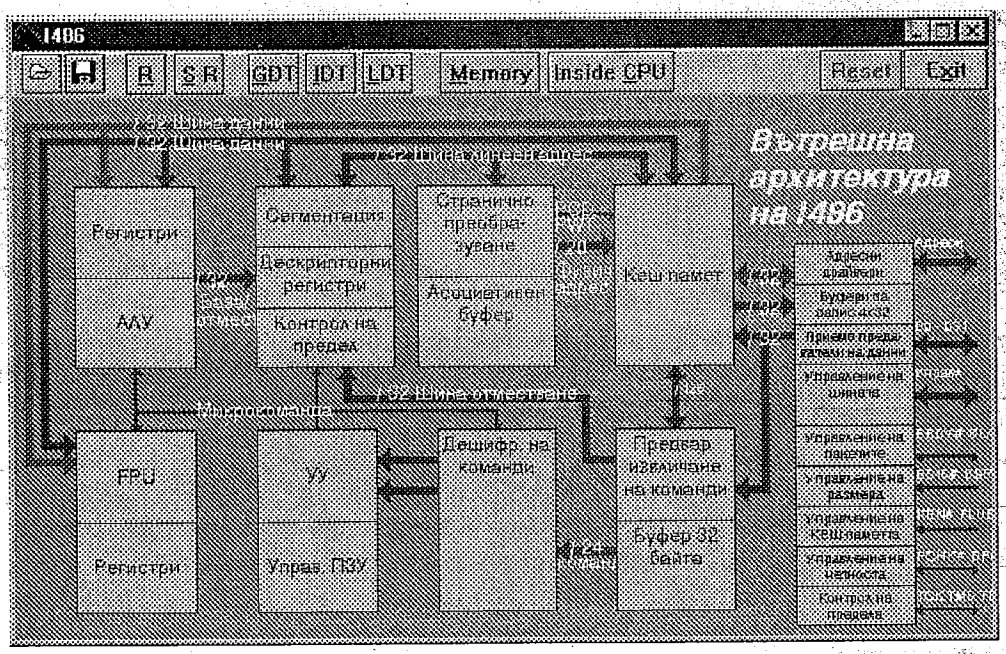
В какъв ред получените mops ще бъдат разпределени за изпълнение?

# 2. Изследване на архитектурните особености на I486

Микропроцесорът I486 е базов за фамилията 32 разредни микропроцесори Ix86. Той притежава всички основни архитектурни концепции, заложени в съвременните микропроцесори на фирмата Intel. Благодарение на своята относителна простота, I486 е подходящ за обучение по архитектурата Intel. В катедра "Изчислителна техника" на ТУ-Варна е разработен симулатор на този МП (sim486), който дава възможност за проследяване на работата му с точност до 1 такт на вътрешния тактов генератор. Симулаторът sim486 предоставя възможност за наблюдение на процесите, протичащи във вътрешните устройства на МП при изпълнение на инструкциите на дадена програма. Той може да се използва за изучаване на особеностите на конвейера на инструкциите, методите за оптимизация на програмния код, средствата и механизмите на многозадачния режим на работа, действието на различните видове защити и привилегии. В симулатора са включени богати възможности за динамична визуализация и интерактивна промяна, както на съдържанието на регистрите на МП, така и на съдържанието на симулираната памет.

### 1. Симулация на вътрешните компоненти на микропроцесора

Най-съществената особеност на sim486 е симулацията на паралелната работа на вътрешните компоненти (устройства) на МП, благодарение на което е възможно наблюдаване на отделните фази (степени) от изпълнението на инструкциите. Вътрешната структура на процесора I486 се визуализира в основния прозорец на симулатора, който се извежда след неговото стартиране (фиг. 2.1).




Фиг. 2.1. Основен прозорец на симулатора.

В състава на процесора влизат девет вътрешни функционални устройства: шинен интерфейс; вътрешна кеш памет; устройство за предварително извличане на инструкции; двустепенен дешифратор на инструкции; управляващо устройство (УУ); устройство за целочислени операции (АЛУ); устройство с плаваща запетая (FPU); устройство за сегментация; устройство за странично преобразуване на адреса. Устройствата организират петстепенен конвейер за изпълнение на инструкциите.

**Устройство за предварително извличане на инструкции.** Устройството за предварително извличане (IFU) чете 16 байтов блок или от вътрешната кеш памет за един такт или в случай на отсъствие в кеш на заявения блок от външната памет за по-голям брой тактове. Четените от

външната памет блокове едновременно се подават в кеш и IFU. В устройството се поддържа опашка от инструкции, реализирана на базата на два 16 байтови буфера. Те се изчистват, когато се наруши естественият ред на изпълнение на инструкциите, например при преходи, превключване на задачи, изключение и прекъсване.

**Устройство за дешифрация.** Дешифриращото устройство извлича от буферите на IFU поредната инструкция, разпознава и обработва отделните полета от нейния формат и определя входната точка за изпълнението ѝ в управляващите микропрограми на УУ. Константната част на инструкцията (отместването) се подава в устройството за сегментация, където участва в паралелното изчисление на адреса на операндите. Дешифраторът реализира две степени на конвейера Dec.I и Dec.II и се нулира при всяко нулиране на IFU.

**Устройство за управление.** Устройството за управление съдържа микропрограмите за изпълнение на инструкциите. То получава от дешифратора входната точка в микрокода за интерпретация на поредната инструкцията. УУ управлява целочисленото устройство, устройството с плаваща запетая и сегментацията на адреса. Много инструкции имат по една микрокоманда и се изпълняват само за един такт. За сложните инструкции са необходими няколко такта.

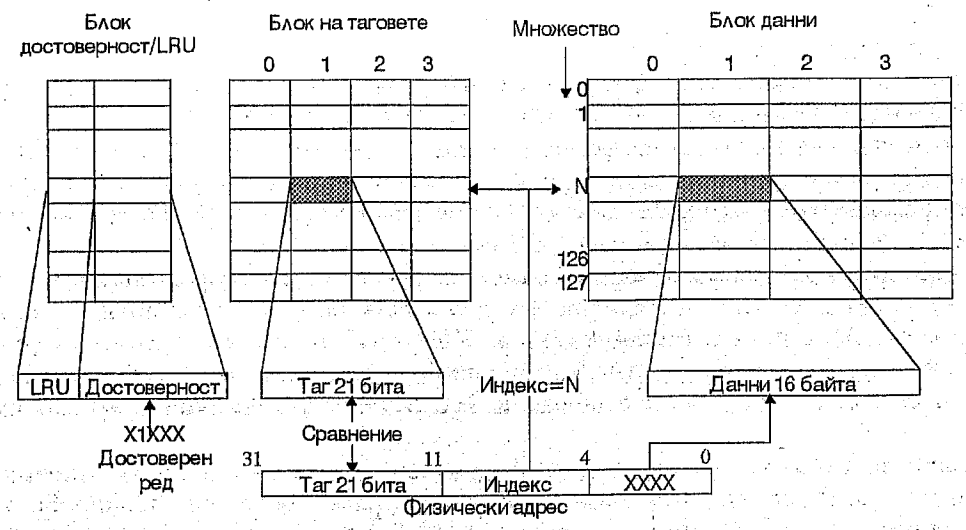
**Устройство за целочислени операции.** Устройството изпълнява всички аритметични и логически инструкции. В него са разположени осемте програмно достъпни 32-битови регистъра с общо предназначение, няколко специализирани регистъра, аритметико - логическо устройство и регистър за паралелно преместване. Инструкциите за зареждане, съхранение, събиране, логически операции и преместване се изпълняват за един такт. Целочисленото устройство и устройството с плаваща запетая са съединени с две 32-битови двупосочни шини, които се използват съвместно за предаване на 64-битови операнди. Тези шини свързват операционните устройства с кеш паметта. Съдържанието в регистрите с общо предназначение се подава по отделна 32 битова шина в устройството за сегментация за изчисляване на ефективните адреси.

**Устройство с плаваща запетая.** Вграденото в МП устройство има същата система инструкции и набор регистри, както и математическия копроцесор 80387. Това устройство съдържа регистров стек от осем 80-разредни регистъра и специализирани схеми за интерпретации на 32/64/80-разредни числа с плаваща запетая, определени от стандарт IEEE754.

**Устройство за управление на паметта (MMU).** Устройството се състои от две секции - устройство за сегментно и устройство за странично преобразуване на адреса, получаван от дешифратора на инструкции. Работата на устройството за сегментиране се базира на сегментните регистри на процесора (CS, DS, SS, ES, FS, GS) и данновите структури за съхраняване на атрибутите на сегментите - (глобална дескрипторна таблица (GDT), локална дескрипторна таблица (LDT) и дескрипторна таблица за прекъсвания (IDT), указвани от регистрите GDTR, LDTR и IDTR съответно. Устройството за странично преобразуване генерира физически адреси на базата на каталог и таблици на страници. Каталогът на страниците е достъпен през управляващ регистър CR3. Описателите на текущо използваните страници се кешират в асоциативен буфер TLB. Обикновено генерацията на адреса заема един такт, освен в случаите, когато се използва индексна адресация, тогава са нужни два такта. Подробно преобразуването на адреса се разглежда в следващите упражнения.

**Шинен интерфейс.** Шинният интерфейс отчита приоритета при обръщение към шината и координира предаването на данните, осъществява предварителното извличане на инструкции и изпълнява функции по съгласуване на действията на вътрешните устройства на процесора и външната система. Той взаимодейства с кеш паметта и устройството за предварително извличане по три 32-битови шини. С външните схеми шинният интерфейс взаимодейства формирайки сигналите на шината на процесора. Включените в устройството буфери за запис осигуряват буфериране до четири записа, което позволява продължаване на вътрешните операции без да е необходимо изчакване завършването на циклите за запис по шината на процесора.

**Кеш памет.** Вътрешната кеш памет съдържа копия на последно прочетени блокове с данни и инструкции. Четенето на данни, които вече се намират в кеш паметта, се изпълнява за един такт, като не е необходимо обръщение по шината към външната памет. При отсъствие на данните в кеш паметта те се прочитат от външната памет и се зареждат в кеш за едно или няколко 16-байтни кешируеми предавания по шината. Ако се формира заявка за запис в област, намираща се в кеш, то се извършва обновяване на кеш и запис в основната памет. Кеш паметта има четириканална асоциативна организация (фиг. 2.2).




Фиг. 2.2. Структура на вътрешната кеш памет

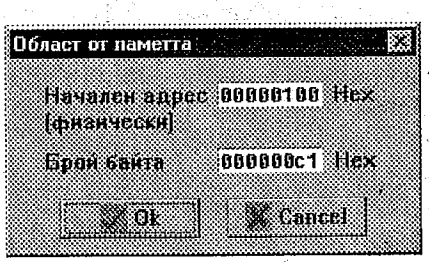
Вътрешната кеш памет се състои от три блока - блок данни, блок тагове, блок достоверност/LRU. Блокът данни от 8Kb е разделен на четири банки, всяка от които съдържа 128 реда. Всеки ред съдържа 16 съседни байта от паметта с начален адрес, кратен на 16. Общо блокът данни съдържа 128 множества по 4 реда. На всеки ред съответства запис в блока тагове. Адресацията на кеш паметта се осъществява чрез разделяне на 32 разредния физическия адрес на 3 части. Седем бита с номера 4-10 определят номер на множество от 128-те множества. Старшите 21 бита са таг (признак), който се сравнява с таговете в четирите полета на индексираното множество и показва дали 16 байтовото поле съдържа данни от този физически адрес. Младшите четири бита на физическия адрес задават байт в избрания ред.

На всяко множество в блока достоверност/LRU съотвества запис, състоящ се от 3 разредно поле LRU и 4 разредно поле достоверност. Всеки бит за достоверност показва дали данните, намиращи се в съответния ред, са достоверни или не. Полето LRU се използва от стратегията за замяна на ред в съответното множество.

## 2. Интерфейс и основни възможности на симулатора

Функционалните бутони, разположени на основния прозорец (фиг. 2.1), имат следното предназначение (от ляво на дясно):

• **зареждане на файл в оперативната памет на симулирания процесор.** При активиране на този бутон се визуализира прозорец за отваряне на файл, имащ стандартна за Windows форма. За изпълнение от sim486 могат да бъдат зареждани файлове с разширение .com или .exe. След избиране на желания файл, се визуализира прозорец, в който се задава началният адрес на зареждане на програмата в паметта на симулирания МП и обемът й в брой байтове,

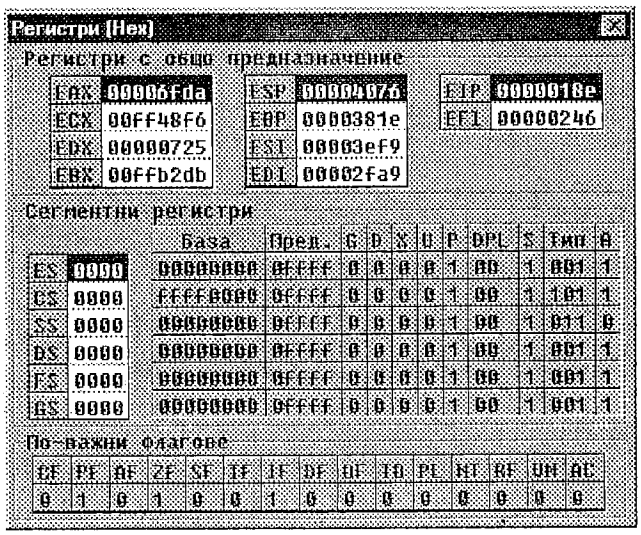



Фиг. 2.3. Прозорец за задаване на началния адрес на зареждане на програмата

При натискане на бутона "Cancel" файлът не се прочита, а на бутона "OK" се прави проверка за коректност на данните. При грешни данни се извеждат съответните съобщения.

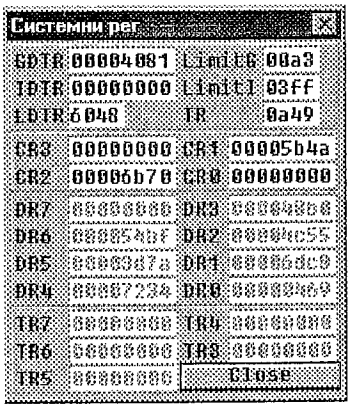
• **запис на блок от паметта във файл** - процедурата по записване на произволен участък от симулираната памет във файл изисква задаване на началния адрес на блока в паметта и неговия размер в байтове, след което се визуализира стандартен Windows прозорец за запис.

• **програмно достъпни регистри (R)** - извежда прозорец, визуализиращ текущото съдържание на програмно достъпните регистри на процесора. Показани са и съдържанията на кеш регистрите към сегментните регистри, в които са кеширани дескрипторите на текущо използваните сегменти. По-важните флагове на регистъра EFLAGS се визуализират в отделни полета. При необходимост от промяна на съдържанието на някой регистър, той се избира с мишката, след което от клавиатурата се въвежда нужното значение.




Фиг. 2.4. Прозорец за съдържанието на програмно достъпните регистри

• **системни регистри (SR)** - визуализира прозорец с текущото значение на системните регистри на I486. Промяна на съдържанието на системен регистър се извършва по аналогичен начин, както при програмно достъпните регистри. Системните регистри са описани в упражнение 5.




Фиг. 2.5. Прозорец за съдържанието на системните регистри

• **визуализация на дескрипторните таблици.** Бутони GDT, LDT и IDT визуализират съдържанието на областите от паметта, заети от съответните дескрипторни таблици. Тези дескрипторни таблици се използват при работа на процесора в защитен режим и подробно се разглеждат в упражнение 5. Форматът на съхраняваната в тях информация е еднотипен. На фиг. 2.6

е показано съдържанието на отделните полета на дескрипторите в дескрипторна таблица. Във втория ред на прозореца се извежда информация за местоположението (базов адрес) и заемания обем (предел) в паметта на съответната таблица.

Глобална дескрипторна таблица

Базов адрес 0000259c h  Лимит 00204 h

| № | База | Лимит | G | D | X | U | P | DPL | S | Тип | A | Вид |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| h | h | h | b | b | b | b | b | b | b | b | b | Text |
| 0000 | e0ffff7b | 9e718 | 1 | 0 | 0 | 0 | 0 | 11 | 1 | 001 | 0 | Данни R/W |
| 0001 | e0ffff5a | de718 | 1 | 0 | 0 | 0 | 0 | 11 | 1 | 000 | 1 | Данни R |
| 0002 | 004a4e73 | ce018 | 1 | 0 | 0 | 1 | 0 | 00 | 1 | 110 | 1 | Подч. E |
| 0003 | ff18e039 | ce0ff | 1 | 0 | 0 | 1 | 1 | 11 | 0 | 111 | 1 | Шлюз Oc.Сл |
| 0004 | e018e039 | 8e0ff | 0 | 0 | 0 | 1 | 1 | 11 | 0 | 000 | 0 | ??? |
| 0005 | e018e018 | a45ae | 0 | 1 | 0 | 0 | 0 | 10 | 1 | 011 | 0 | Стек R/W |

Фиг. 2.6. Прозорец за визуализация на дескрипторна таблица

• **Визуализация на съдържанието на област от паметта (memory)** - извежда екран със съдържанието на област от симулираната памет. Чрез позициониране на мишката върху необходимия сегментен регистър на втория ред на прозореца може да се избира нужният сегмент. За избрания сегмент се визуализират базовият адрес и размерът му в байтове. Съдържанието на клетките от сегмента се разполага във вид на таблица по 8 последователни байта в ред. Левите три колони указват отместването по отношение на базовия адрес на сегмента (offset), линейния и физическия адреси на клетките в реда. Движението по паметта се осъществява чрез съответните бутони в дясната част на прозореца. Съдържанието на клетка (байт) може да бъде променено чрез избирането ѝ с мишката и въвеждане на ново значение от клавиатурата.

Памет 102400 ($19000) байта

CS  DS  SS  ES  FS  GS

База  Лимит  Линеен адрес

00000000 0000FFFF  00000178

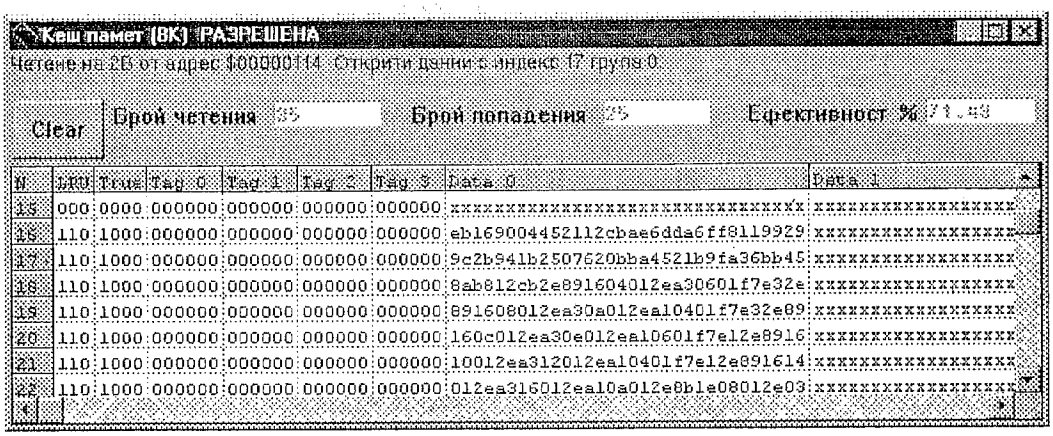
| Offset | Лин.адр. | Физ.адр. | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 |
|---|---|---|---|---|---|---|---|---|---|---|
| 00000100 | 00000100 | 00000100 | eb | 16 | 90 | 04 | 45 | 21 | 12 | cb |
| 00000108 | 00000108 | 00000108 | ae | 6d | da | 6f | f8 | 11 | 99 | 29 |
| 00000110 | 00000110 | 00000110 | 9c | 2b | 94 | 1b | 25 | 07 | 62 | 0b |
| 00000118 | 00000118 | 00000118 | ba | 45 | 21 | b9 | fa | 36 | bb | 45 |
| 00000120 | 00000120 | 00000120 | 8a | b8 | 12 | cb | 2e | 89 | 16 | 04 |
| 00000128 | 00000128 | 00000128 | 01 | 2e | a3 | 06 | 01 | f7 | e3 | 2e |
| 00000130 | 00000130 | 00000130 | 89 | 16 | 08 | 01 | 2e | a3 | 0a | 01 |
| 00000138 | 00000138 | 00000138 | 2e | a1 | 04 | 01 | f7 | e3 | 2e | 89 |
| 00000140 | 00000140 | 00000140 | 16 | 0c | 01 | 2e | a3 | 0e | 01 | 2e |
| 00000148 | 00000148 | 00000148 | a1 | 06 | 01 | f7 | e1 | 2e | 89 | 16 |
| 00000150 | 00000150 | 00000150 | 10 | 01 | 2e | a3 | 12 | 01 | 2e | a1 |
| 00000158 | 00000158 | 00000158 | 04 | 01 | f7 | e1 | 2e | 89 | 16 | 14 |
| 00000160 | 00000160 | 00000160 | 01 | 2e | a3 | 16 | 01 | 2e | a1 | 0a |
| 00000168 | 00000168 | 00000168 | 01 | 2e | 8b | 1e | 08 | 01 | 2e | 03 |
| 00000170 | 00000170 | 00000170 | 1e | 0e | 01 | 2e | 83 | 16 | 0c | 01 |
| 00000178 | 00000178 | 00000178 | 00 | 2e | 03 | 1e | 12 | 01 | 2e | 8b |

Фиг. 2.7. Прозорец за съдържанието на сегмент от паметта

• **Reset** - бутонът Reset рестартира симулирания процесор и привежда вътрешните компоненти в начално състояние;

• **Exit** - изход от симулатора.

Вътрешните устройства на процесора са достъпни и чрез натискане на десния бутон на мишката върху съответния панел, изобразяващ устройството в главния прозорец на симулатора. По този начин например може да се визуализира текущото съдържание на на буфера на устройството за страницираме (TLB) или на кеш паметта с информация за нейната ефективност (фиг. 2.8).

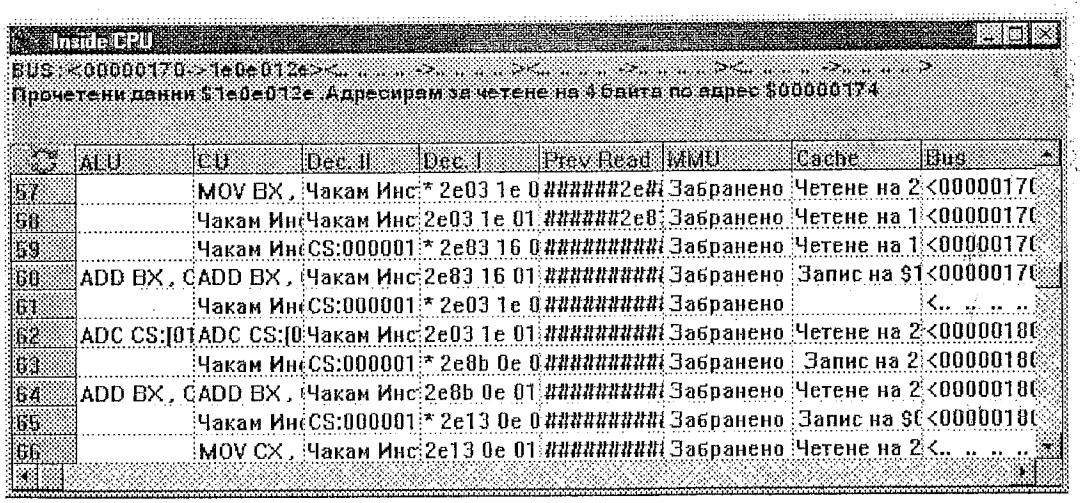



Фиг. 2.8. Визуализация на съдържанието на кеш паметта.

Съдържанието на кеш паметта се извежда във вид на таблица, редовете на която съответстват на отделните множества. Номерът на множеството се извежда в началото на всеки ред. За всяко множество се визуализират съответно значението на полетата достоверност/LRU, четирите тага и четирите 16 байтови реда данни. С бутоните стрелки в долната и дясната част на таблицата могат да се разкриват определени участъци на кеш паметта. Над таблицата се описват действията, които се извършват от устройството за управление на кеш при текущата операция четене или запис, както и статистика за нейната ефективност.

### 3. Симулация на конвейера на процесора при изпълнение на програма

В режим "Inside CPU", стартиран от едноименния бутон на главния прозорец, заредената програма се изпълнява потактово и се визуализира преминаването на инструкциите по степените на конвейера. Прозорецът в този режим е показан на фиг. 2.9.




Фиг. 2.9. Визуализация на конвейера на инструкциите

В прозореца се извежда таблица, всеки ред на която отразява състоянието на конвейера на процесора в един такт. Поредният номер на такта се извежда в най-лявата колона. В горния ляв ъгъл на таблицата се намира бутонът за потактово изпълнение на инструкциите, а в горния край на прозореца се извежда пълна информация за степента на конвейера, върху която е позиционирана мишката по време на даден такт. Например при позициониране върху устройството за предварително извличане на инструкции от фиг.2.9 (PrevRead) се визуализира съдържанието на неговите буфери, а

при позициониране върху шинния интерфейс (Bus) - четирите буфера за запис. Престоите в степените на конвейера се отбелязват с израза "чакам инструкция". Отделните колони в таблицата могат да бъдат разширявани или свивани чрез позициониране на мишката върху съответната гранична линия и придвижването ѝ в нужната посока. Историята от развитието на изчислителния процес в конвейера може да се разглежда чрез бутоните стрелки в дясната част на прозореца.

**Въпроси и задачи:**

1. Напишете примерна програма за умножение на елементите на масив с константа. Асемблирайте програмата и я заредете за изпълнение в симулатора sim486. Проследете постъпковото ѝ изпълнение. Снемете състоянието на конвейера и вътрешните устройства на процесора във всеки такт.

2. Анализирайте задръжките в конвейера на инструкциите. Обяснете причините за тяхното появяване. Опитайте да оптимизирате програмата. Проверете със симулатора правилността на решението.

3. Изпълнете програмата със забранена и разрешена кеш памет. Какъв е ефектът от включването на кеш паметта?

Разред 30 на управляващия регистър CR0 (CD) разрешава (CD = 0) или забранява (CD = 1) вътрешната кеш памет, а разред 29 (NW) разрешава (NW = 0) или забранява (NW = 1) едновременният запис в кеш и външната памет. Комбинацията CD = 0 и NW = 1 е забранена, т.е. кеш паметта в I486 се използва само по метода едновременен запис, известен още като директен запис.

4. Наблюдавайте запълването на кеш паметта и формирането на битовете LRU. Открийте стратегията за запълване на полето LRU и замяна на редовете.

5. Ако една програма модифицира своя инструкция (самомодифициращ се код), то е възможно възникване на грешка. В случай че модифицируемата инструкция вече е в буферите на устройството за предварително извличане, то ще се изпълни старото ѝ копие. Какво трябва да се направи, за да се избегне това? Дайте пример и проверете правилността на решението.

# 3. Развитие на системата инструкции на архитектурата Intel

В упражнението се разглежда развитието на функционалните възможности на архитектурата Intel по сравнение с базовата система инструкции, заложена в микропроцесора I8086. Основно внимание се отделя на качествено новата технология MMX, предназначена за повишаване на производителността при мултимедийни продукти и Internet приложения.

### 1. Развитие на базовата система инструкции

Развитието на системата инструкции на фамилиите процесори на Intel се осъществява в няколко посоки:

- увеличаване размера на операндите и възможност за работа с операнди с различна дължина - байт (8 разреда), дума (16 разреда), двойна дума (32 разреда), четворна дума (64 разреда). Добавяне на нови типове операнди;
- разширение на функционалните възможности за:
  - поддържане на многозадачни и многопотребителски операционни системи;
  - поддържане на езиците от високо ниво;
  - поддържане на многопроцесорна конфигурация;
  - ускоряване на обработката при често изпълнявани приложения (например мултимедийни приложения - MMX инструкции);
  - оптимизация на кода, например инструкции за избягване на условните преходи (инструкции CMOVcc, FCMOVcc за условно прехвърляне на данни).
- разширение на методите за адресация.

### 2. Разширение на програмния модел

Програмно достъпните регистри в 32 разредните микропроцесори включват като подмножество 16 разредните регистри на базовия микропроцесор и се състоят от шест 16 разредни сегментни регистри (CS, DS, SS, ES, FS, GS), осем 32 разредни регистри за общо предназначение (EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP), 32 разреден регистър на флаговете (EFLAGS), 32 разреден програмен брояч (EIP). С цел съвместимост младшите думи на осемте регистъра с общо предназначение образуват 16 разредното подмножество регистри на I8086, имат същите имена (AX, BX, CX, DX, SI, DI, SP, BP) и могат да се третират като отделни единици. Освен това, аналогично на I8086, всеки байт на регистрите AX, BX, CX, DX има собствено име и може да се използва самостоятелно - AH, BH, CH, DH (старши байтове) и AL, BL, CL, DL (младши байтове). Флаговият регистър е разширен с нови флагове, свързани с особеностите на защитения и виртуалния режим на работа. С цел осигуряване на указаните режими в програмния модел са въведени и системни регистри: 4 управляващи регистъра (CR0-CR3), 8 регистъра за трасиране на програмата (DR0-DR7), регистри за управление достъпа до паметта в защитен режим (IDTR, GDTR, LDTR), регистър на задачите за осигуряване на мултизадачна работа(TR). Започвайки от I486, са въведени и регистри за тестване на вътрешната кеш памет и асоциативния буфер TLB (TR0-TR7). В множеството системни регистри на Pentium и P6 са включени и 64 разредни регистри със специално предназначение (MSR), които съдържат специфична, свързана с функционирането на процесора, информация.

### 3. Типове данни

Разширението на типовете данни, с които оперират инструкциите на МП, произтича от увеличаването на разредността на вътрешните регистри на процесора, интегриране на устройството за плаваща запетая в архитектурата на МП, изискванията на бурно развиващите се приложения в областта на мултимедиите, Internet и др. Основните типове данни в архитектурата Intel са байт, дума (16 бита), двойна дума (32 бита) и четворна дума (64 бита). Най-младшият байт на всеки тип данни се разполага на най-малкия адрес от последователността адреси, заемана от типа. Той е и адрес на операнд от дадения тип (т.н. little Endian формат). Операнди в инструкциите могат да бъдат:

- цели числа (Integers). За представяне на цели числа се използват знакови или беззнакови байт, дума или двойна дума;

• двоично-десетични числа (BCD). BCD операндите могат да бъдат пакетирани (един байт съдържа две десетични цифри в интервала от 0 до 9) или непакетирани (една десетична цифра се съдържа в един байт);

• указатели - адреси на обекти в паметта. Те бивам близки (near) и далечни (far). Близките указатели са 32 разредни и задават адрес в рамките на текущия сегмент, а далечните са 48 разредни и сочат обект в друг сегмент;

• битови полета - последователност от битове, започваща от произволна позиция на произволен байт в паметта и с максимална дължина 32 бита;

• низове - непрекъсната последователност от байтове в паметта;

• числа с плаваща запетая (real). Операндите с плаваща запетая могат да бъдат с единична (32 бита), двойна (64 бита) и разширена (80 бита) точност;

• MMX операнди.

**4. Разширение на функционалните възможности на МП**

**Инструкции за поддържане на многозадачни и многопотребителски операционни системи.**

За осигуряване на мултизадачен режим и други важни функции на операционните системи (ОС) са въведени следните множества инструкции:

• Инструкции за защита и изолиране на адресните пространства на отделните задачи в многозадачни ОС: зареждане на системните регистри GDTR, LDTR и IDTR с базовите адреси на съответните дескрипторни таблици - ***lgdt***, ***lldt*** и ***lidt***; инструкции за проверка на атрибутите на сегмент - получаване на правата на достъп (***lar***), лимита (***lsl***), достъпността за четене (***verr***), достъпността за запис (***verw***).

• Инструкции за превключване на задачи: зареждане на регистъра на задачата TR с указател към сегмента с нейния статус (***ltr***), съхраняване на съдържанието на TR в паметта (***str***); разширение на инструкциите за безусловен преход (***jmp***) и извикване на процедура (***call***) с възможност за превключване към задача.

• Инструкции за промяна реда на съхраняване на байтовете на многобайтовите операнди в регистрите и паметта за подпомагане работата в разпределена среда, изпълнение на разпределени и Internet приложения: реверсиране на байтовете в 32-разреден регистър, така че битове 0-7 се разменят с битове 24-31 и битове 8-15 с битове 16-23 (***bswap***);

• Неделими инструкции за реализация на семафори или други структури за синхронизация на процеси: размяна на съдържанието на два операнда (***xchg***), сравнение и размяна (***cmpxchg***); сравнение и размяна на 8 байта (***cmpxchg8b***); размяна и сумиране (***xadd***);

• Инструкции за запис и четене в/от системните регистри за смяна и проверка режимите на работа на процесора, четене на брояча на временни интервали (***rdtsc***), четене на MSR регистър (***rdmsr***), запис в MSR регистър (***wrmsr***), четене от броячите за отчитане на производителността (***rdpmc***).

**Инструкции за поддържане на езиците от високо ниво.**

За опростяване програмирането на процедури в блоково структурираните езици са въведени инструкциите ***enter*** и ***leave***. С инструкцията ***enter*** се създава стеков кадър за локалните променливи на процедура при нейното извикване, а с инструкцията ***leave*** паметта, заета за стеков кадър на текущата процедура се освобождава. Инструкцията ***bound*** се използва за проверка дали знаков индекс на масив се намира в рамките на заделената за масива област памет. Инструкцията генерира изключение, ако проверяваният индекс е извън границите на масива. За по-бързо съхранение на осемте регистъра на МП в стек се използват инструкциите ***pusha*** (за 16 разредното подмножество) и ***pushad*** (за 32 разрядното множество), а инструкциите ***popa*** и ***popad*** за извличане на съдържанието на тези регистри от стека.

**Поддържане на многопроцесорна конфигурация**

Поддържането на мултипроцесорни системи се изразява основно в гарантиране на неделимостта на инструкциите за синхронизация на процесите по отношение на цялостната система микропроцесори посредством автоматично или с префикс ***lock*** заключване на шината при тяхното изпълнение и поддържане консистентността на вътрешната кеш памет с помощта на инструкциите ***invd*** (изчистване на кеш паметта), ***wbinvd*** (обратен запис в паметта и изчистване на кеш паметта), ***invlpg*** (изчистване на запис в буфера TLB). При суперскаларните процесори Intel се въвеждат

инструкции със сериализиращо действие с цел правилно подреждане на обръщенията към паметта и входно-изходните устройства. Такова действие имат всички привилегировани инструкции и инструкциите *cpuid* (тестване типа на процесора), *iret* (връщане от прекъсване), *rsm* (изход от режима за управление на МП).

**Инструкции за оптимизация на програмния код**

За избягване на някои типове преходи, снижаващи производителността на конвейера на инструкциите, могат да се използват условни прехвърляния на данни - ***CMOVcc*** за целочислени операнди и ***FCMOVcc*** за операнди с плаваща запетая. Със "*cc*" е обозначено проверяваното условие от регистъра EFLAGS, при удовлетворяване на което се извършва прехвърляне на данни между паметта и регистрите с общо предназначение в случай на ***CMOVcc*** или между регистрите в FPU стека при ***FCMOVcc.***

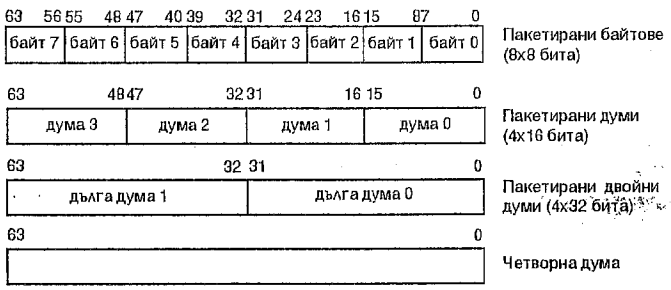
## 5. Инструкции MMX

Архитектурата на новите фамилии процесори Intel е разширена с възможност за ускорение на приложения, при които се изпълняват еднотипни операции над множество данни. За тази цел са въведени следните архитектурни компоненти:

- осем 64 разредни MMX регистъра MM0 - MM7;
- четири нови типа данни;
- набор MMX инструкции.

Регистрите MMX не са физически отделни регистри, а синоними на регистрите с плаваща запетая R0 - R7 и заемат 64 разредната им част за мантиса. Те могат да бъдат директно адресирани в MMX инструкциите за изпълнение на определени операции над съдържащите се в тях данни, но не могат да бъдат източници на адресни компоненти. Поради общите регистри превключването от MMX инструкция към инструкция с плаваща запетая може да заеме до 50 такта. Желателно е двата типа инструкции да не се смесват в кода на програмата.

Типовете данни MMX (фиг. 3.1) са: 8 пакетирани байта; 4 пакетирани думи; 2 пакетирани двойни думи и четворна дума. Пакетираните байтове, думи и двойни думи могат да се интерпретират като цели числа със или без знак.




Фиг.3.1. Типове данни MMX

Аритметичните и логически операции над MMX операндите се реализират като SIMD операции над байтовете, думите и дългите думи в 64 разредните MMX регистри. По този начин една и съща операция се изпълнява паралелно над множество данни.

Множеството от 57 MMX инструкции включва:

- инструкции за прехвърляне на данни;
- аритметични инструкции;
- инструкции за сравнение;
- инструкции за преобразуване;
- логически инструкции;
- инструкции за преместване;
- инструкция за изчистване на MMX състоянието (*emms*).

MMX инструкциите се характеризират с някои особености:

- въведена е нова аритметична техника, известна като аритметика на насищане. При препълване и изгубване на значимост резултатът се установява равен съответно на горната или

БИБЛИОТЕКА ТУ - ДАВА Инв. № Х-3204/999

долната граница на съответния диапазон на представимост на числата за използвания формат. Например при препълване на знаков байт резултатът се установява равен на 127, а при изгубване на значимост на -128 (таблица 3.1). Аритметичните инструкции могат да се изпълняват и с отрязване, при което положителното или отрицателно препълване се игнорират и резултатът се отрязва в рамките на използваните разреди;

• MMX инструкциите не предизвикват прекъсване или установяване на флагове при препълване или изгубване на значимост;

• инструкциите са двуоперандни от вида ***op dst, src***, където ***op*** задава изпълняваната операция над операндите ***dst*** (назначение) и ***src*** (източник). Резултатът от операцията се записва в ***dst***. Операндът ***dst*** може да се задава само чрез MMX регистър, докато ***src*** може да бъде както в MMX регистър, така и в паметта. Изключение правят инструкциите за прехвърляне, при които и двата операнда могат да бъдат в паметта или в целочислените регистри;

• върху MMX инструкциите въздействат само префиксите за размер на адреса (67h) и смяна на сегмента, ако оперират с операнд от паметта. Останалите префикси се игнорират, а префиксът ***lock*** предизвиква изключение.

**Таблица 3.1**

| Типове данни | Долна граница | | Горна граница | |
|---|---|---|---|---|
| | | | | |
| знаков байт | 80 H | -128 | 7F H | 127 |
| знакова дума | 8000 H | -32768 | 7FFF H | 32767 |
| беззнаков байт | 00 H | 0 | FF H | 256 |
| беззнакова дума | 0000 H | 0 | FFFF H | 65585 |

В таблица 3.2 са класифицирани инструкциите MMX по функционално назначение и режим на изпълнение (отрязване или насищане).

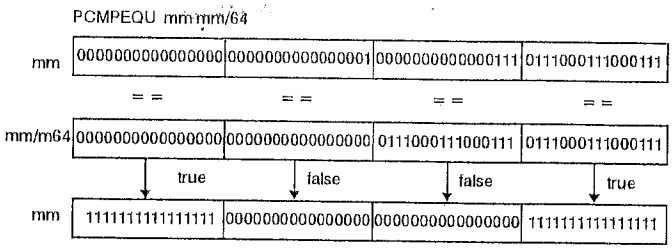
**Инструкции за прехвърляне на данни.** С инструкциите за прехвърляне се преместват 32 разредни (***movd***) или 64 разредни (***movq***) данни между паметта и MMX регистър или между MMX регистър и целочислен регистър.

**Инструкции за събиране и изваждане.** Инструкциите ***paddb***, ***paddw*** и ***paddwd*** (пакетирано събиране) и ***psubb***, **psubw** и ***psubd*** (пакетирано изваждане) прибавят или съответно изваждат пакетираните знакови или беззнакови елементи данни на операнда *src* към или от съответните елементи на операнда *dst* в режим на отрязване. Инструкциите ***paddsb*** и ***paddsw*** (пакетирано събиране с насищане) и ***psubsb*** и ***psubsw*** (пакетирано изваждане с насищане) прибавят или изваждат знаковите елементи на *src* към или от елементите на *dst* и насищат резултата до границите на диапазона на съответния знаков тип (*b* - байт, *w* - дума). Действието на **paddusb** и ***paddusw*** (беззнаково събиране с насищане) и ***psubusb*** и ***psubusw*** (беззнаково изваждане с насищане) е аналогично.

**Инструкции за умножение.** Инструкциите за пакетирано умножение изпълняват 4 умножения над двойките знакови 16 разредни елементи на операндите *src* и *dst*, изработвайки 32 разредни междинни произведения. 16 разредните резултати в *dst* могат да се формират или от младшата (***pmullw***) или от старшата (***pmulhw***) част на тези произведения.

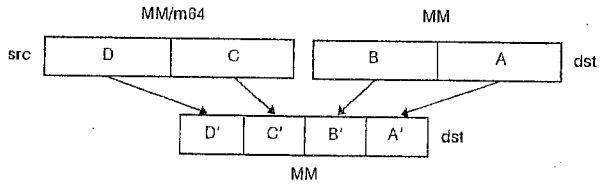
При инструкцията ***pmaddwd*** (пакетирано умножение и събиране) междинните 32 разредни произведения се събират по двойки за получаване на резултат от две 32 разрядни двойни думи.

**Инструкции за сравнение.** Инструкциите за сравнение сравняват съответстващите пакетирани елементи данни в *src* и *dst* за равенство (***pcmpeqb***, ***pcmpeqw***, ***pcmpeqd***) или за по-голяма стойност (***pcmpgtb***, ***pcmpgtw***, ***pcmpgtd***). Тези инструкции генерират маски от нули и единици, които се записват в операнда *dst* (фиг. 3.2). Маските могат да бъдат използвани за операциите условно пакетно прехвърляне за избягване на преходите или за реализация на множество инструкции за преход. Инструкциите за сравнение не установяват флагове в регистъра EFLAGS.




Фиг. 3.2. Действие на инструкциите за сравнение.

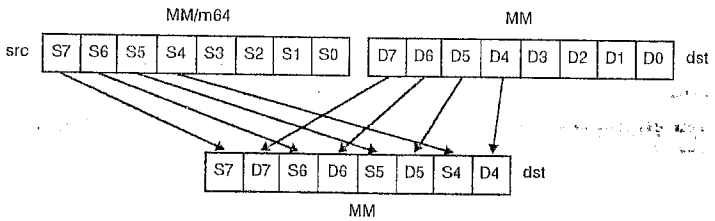
**Инструкции за преобразуване.** Инструкциите за пакетиране преобразуват знакови думи в байтове (*packsswb*) или знакови двойни думи в думи (*packssdw*) с насищане. Например инструкцията *packssdw* пакетира двете знакови двойни думи на операнда *src* и двете знакови двойни думи на *dst* в 4 думи в операнда *dst* (фиг. 3.3.). Ако знаковото значение на двойната дума е извън границата на диапазона за знакова дума (по-голямо от 7FFFh или по-малко от 8000h), то в *dst* се записва съответната наситена стойност 7FFFh или 8000h.




Фиг.3.3. Действие на инструкцията **packssdw** *MM, MM/m64*

Инструкцията *packuswb* (пакетиране с беззнаково насищане) преобразува знакови думи в беззнакови байтове с беззнаково насищане.

Инструкциите *punpckhbw*, *punpckhwd* и *punpckldq* (разпакетиране на старшите пакетирани данни) *punpcklbw*, *punpcklwd* и *punpckldq* (разпакетиране на младшите пакетирани данни) преобразуват байтове в думи, думи в двойни думи или двойни думи в четворни думи (фиг. 3.4).

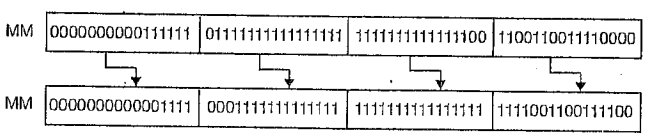



Фиг. 3.4. Действие на инструкцията **punpckhbw** *MM, MM/m64*

**Логически инструкции.** Логическите операции изпълняват съответните побитови логически операции над 64 разредни операнди.

**Инструкции за преместване.** Инструкциите за пакетирано преместване *psllw*, *pslld* и *psllq* (логическо наляво) и *psrlw*, *psrld* и *pslld* (логическо надясно) изпълняват логическо преместване наляво или надясно над всеки пакетиран елемент (дума, двойна дума или четворна дума) на зададения с src брой разреди със запълване на освободените разреди с нули.

Инструкциите за пакетирано аритметично преместване надясно *psraw* и *psrad* се изпълняват над всяка от четирите пакетирани думи или над всяка от двете двойни думи на операнда *dst*, като копират знаковия разред в освобождаващите се разреди (фиг. 3.5).




Фиг. 3.5. Действие на инструкцията psraw *MM, 2*

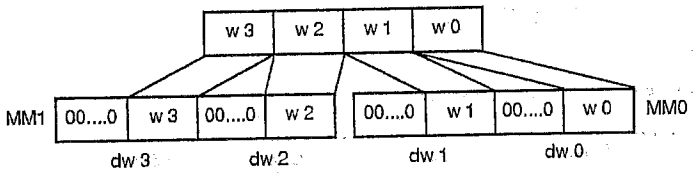
## 6. Примери за използване на инструкциите MMX

### 6.1. Беззнаково разпакетиране

Примерът демонстрира разпакетиране на операнд от 4 безнакови думи в два операнда от по две двойни думи с нулево разширение (фиг. 3.6).

Входни данни:
MM0 - начален операнд
MM7 - 0
Изходен резултат:
MM0 - разпакетирани младши думи
MM1 - разпакетирани старши думи

```
movq        MM1, MM0   ; създаване на копие на операнд MM0
punpcklwd   MM0, MM7   ; разпакетиране на двете младши думи на
                         изходния операнд в 32 битови двойни думи
punpckhwd   MM1, MM7   ; разпакетиране на двете старши думи на
                         изходния операнд в 32 битови двойни думи
```




Фиг. 3.6. Беззнаково разпакетиране

### 6.2. Знаково разпакетиране

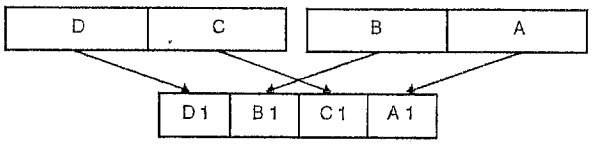
Примерът демонстрира разпакетиране на операнд от 4 знакови думи в два операнда от по две двойни думи със знаково разширение.
Входни данни:
MM0 - начален операнд
Изходен резултат:
MM0 - разпакетирани младши думи със знаково разширение
MM1 - разпакетирани старши думи със знаково разширение

```
punpckhwd   MM1, MM0   ; разпакетиране на двете старши думи на изходния
                         операнд във втора и четвърта дума на MM1
punpcklwd   MM0, MM0   ; разпакетиране на двете младши думи на изходния
                         операнд във втора и четвърта дума на MM0
psrad       MM0, 16    ; знаково разширение на двете двойни думи в MM0
psrad       MM1, 16    ; знаково разширение на двете двойни думи в MM1
```

### 6.3. Пакетиране

Инструкциите за пакетиране събират двете стойности, зададени с *dst* и *src* в *dst* в регламентиран ред. Следващият пример демонстрира пакетиране на знакови двойни думи с редуване и насищане.
Входни данни:
MM0 - начален знаков операнд
MM1 - начален знаков операнд
Изходен резултат:
MM0 - първата и третата думи съдържат знаково наситени двойни думи от MM0
MM0 - втората и четвъртата думи съдържат знаково наситените двойни думи от MM1 (фиг.3.7).

```
packssdw    MM0, MM0   ; пакетиране и знаково насищане
packssdw    MM1, MM1   ; пакетиране и знаково насищане
punpklwd    MM0, MM1   ; редуване на младшите 16 разреда на операндите
```




Фиг. 3.7. Пакетиране с редуване

### 6.4. Умножение на комплексно число с константа

Умножението на комплексни числа изисква 4 умножения и две събирания и се извършва с инструкцията ***pmoddwd***. Реалната Dr и имагинерната Di част на комплексното число са 16 разредни. Форматът на комплексната константа се представя чрез четири 16 разредни стойности (пакетирани думи) - [Cr -Ci Ci -Cr].

Входни данни:

MM0 - комплексно число Dr, Di

MM1 - комплексна константа формат [Cr -Ci Ci -Cr]

Изходен резултат:

MM0 - две 32 разредни двойни думи, съдържащи [Pr Pi], където Pr е реалната част на резултата, а Pi - имагинерната част.

```
Pr = Dr * Cr - Di * Ci
Pi = Dr * Ci - Di * Cr
```

За получаването на резултата са необходими две MMX инструкции:

```
punpckldq  MM0, MM0   ; получаване на [Dr Di Dr Di]
pmaddwd    MM0, MM1   ; получаване на резултата в вид
                      ; [(Dr*Cr-Di*Cr)(Dr*Ci+Di*Cr)]
```

### 6.5. Абсолютна разлика на беззнакови числа

Входни данни:

MM0 - операнд *a* от тип пакетирани беззнакови байтове

MM1 - операнд *b* от тип пакетирани беззнакови байтове

Изходен резултат:

MM0 - абсолютна разлика на беззнаковите операнди

```
movq       MM2, MM0   ; копие на операнд a
psubusb    MM0, MM1   ; разлика a-b
psubusb    MM1, MM2   ; разлика b-a
por        MM0, MM1   ; избор на по-голямата разлика
```

Примерът не е коректен, ако операндите са със знак.

### 6.6. Изчисляване на абсолютна стойност

Примерът демонстрира изчисляване на абсолютна стойност на пакетирани знакови думи - |x|.

Входни данни:

MM0 - операнд от пакетирани знакови думи

Изходен резултат:

MM1 - абсолютна стойност на пакетираните знакови думи

```
movq       MM1, MM0   ; копие на операнда
psraw      MM0, 15    ; размножаване на знаковия бит на пакетираните
                      ; думи във всичките им разреди
pxor       MM0, MM1   ; обратен код на отрицателните думи
psubs      MM1, MM0   ; получаване на абсолютна стойност
```

**Въпроси и задачи:**

1. Съставете алгоритъм за изчисляване на абсолютната разлика между 16-разредните елементи на два едномерни масива с размер 10. Програмирайте алгоритъма по технология MMX и със стандартните инструкции на МП. Сравнете и оценете двете програми.

2. Напишете програма за разпознаване на работния микропроцесор.

За разпознаване на ранните МП могат да се използват следните архитектурни особености: 8086 - битове 12 до 15 на регистъра EFLAGS винаги са установени в "1"; I286 - битове 12 до 15 на EFLAGS са нули; I386 - бит 18 (AC - контрол за изравняване) на EFLAGS не може да бъде променян. Инструкцията *cpuid* е въведена от процесорите Pentium и бит 21 (ID) на EFLAGS определя дали инструкцията е реализирана в дадения МП. Възможността да се променя този бит индицира, че МП е Pentium или от фамилията P6. Чрез проверка възможността да се променят указаните битове на EFLAGS (с помощта на инструкциите *pushf/pushfd* и *pop/popfd*) може да се определи от кой тип е МП.

Инструкцията *cpuid* връща информация за МП в регистрите EAX, EBX, ECX и EDX. Тя може да се изпълни в три режима , връщащи различна информация за МП. Режимите се различават по значението на регистъра EAX преди стартиране на *cpuid*:

- EAX = 0 - в EAX се установява номер съответстващ на типа на МП (например 1 - Pentium, 2 - PentiumPro и т.н.), EBX съдържа низа "Genu", ECX- "ntel", а EDX - "ineI". Конкатенацията на съдържанието на EBX, EDX и ECX задава наименованието на производителя ("GenuineIntel" за процесорите на Intel);
- EAX = 1 - в EAX се установява информация за типа, фамилията и модела на МП, в EDX се указва наборът фукционални възможности, EBX и ECX не се използват;
- EAX = 2 - в EAX, EBX, ECX EDX се записва информация за вътрешната кеш памет и буфера TLB.

При режим 2 установяване в единица на отделни битове на регистър EDX задава: EDX[0] - наличие на FPU в чипа; EDX[1] - разширен виртуален режим; EDX[4] - наличие в чипа на TSC брояч на временни интервали; EDX[6] - разширен физически адрес; EDX[8] - реализирана инструкция cmpxchg8b; EDX[9] -APIC в кристала; EDX[15] - реализирани инструкции CMOVcc; EDX[23] - MMX инструкции и др.

**Таблица 3.2**

| Тип | Отрязване | Знаково насищане | Беззнаково насищане |
|---|---|---|---|
| Аритметическо Събиране | PADDB, PADDW, PADDD | PADDSB, PADDSW | PADDUSB, PADDUSW |
| Изваждане | PSUBB, PSUBW, PSUBD | PSUBSB, PSUBSW | PSUBUSB, PSUBUSW |
| Умножение | PMULL, PMULLH | | |
| Умножение и събиране | PMADD | | |
| Сравнение Сравнение за равенство | PCMPEQB, PCMPEQW, PCMPEQD | | |
| Сравнение за по-голямо | PCMPGTPB, PCMPGTPW, PCMPGTPD | | |
| Преобразуване Пакетиране | | PACKSSWB, PACKSSDW | PACKUSWB |
| Разпакетиране старши | PUNPCKHBW, PUNPCKHWD, PUNPCKHDQ | | |
| Разпакетиране младши | PUNPCKLBW, PUNPCKLWD, PUNPCKLDQ | | |
| | Пакетирани | | Четворна дума |
| Логически And | | | PAND |
| And Not | | | PANDN |
| Or | | | POR |
| Exclusive Or | | | PXOR |
| Преместване Логическо наляво | PSLLW, PSLLD | | PSLLQ |
| Логическо надясно | PSRW, PSRLD | | PSRLQ |
| Аритметическо Надясно | PSRAW, PSRAD | | |
| Нулиране на състоянието MMX | EMMS | | |

# 4. Оптимизация на код

Оптимизацията на програмите цели да се ускори тяхното изпълнение. Познаването на детайлите от работата на конвейера на инструкциите и особеностите на архитектурата на процесорите позволява да се изгради стратегия за оптимизация на програмния код. Поради приемствеността в архитектурата могат да се определят общи оптимизационни механизми за цялата фамилия процесори Intel. От друга страна обаче, разликите в организацията на конвейера предполагат и специфични техники за оптимизация за всеки тип микропроцесор. Болшинството от подходите за оптимизация са базирани на избор и разместване на инструкциите на програмата за осигуряване ритмичността на конвейера.

### 1. Влияние на методите за адресация върху оптимизацията на програмата

Последователността на използване на някои методи на адресация влияе на ритмичността на конвейера. Например използване на регистър за база или индекс при изчисляване на адрес изисква допълнителен такт, ако в този регистър е извършван запис в предшестващата инструкция. Ефектът е известен като блокиране на генерацията на адреса или AGI (Address Generation Interlock).

**Пример:**

```
add     esi, eax        ; esi - регистър за резултат
mov     eax, [esi]      ; esi - базов регистър.
                        ;1 допълнителен такт
```

Ефектът се появява и пред инструкции, генериращи адрес неявно чрез регистъра ESP (***push***, ***pop***, ***ret***, ***call***):

```
sub     esp, 24
                        ; 1  такт задръжка - push генерира адрес чрез esp
push    ebx
mov     esp, ebp        ; след push и pop няма AGI
                        ; 1 такт задръжка
pop     ebp
```

Pentium е неутрален по отношение на индексната адресация.

В I486 допълнителен такт е необходим при непосредствен операнд, а в Pentium комбинация с непосредствен операнд не може да образува двойка в конвейерите U и V.

### 2. Регистрова зависимост

В I486 се получава 1 такт задръжка, а в процесорите от фамилията P6 много по-голяма, когато се използва регистър, непосредствено след като е извършен запис в негов подрегистър. В Pentium този тип задръжка не се наблюдава.

**Пример:**

**i486**

```
mov   al, 0       ;
                  ; 1  такт задръжка
mov   [esp], eax ;
```

**Pentium**

```
mov   al, 0
mov   [esp], eax ; няма задръжка
```

Задръжката при микропроцесорите от фамилия P6 при четене от 32 битов регистър след запис в негов 16 или 8 битов подрегистър продължава, докато микроинструкцията за запис бъде отстранена от пула по реда си на завършване. Това може да заеме минимум 7 такта. Тъй като изпълнението не е по реда в програмата, то ефектът се появява и когато инструкцията за четене не е непосредствено след инструкцията за запис.

```
mov   al, 8
mov   edx, 0x40
mov   edi, new_value
add   edx, eax          ;задръжка при достъп до регистър EAX
```

Естествено задръжката ще се разпростре и към следващите mops, докато задържаната mop не възстанови движението си в конвейера. За да се отстрани този ефект са разработени специални

случаи на запис и четене в двойките подрегистър - регистър с използване на инструкциите *xor* и *sub*. Същността е нулиране на по-големия регистър, преди запис в подрегистър.

**Примери:**

```
xor     eax, eax
movb    al, mem8      ;запис в регистъра AL на 8-разреден операнд
use     eax           ;използването на EAX след запис в AL не предизвиква задръжка

sub     ax, ax
movb    ax, mem16
use     ax            ;използването на ax след запис в al не предизвиква задръжка
```

## 3. Изравняване

### 3.1. Изравняване на код

Извличането на инструкциите от вътрешната L1 кеш на инструкциите се извършва по редове от 32 байта с изравняване по адресната граница на 16 байта. Следователно изравняването на кода има директно влияние върху производителността при извличане.

### 3.2. Изравняване на данните

Неизравнен достъп до данновата L1 кеш или до външната памет може да коства повече от 3 допълнителни такта за Pentium и между 9 и 12 такта при P6. Важат следните правила за изравняване на данните:

- двубайтов операнд изцяло трябва да се съдържа в изравнена 4 байтова дума (двоичните адреси могат да бъдат xxxx00, xxxx01, xxxx10, но не и xxxx11);
- 4 байтови обекти трябва да бъдат изравнени от четири байтова адресна граница;
- 8 байтови данни (например числа с плаваща запетая, MMX операнди и др.) трябва да бъдат изравнени по 8 байтова адресна граница.

## 4. Влияние на префиксите

В I486 и Pentium използването на префикси пред инструкциите може да предизвика задържане и да препятства сдвояването. Изключение прави само префикс 0F (втори байт за код на операция), който не въвежда допълнителен такт. Префиксите за промяна на размера на адреса (67h) и операнда (66h) изискват 3 такта - 1 такт за откриване на префикса, 1 такт за определяне на дължината и 1 такт за помещаване в опашката на инструкциите. Другите видове префикси изискват само такт за разпознаване (декодиране).

При фамилията P6 префикс може да увеличават дължината на модифицираните от тях инструкции над 7 байта, тъй като добавят 1 или 2 байта към тях и така да намалят броя на декодираните в един цикъл инструкции. С цел оптимизация се препоръчва да не се използват префикси или по възможност те да се разполагат след инструкции, които задържат конвейера по други причини (AGI например).

## 5. Оптимизация на преходите

Оптимизацията на преходите е най-важният фактор за ефективността на динамичното изпълнение при микропроцесорите от P6. Тя дава добри резултати и при Pentium.

### 5.1 Динамично предсказване

Три са важните елементи в динамичното предсказване на преходите:

- ако адресът на инструкцията за преход не е в BTB (среща се за първи път при изпълнение), преходът се предсказва по статичен алгоритъм;
- предсказван вече преход заема 1 такт;
- в BTB се съхранява 4 битова история на предсказване за всеки срещан преход. На базата на тази история BTB предсказва коректно регулярни преходи. Например коректно се предсказва преходът при цикъл, който се осъществява при всяка четна итерация и не се осъществява при всяка нечетна итерация на цикъла.

### 5.2. Статично предсказване

Преходи, за които няма история в BTB, т.е. срещат се за първи път, се предсказват на базата на статичен алгоритъм, при който:

- безусловните преходи се предсказват като осъществени;
- условните преходи назад се предсказват като осъществени. Това е удобно за предсказване на преходите на цикли;
- условните преходи напред се предсказват като неосъществени.

Задръжката при статично предсказване е 6 такта, а задръжката поради грешно предсказване е по-голяма от 12 такта.

**Примери:**

```
begin:  mov   eax, mem32
        and   eax, ebx
        imul  eax, edx
        shld  eax, 7
        jc    begin
```

В примера преходът назад (**jc begin**) не се намира в BTB, когато се преминава през него за пръв път и следователно BTB не генерира динамично предсказване. Статичният алгоритъм обаче предсказва осъществяване на прехода.

```
        mov   eax, mem32
        and   eax, ebx
        imul  eax, edx
        shld  eax, 7
        jc    begin
        mov   eax, 0
begin:  call  convert
```

Преходът напред (**jc begin**) се среща за първи път и за него няма запис в BTB, но статичният алгоритъм предсказва неосъществяване на прехода, т.е. след него ще се извлече инструкцията **mov eax, 0**. Инструкцията **call convert** също не е в BTB, но се предсказва на базата на статичния алгоритъм като осъществен безусловен преход.

### 5.3. Правила за оптимизиране на преходите

С цел оптимизация на програмата се препоръчва:

- намаляване и даже елиминиране на преходите;
- за всяка инструкция *call* трябва да съществува съответната *ret*, тъй като в процесорната стеков буфер, съхраняващ адреса на връщане;
- избягване на прекалено кратки цикли.
- не се препоръчва разполагане на данни между инструкциите в кодовия сегмент;

Елиминирането на преходите увеличава производителността на процесора, отстранявайки вероятността за грешно предсказване и намаляване на заетите елементи в BTB. Преходите могат да се елиминират чрез използване на инструкцията *SETcc* или в P6 на инструкциите за условно прехвърляне (**CMOVcc** и *FCMOVcc*).

**Пример:**

Фрагментът от програма на език C

```
ebx = (A < B) ? C1 : C2;
```

сравнява значенията на A и B. Ако сравнението е true, то в регистър EBX се установява значение C1, в противен случай - значение C2. Асемблерен еквивалент на този код може да бъде:

```
        cmp   A, B        ; определяне на условието
        jge   L30         ; условен преход
        mov   ebx, C1     ;
        jmp   L31         ; безусловен преход
L30:
        mov   ebx, C2
L31:
```

Асемблерният код може да бъде оптимизиран като се елиминират преходите с използване на инструкцията ***SETcc***, която записва байт със значение 1 в своя операнд, ако условието **cc** е изпълнено или байт със значение 0, ако условието не е изпълнено.

```
xor   ebx, ebx              ; нулиране на регистъра EBX
cmp   A, B
setge bl                    ; установява 1 или 0 в BL в зависимост от условието
dec   ebx                   ; EBX=00...00 или 11...11
and   ebx, (C2 - C1)        ; EBX=0 или (C1-C2)
add   ebx, min (C1,C2)      ; EBX=C1 или C2
```

Преходите могат да бъдат елиминирани и посредством инструкция за условно преместване ***CMOVcc***, която проверява значението на зададения чрез *cc* флаг (или флагове) от регистър EFLAGS и изпълнява операцията ***mov***, ако той е в зададеното състояние.

**Пример:**

Асемблерен фрагмент с условен преход, зависим от значението на данните, който е непредсказуем и следователно силно намалява производителността на изпълнение:

```
        test ecx, ecx
        jne  lh
        mov  eax, ebx
lh:
```

Непредсказуемият преход може да бъде отстранен от асемблерния текст по следния начин:

```
test      ecx, ecx   ; установяване на флаговете
cmoveq    eax, ebx   ; ако флагът ZF (равенство) е вдигнат, то се
                         изпълнява mov eax, ebx
```

## 6. Планиране на инструкциите

Суперскаларността се базира на експлоатация на възможния локален паралелизъм в програмата, т.е. възможността за паралелно изпълнение на текущата инструкция със следващите инструкции. Аранжирането (пренареждането) на инструкциите в потока на програмата може да има значителен ефект върху производителността на процесора. Вградените в Pentium два паралелни конвейера за инструкции (U и V) са в състояние да изпълняват паралелно две инструкции, ако те са сдвоими. Сдвоими са тези инструкции, които разположени в съседство в програмата могат безпроблемно да се изпълняват едновременно. Сдвояването и аранжирането на инструкциите оптимизира производителността и на процесорите P6.

### 6.1. Основни правила за сдвояване в Pentium

Организацията на конвейерите в Pentium налага определени правила за разпределяне на инструкциите за изпълнение в тях. Основните правила включват:

- несдвоимите инструкции винаги се изпълняват в конвейер U;
- за да се изпълнят паралелно две сдвоими инструкции, първата от тях трябва да бъде разположена на такава граница в програмния поток, от която инструкциите се запускат за изпълнение в конвейер U:
  - ако първата от две сдвоими инструкции е първа в програмен блок, то тя ще се изпълни в конвейер U, а втората паралелно в конвейер V;
  - ако първата от две сдвоими инструкции следва след несдвоима инструкция в програмния поток, то тя ще се изпълни в конвейер U, а втората в конвейер V;
  - двойка инструкции, следваща след паралелно изпълнени инструкции, също ще се изпълни в паралел и така до срещане на несдвоима инструкция.
- две съседни инструкции няма да се изпълнят паралелно, ако:
  - не са сдвоими;
  - използват явно или неявно един и същ регистър (регистрова зависимост);
  - не се намират и двете в кеш паметта;
  - процесорът работи в стъпков режим;
  - имат даннова зависимост.

• не могат да се сдвояват две инструкции, ако едната от тях е по-дълга от 7 байта;

• инструкциите с префикси са сдвоими само в U конвейера. При Pentium MMX инструкциите с префикси 0Fh, 66h и 67h са сдвоими и в V конвейера;

• при едновременната работа на двата конвейера трябва да се съхранява редът на обръщение към паметта. Така многотактна команда в U ще се изпълнява сама до последното обръщение към паметта;

• инструкциите с плаваща запетая не са сдвоими с MMX инструкциите.

### 6.2. Сдвоими инструкции

Не всички двойки инструкции са сдвоими. За инструкциите от всеки тип се задава възможността за сдвоимост, а именно:

• основната част целочислени аритметико-логически инструкции (***add***, ***sub***, ***and***, ***xor***, **inc** и др.) са сдвоими;

• всички инструкции за сравнение (***cmp***, ***test***) са сдвоими;

• всички стекови операции, използващи регистрови операнди, са сдвоими (***push reg***, ***pop reg***);

• не могат да се сдвояват инструкциите:

  •за преместване и ротация, когато броят на разредите се задава чрез регистър CL;

  •дългите аритметични операции, като ***mul***, ***div*** и др.;

  •междусегментни инструкции, например ***call far***, ***push sreg***;

  •някои инструкции с плаваща запетая.

Освен това редица инструкции могат да се изпълняват само в един конвейер и изпълнявайки се в него, могат да се сдвоят с подходящи инструкции в другия конвейер:

• към инструкциите, които могат да се изпълняват само в U конвейера (PU инструкции), спадат: аритметико-логически инструкции с отчитане на пренос и заем (***adc***, ***sbb*** и др.); префиксни инструкции; преместване с непосредствен операнд;

• инструкции, които, ако се изпълняват в V конвейера, могат да се сдвоят с подходящи инструкции в конвейер U (PV инструкции). Например простите инструкции за предаване на управлението ***call near***, ***jmp near*** и ***jcc*** могат де се изпълняват и в двата конвейера, но са сдвоими с други инструкции само, когато са в конвейер V.

Таблица 4.1 илюстрира сдвоимостта на инструкциите в Pentium.

**Таблица 4.1**

| Сдвоими целочислени инструкции в U - конвейера | | | Сдвоими целочислени инструкции в V - конвейера | | |
|---|---|---|---|---|---|
| mov reg, reg | ALU reg, imm | push reg | mov reg, reg | ALU reg, imm | push reg |
| mov reg, mem | ALU mem, imm | push imm | mov reg, mem | ALU mem, imm | push imm |
| mov mem, reg | ALU eax, imm | pop reg | mov mem, reg | ALU eax, imm | pop reg |
| mov reg, imm | ALU mem, reg | nop | mov reg, imm | ALU mem, reg | jmp near |
| mov mem, imm | ALU reg, mem | shift/rot by 1 | mov mem, imm | ALU reg, mem | jcc near |
| mov eax, mem | inc/dec reg | shift by imm | mov eax, mem | inc/dec reg | or jcc |
| mov mem, eax | inc/dec mem | test reg, r/m | mov mem, eax | inc/dec mem | call near |
| ALU reg, reg | lea reg, mem | test acc, imm | ALU reg, reg | ALU reg, mem | nop |
| | | | | test reg, r/m | test acc, imm |

**Примери:**

Сдвоими инструкции, които не могат да се изпълнят в паралел поради регистрова зависимост в случай, че първата от тях записва в общия регистър:

```
mov  eax, 8
mov  [ebp], eax
```

Регистрова зависимост не се наблюдава, ако първата инструкция чете от общия регистър. Следните две инструкции са сдвоими:

```
mov  eax, ebx
mov  ebx, [ebp]
```

Регистрова зависимост не се наблюдава и когато двойката инструкции записва в регистър EFLAGS (например две аритметико-логически инструкции, които променят кодовете на условие).

Регистрованата зависимост е отстранена и при т.н. специални двойки целочислени инструкции, които неявно използват регистър ESP или неявно записват в кодовете на условие:

• Сдвояване при неявно четене/запис в ESP:

```
push reg/imm   ;    push reg,imm
push reg/imm   ;    call
pop  reg       ;    pop reg
```

• Сдвояване при запис в кодовете на условие:

```
cmp            ;    jcc
add            ;    jne
```

Програмният фрагмент на C, реализиращ увеличаване с 1 елементите на два десетелементни масива **a** и **b**, има следния вид:

```
static int a[10], b[10];
int  i;
for ( i = 0; i < 10; i++ )
     {
          a[i] = a[i] + 1;
          b[i] = b[i] + 1;
     }
```

Възможни са различни правилни асемблерни кодове за този текст, но с различни експлоатационни характеристики. Един елемент на масивите (число от тип integer) заема 4 байта (dword).

**Вариант 1:**

```
     xor  eax, eax              ; нулиране брояча на цикъла EAX
loop:
     mov  edx, eax
     shl  edx, 2                ; в EDX индекса на поредния елемент на a
     inc  dword ptr  [edx+a]    ; инкрементиране на поредния елемент на a
     mov  edx, eax
     shl  edx, 2                ; в EDX индекса на поредния елемент на b
     inc  dword ptr  [edx+b]    ; инкрементиране на поредния елемент на b
     inc  eax                   ; инкрементиране на брояча на цикли
     cmp  eax, 10               ; изпълнени 10 итерации ?
     jl   loop                  ; не - преход към следваща итерация
```

В Pentium горният код ще се разпредели по следния начин в двата конвейера:

| *U конвейер* | *V конвейер* | *такт* |
|---|---|---|
| mov edx, eax | | ;1 |
| shl edx, 2 | | ;2 |
| inc dword ptr [edx+a] | | ;3 |
| (inc) | | ;4 |
| (inc) | | ;5 |
| (inc) | mov edx, eax | ;6 двойка |
| shl edx, 2 | | ;7 |
| inc dword ptr [edx+a] | | ;8 |
| (inc) | | ;9 |
| (inc) | | ;10 |
| (inc) | inc eax | ;11 двойка |
| cmp eax, 10 | jl loop | ;12 |
| mov edx, eax | | ;13 следваща итерация |

Първите две инструкции (*mov* и *shl*) не могат да бъдат сдвоени поради регистрова зависимост по отношение на EDX. Когато многотактна инструкция в U конвейера се сдвоява с друга инструкция, то последното обръщение към паметта в U конвейера образува двойка с първата операция в V конвейера (тактове 6 и 11). При даденото решение една итерация на цикъла се изпълнява за 12 такта.

**Вариант 2:**

```
    xor   eax, eax                ; EAX брояч на итерации и индекс в масивите
loop:
    inc   dword ptr  [eax*4+a]    ; инкрементиране на поредния елемент на a
    inc   dword ptr  [eax*4+b]    ; инкрементиране на поредния елемент на b
    inc   eax                     ; инкрементиране на брояча и индекса
    cmp   eax, 10                 ; изпълнени 10 итерации ?
    jl    loop                    ; не - преход към следваща итерация
```

В Pentium този код ще се изпълни по следния начин в двата конвейера:

| *U конвейер* | *V конвейер* | *такт* |
|---|---|---|
| `inc dword ptr [eax*4+a]` | | ;1 |
| `(inc)` | | ;2 |
| `(inc)` | `inc dword ptr [eax*4+b]` | ;3 |
| | `(inc)` | ;4 |
| | `(inc)` | ;5 |
| `inc eax` | | ;6 |
| `cmp eax, 10` | `jl loop` | ;7 |
| `inc dword ptr [eax*4+a]` | | ;8 следваща итерация |

Инструкция **inc eax** в такт 6 не образува двойка със **cmp** поради регистрова зависимост. Една итерация на цикъла в този случай се изпълнява за 7 такта.

**Вариант 3:**

Във вариант 3 е използван модел четене/запис (load/store).

```
    mov   eax, -40                ; в EAX отрицателния брой байтове, заемани от масив b
loop:
    mov   edx, [eax+40+a]         ; зареждане в EDX поредния елемент на a
    mov   ecx, [eax+40+b]         ; зареждане в ECX поредния елемент на b
    inc   edx                     ; инкрементиране на поредния елемент на a
    inc   ecx                     ; инкрементиране на поредния елемент на b
    mov   [eax+40+a], edx         ; съхранение в масив a
    mov   [eax+40+a], ecx         ; съхранение в масив b
    add   eax, 4                  ; добавяне в EAX дължината на един елемент
    jnz   loop                    ; край на обработката при EAX = 0
```

Подлежащия на увеличаване операнд се зарежда във вътрешен регистър (EDX и ECX съответно) и след извършване на действието отново се връща в паметта. Освен това броячът на циклите (регистър EAX) брои до нула, като по този начин се избягва операцията сравнение.

В Pentium този код ще се изпълни по следния начин в двата конвейера:

| *U конвейер* | *V конвейер* | *такт* |
|---|---|---|
| `mov edx, [eax+40+a]` | `mov ecx, [eax+40+b]` | ;1 |
| `inc edx` | `inc ecx` | ;2 |
| `mov [eax+40+a], edx` | `mov [eax+40+b], ecx` | ;3 |
| `add eax, 4` | `jnz loop` | ;4 |
| `mov edx, [eax+40+a]` | `mov ecx, [eax+40+a]` | ;5 |
| `(mov)` | `(mov)` | ;6 |

В приведената реализация се появява AGI от инструкцията **add eax,4** в такт 4, която записва в регистър EAX, а инструкциите в такт 5 и в двата конвейера използват EAX за изчисляване на адрес. Следователно се получава един такт задръжка преди започване на следващата итерация на цикъла. Една итерация на цикъла заема 5 такта. Използването на модела четене/запис дава по-големи възможности за сдвояване и планиране. Методът обаче изисква голям брой регистри, които за Pentium са ограничен ресурс, но са напълно достъпни в P6.

**Въпроси и задачи:**

1. Кои са основните фактори, които влияят на ритмичността на конвейера в I486, Pentium и P6?
2. Как влияе изравняването на инструкциите и операндите върху производителността на процесора?
3. Съставете програма за изчисляване на абсолютната разлика на елементите на два масива. Дискутирайте оптималността на вашето решение за I486 и Pentium. Възможно ли е сдвояване на инструкциите в двата конвейера на Pentium при вашето решение?

# 5. Системна архитектура. Управление на паметта в защитен режим

Целта на упражнението е изясняване на особеностите на основния режим на работа на 32-разредните микропроцесори Intel по отношение на управлението и защитата на паметта. Разглежда се механизмът на превключване на МП от реален в защитен режим и обратно.

Системната архитектура на процесорите на Intel се състои от набор регистри, структури от данни и инструкции за поддържане на операционните системи при техните функции за: управление на паметта, защита на програмните модули, мултизадачност, управление на прекъсвания и изключения, мултипроцесорна обработка, управление на кеш паметта, управление на апаратните ресурси и консумацията на мощност, тестване и наблюдение на производителността.

## 1. Режими на работа на МП

За архитектурата Intel са характерни три режима на работа и един подрежим:

• защитен режим (P-режим). Това е нормалният режим на процесора. В него са достъпни всички инструкции и архитектурни възможности на МП, като осигуряване на санкциониран достъп до паметта чрез механизмите на сегментиране и странична организация, и апаратна среда за многозадачно изпълнение. Линейното адресно пространство е 4GB;

• реален режим (R-режим). Осигурява базовата 16-разредна архитектура на микропроцесора I8086 с няколко нови разширения, като възможност за превключване в защитен режим или в режим на системно управление, обръщение към 32-разредни регистри. Максималният размер на паметта е 1 MB. При включване на захранването или Reset МП винаги се установява в този режим;

• режим на системно управление (system management mode - SMM). В този режим се осигурява механизъм за управление на консумацията на мощност и наблюдение на надеждността и производителността на процесора. В режим SMM процесорът се превключва под въздействието на външен сигнал за прекъсване SMI#. След изпълнение на стандартната процедура по съхранение на контекста на прекъснатата задача, МП се превключва към отделно адресно пространство и управлението се предава на съответната обслужваща програма. Връщането от SMM се извършва само с инструкцията *rsm*, по която МП възстановява състоянието на прекъснатата задача. Режимът е въведен за първи път в I386SL;

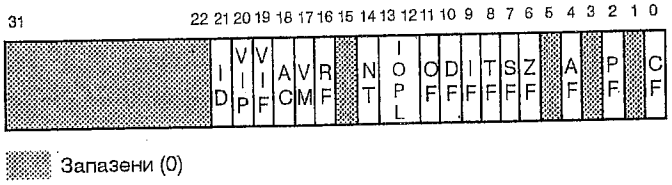
• виртуален 8086 режим (V-режим). Това не е отделен режим, а се поддържа в средата на P-режима. С виртуалния режим се осигурява възможност за изпълнение на програми за I8086 в защитена мултизадачна среда.

## 2. Системни регистри

За инициализация на микропроцесора и управление на неговата работа в архитектурата на Intel се използват системни регистри и системни флагове, включени в регистъра на флаговете EFLAGS. В състава на системните регистри влизат: управляващи регистри (CR0, CR2, CR3, CR4), регистри за тестване, регистри за реализация на сегментирането (GDTR, LDTR, IDTR), регистър на задачите (TR), специфични за модела регистри - MSR.

### 2.1. Регистър на флаговете EFLAGS

Регистърът на флаговете (фиг.5.1) е 32-разреден и съдържа две групи флагове - флагове на състояние и системни (управляващи) флагове.




Фиг. 5.1. Регистър на флаговете

Флаговете на състояние, наречени още аритметични флагове, отразяват особеностите на изпълнение на инструкциите. Назначението на тези флагове е:

• CF (Carry Flag) - флаг за пренос. Вдига се в единица при пренос или заем от старшия бит на резултата;

• PF (Parity Flag) - флаг за четност. Установява се в 1, ако младшият байт на резултата съдържа четен брой единични битове;

• AF (Auxiliary carry Flag) - флаг за спомагателен пренос. Установява се в единица при пренос или заем от младшата тетрада на резултата;

• ZF (Zero Flag) - флаг за нула. Установява се в единица при нулев резултат;

• SF (Sign Flag) - флаг за знак. Състоянието му съвпада със състоянието на старшия (знаков) бит на резултата;

• OF (Overflow Flag) - флаг за препълване. Установява се в единица, ако резултатът излиза извън диапазона за представимост на числата (прекалено голямо положително или твърде малко отрицателно число).

От флаговете за състояние само бит CF може да се модифицира директно с инструкциите ***stc***, ***clc*** и ***cmc***. Един или повече флагове на състояние се използват от инструкциите: за условен преход ***Jxy***, условно установяване на байт ***SETxy***, условни цикли ***LOOPxy*** и условен обмен на данни ***CMOVxy***, където с *x* е обозначен тестваният флаг(ове), а с *y* - проверяваното състояние на този флаг(ове).

• DF (Direction Flag) - флаг за посока. Този флаг задава посоката при обработка на низове с инструкциите ***movs***, ***cmps***, ***scas***, ***lods***, ***stos***, ***ins***, ***outs***. След изпълнение на инструкция за обработка на низове при DF=0 автоматично се инкрементират, а при DF=1 автоматично се декрементират индексните регистри ESI и EDI, адресиращи низовете. Флагът DF се установява в единица с инструкцията ***std*** и нулира с инструкцията ***cld***.

Системните флагове въздействат върху различни аспекти от работата на процесора. Те управляват входа и изхода, маскирането на апаратните прекъсвания, тестването, превключването на задачите и виртуалния режим:

• TF (Trace Flag) - флаг за тестване. При TF=1 се задава стъпков режим на работа на процесора, т.е. след изпълнение на всяка инструкция се генерира изключение за тестване;

• IF (Interrupt Flag) - флаг за прекъсване. При нулиране на IF се забранява обработката на маскируемите прекъсвания по вход INTR. Върху флага IF може да се въздейства директно с инструкциите ***cli*** и ***sti***, и индиректно с инструкциите ***popf***, ***popfd*** и ***iret***;

• IOPL (I/O Privilege Level) - двубитово поле, задаващо нивото на привилегия за достъп до входно-изходното пространство. Изпълняващата се задача може да изпълнява операции за вход-изход само, ако има ниво на привилегия, по-високо или равно от нивото, зададено чрез IOPL. Това поле е съставна част от механизма на защита на МП и действа в P-режим. То може да бъде модифицирано само неявно с инструкциите ***popf*** и ***iret***;

• NT (Nested Task) - флаг за вложена задача. Флагът NT действа в P-режим и управлява верижното извикване и прекъсване на задачите. МП вдига този флаг при извикване на задача с инструкцията ***call***, чрез прекъсване или чрез изключение. Състоянието на NT се проверява и модифицира при връщане от задача с инструкцията ***iret***. При NT=1 се осъществява превключване на задача, а при NT=0 се изпълнява нормално връщане от прекъсване;

• RF (Resume Flag) - флаг за възобновяване. Управлява реакцията на МП по отношение на обработката на изключителни събития при тестване. Флагът RF действа съвместно с регистрите за тестване;

• VM (Virtual Mode) - виртуален режим. При VM=0 процесорът може да работи или в R- или в P-режим, а при VM=1 преминава в V-режим и емулира програмната среда на МП I8086;

• AC (Alignment Check) - контрол на изравняването. Установяването в 1 на флаг AC (при вдигнат флаг AM на управляващия регистър CR0) разрешава контрола за изравняване при обръщение към паметта. Изключение възниква при обръщение към неизравнен операнд (например 16-разреден операнд, разположен на нечетен адрес или 32-разреден операнд, неразположен на адрес кратен на 4);

• VIF (Virtual Interrupt) - флаг за виртуално прекъсване. Съдържа виртуалния образ на флага IF. Използва се в съчетание с флага VIP. МП разпознава флага VIF само ако или флагът VME или флагът PVI в управляващия регистър CR4 са установени в 1 и значението на IOPL е по-малко от 3;

• VIP (Virtual Interrupt Pending) - флаг за прието виртуално прекъсване. Установява се програмно и индицира, че има прието, но още необработено виртуално прекъсване. МП чете, но не модифицира флага VIP. Използва се в съчетание с флага VIF и за него важат същите условия за разпознаване.

Нулирането на този флаг означава, че няма необработени прекъсвания, генерирани от задачите в разширения виртуален режим VME (Virtual Mode Extension);

• ID (IDentification) - флаг за идентификация. Възможността за програмна промяна на този флаг означава, че МП поддържа инструкция *cpuid* за идентификация на процесора.

Флаговете VIP и VIF са въведени за първи път в Pentium и са достъпни само за процесори, които поддържат разширен виртуален режим VME.

### 2.2. Управляващи регистри

Управляващите регистри (фиг.5.2) служат за определяне режима на работа на МП и състоянието на вътрешните му компоненти (FPU, кеш, устройство за странично преобразуване). Те включват:

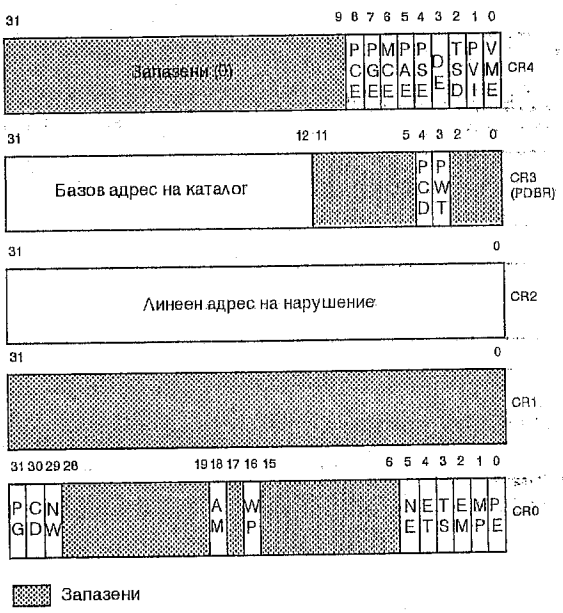
• CR0 - съдържа системни управляващи флагове, които управляват режима на работа и състоянието на процесора;

• CR1 - резервиран за бъдещи разширения;

• CR2 - съдържа линейния адрес, предизвикал последното странично нарушение;

• CR3 - съдържа физическия базов адрес на каталога на страниците. Флаговете PCD и PWT управляват кеширането на каталога във вътрешната кеш памет за данни;

• CR4 - съдържа група флагове, разрешаващи някои архитектурни разширения в P-режим.




Фиг. 5.2. Управляващи регистри

В защитен режим управляващите регистри могат да бъдат четени с инструкцията ***move*** от всяко привилегировано ниво. Зареждането им с ново съдържание обаче може да става само от най-високото (нулево) ниво на привилегия. Флаговете в регистрите CR0 и CR4 имат следното функционално предназначение:

• флагове за управление на страницирането:

• PG (Paging) - CR0[31]. Разрешава (PG=1) или забранява (PG=0) механизма на страницираме в защитен режим. Вдигането на PG в друг режим (PE=0) предизвиква изключение #GP (нарушение на защитата);

• PSE (Page Size Extensions) - CR4[4]. Установен в 1 флагът разрешава използване на страници с размер 4MB;

• PGE (Page Global Enable) - CR4[7]. Разрешава (PGE=1) използването на глобални страници. Въведен е в процесорите P6;

- флагове за управление режимите на кеширане:
  - CD (Cache Disable) - CR0[30]. При нулирани флагове CD и NW кеширането във вътрешните кеш памети на МП е разрешено;
  - NW (Not Write_trough) - CR0[29]. При нулирани флагове CD и NW при попадение в кеш за Pentium и P6 е разрешен обратен запис, а за I486 - едновременен запис в паметта;
- флагове за работа с паметта:
  - AM (Alignment Mask) - CR0[18]. При AM=1 е разрешен контрол за изравняване на операндите в паметта. Необходимото допълнително условие за тази цел е: AC=1, текущо ниво на привилегия - 3 и P- или V-режим на работа на МП;
  - WP (Write Protect) - CR0[16]. Забранява (WP=1) на системни програми (с привилегия супервайзор) да извършват запис в разрешени само за четене потребителски страници;
  - PAE (Physical Address Extension) - CR4[5]. Разрешава (PAE=1) 36 битов физически адрес;
- флагове за разширение на виртуалния режим:
  - VME (Virtual-8086 Mode Extensions) - CR4[0]. Разрешава (VME=1) подобрена обработка на прекъсвания и изключения за V-режим с цел повишаване на производителността. Осигурява апаратно поддържане на флага VIF за увеличаване на надеждността при изпълнение на програми за I8086 в мултизадачна и мултипроцесорна среда;
  - PVI (Protected_Mode Virtual Interrupts) - CR4[1]. Разрешава (PVI=1) в защитен режим действието на флага VIF.
- флагове за работа с копроцесор:
  - NE (Numeric Error) - CR0[5]. Разрешава (NE=1) вътрешния механизъм за съобщаване за грешка от вътрешното устройство с плаваща запетая FPU. При NE=0 се установява PC стил за съобщаване на грешка от копроцесор чрез сигналите на МП IGNNE#и FERR#;
  - ET (Extension Type) - CR0[4]. Не се използва в Pentium и P6, а в I486 и I386 индицира поддържане на копроцесор I387DX;
  - EM (Emulation) - CR0[2]. EM=1 индицира отсъствие, както на вътрешно FPU, така и на външен копроцесор. Инструкциите с плаваща запетая и MMX не се разпознават и предизвикват изключение #UD (невалиден код на операция);
  - MP (Monitor Coprocessor) - CR0[1]. Управлява генерирането на изключение #NM (недостъпно устройство) при инструкциите *wait* и *fwait* в зависимост от състоянието на флага TS;
- други флагове:
  - PE (Protection Enable) - CR0[0]. При PE=1 разрешава защитен режим, а при PE=0 - реален режим;
  - TS (Task Switched) - CR0[3]. МП вдига флага TS при всяко превключване на задача. Този флаг позволява да се задържи съхраняването на контекста на FPU до появата на инструкция с плаваща запетая в новата задача;
  - TSD (Time Stamp Disable) - CR4[2]. TSD=1 разрешава използването на инструкцията *rdtsc* само от нулево ниво на привилегия, а при TSD=0 тази инструкция може да бъде използвана от всяко привилегировано ниво. Инструкцията *rdtsc* чете в регистрите EDX:EAX брояча на време;
  - * MCE (Machine Check Enable) - CR4[6]. Разрешава (MCE=1) генериране на изключение от схемите за машинен контрол;
  - DE (Debugging Extension) - CR4[3]. Разширява възможностите на МП за тестване.

### 2.3. Специфични за модела на микропроцесора регистри

В архитектурата на Intel е включен масив от 64 разредни регистри със специализирано предназначение, наречени MSR (Model Specific Registers). MSR са достъпни главно за ОС, т.е. за програми, изпълнявани на привилегировано ниво 0. Тези регистри се използват за разширените възможности за тестване, за наблюдение на производителността на процесора, за целите на вътрешния контрол на процесора, за разширените възможности за управление на паметта чрез обособяване на различни типове области в нея. Броят и функциите на регистрите MSR варират в различните членове на фамилията Intel.

### 3. Управление на паметта в защитен режим

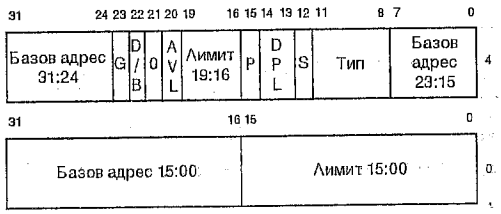
В Intel архитектурата се използват два механизма за управление на паметта - сегментиране и странициране. И двата механизма целят да осигурят изолиране на адресните пространства на отделните задачи (програми) в многозадачна (мултипрограмна) среда. Използваните защити се базират на нива на привилегии, тип на съхраняваната информация, права на достъп, размер на адресираната област.

#### 3.1. Сегментация на паметта

Адресируемата област на МП, наречена линейно адресно пространство, се разделя на по-малки защитени области - сегменти. В мултипрограмна среда всяка програма работи със собствен набор сегменти. Всеки сегмент се характеризира с набор атрибути, които задават разположението и заемания обем в паметта, тип и права на достъп до информацията в него, ниво на привилегия и др. С всеки сегмент е свързан дескриптор, съхраняващ значението на неговите атрибути.

##### 3.1.1. Сегментни дескриптори

Достъпът до данните в даден сегмент се осъществява чрез неговия дескриптор. Дескрипторът е 8 байтова структура, която съдържа следната информация за сегмента (фиг.5.3):




Фиг. 5.3. Структури от данни за сегментно преобразуване на адреса.

• базов адрес (байтове 2,3,4 и 7). Задава началния адрес на сегмента в 4GB линейно адресно пространство на МП. С цел оптимизация на производителността се препоръчва този адрес да бъде изравнен на 16 разредна граница;

• лимит. 20-разредният лимит задава размера на сегмента, който се изчислява в зависимост от значението на бита за грануларност G (Granularity). При G=0 размерът на сегмента е в границите от 1 байт до 1 MB. При G=1 размерът се изчислява със стъпка 1 страница - 4KB и е в границите от 4KB до 4GB, т.е. може да покрие цялото линейно адресно пространство. МП използва лимита в зависимост от типа на сегмента - данков или стеков. При данков сегмент допустимият адрес е в границите *базов адрес ÷базов адрес + лимит*. При стеков сегмент допустимият адрес е в границите *максимален адрес ÷базов адрес + лимит*.

• тип. Определя типа на достъп до информацията в сегмента и посоката на нарастването му. Интерпретацията на това поле зависи от типа на дескриптора, задаван чрез бита S (System): S=0 - системен дескриптор, S=1 - дескриптор на данков или кодов сегмент. Полето за тип при S=1 задава следните права за достъп:

000B - сегмент данни, разрешен само за четене
001B - сегмент данни, разрешен за четене и запис
010B - стеков сегмент, разрешен само за четене (на практика не се използва)
011B - стеков сегмент, разрешен за четене и запис
100B - кодов сегмент, разрешен само за изпълнение
101B - кодов сегмент, разрешен за изпълнение и четене
110B - подчинен кодов сегмент, разрешен само за изпълнение
111B - подчинен кодов сегмент, разрешен за изпълнение и четене

В регистър CS например, не трябва да се зареждат селектори към сегменти с типове 0÷3, тъй като това са данкови сегменти. Никоя програма не може да модифицира сегмент, разрешен само за четене. Сегментите с код, за които е разрешено само изпълнение, не могат да се четат, което препятства несанкционираното копиране на програми. Нарушаването на тези правила предизвиква изключение #GP (нарушение на защитата).

Стековите сегменти се различават от сегментите с данни с посоката на разширение на адреса

и интерпретацията на лимита. При стековите сегменти адресът се разширява надолу - от големи към малки адреси, но трябва да бъде по-голям от лимита на сегмента.

• достъп - бит A (Accessed). Битът A се установява в 1 автоматично от МП, когато се осъществява обръщение към сегмента. Използва се от ОС за избиране на сегментите за извеждане върху диска при свопинг;

• ниво на привилегия DPL (Descriptor Privilege Level). Задава нивото на привилегия на сегмента, което може да бъде в диапазона от 0 (най-висока привилегия) до 3 (най-ниска привилегия);

• присъствие P (Present). Индицира дали сегментът е резидентен (присъстващ) в паметта (P=1) или не (P=0). При зареждане в селекторен регистър на селектор към нерезидентен сегмент се генерира изключение #NP (отсъстващ сегмент). В полетата за базов адрес и лимит в дескриптора на нерезидентен сегмент ОС може да записва системна информация за сегмента;

• размер по премълчаване D (Default size). Въведен е за съвместимост с I286 и се интерпретира в зависимост от типа на сегмента. За данков сегмент D=1 означава 32 разредни операнди, а D=0 - 16 разредни операнди. При кодов сегмент D определя размера по премълчаване на ефективния адрес и размера на адресираните операнди. D=1 определя 32 разреден адрес и 32/8 разредни операнди, а D=0 - 16 разреден адрес и 16/8 разредни операнди. За промяна на значението по премълчаване се използват префиксите 66h и 67h. За стековите сегменти D определя размера на стековия указател, използван при неявните операции със стек - ***push***, ***pop***, ***call***. D=1 определя 32 разреден указател, съхраняван в ESP и горна граница на стека 4GB (FFFFFFFFh), а D=0 - 16 разреден указател в SP и горна граница 64KB (FFFFh);

• свободен за използване от системата бит AVL (AVaiLable).

Сегментните дескриптори типично се създават от компилиращи, свързващи и зареждащи програми или от ОС, но не и от приложните програми.

### 3.1.2. Дескрипторни таблици

Дескрипторите на сегментите за удобство при ползване се съхраняват в специални структури, наречени дескрипторни таблици. Максималният размер на една таблица е 64KB, т.е. 8192 различни дескриптора. В МП се използват три типа дескрипторни таблици:

GDT (Global Descriptor Table) - глобална дескрипторна таблица. Тази общосистемна таблица съдържа дескрипторите на общите сегменти, които могат да се използват от всички задачи в системата. За локализация на таблицата GDT се използва специален регистър GDTR, който съдържа 32 разредния линеен базов адрес на таблицата и 16 разреден лимит. Първият елемент на GDT, имащ нулев сегментен селектор, не се използва от МП. Зареждането на сегментните регистри CS и SS с нулев селектор предизвиква изключение #GP. Опитът да се използва нулев селектор, зареден в другите селекторни регистри, също предизвиква изключение.

IDT (Interrupt Descriptor Table) - дескрипторна таблица на прекъсванията. Общосистемна таблица, съдържаща дескрипторите на специални системни обекти (S=0 в дескрипторите). Тези обекти, наречени шлюзове, определят входните точки в процедурите за обработка на прекъсвания и изключения. За локализация на таблицата IDT се използва специален регистър IDTR, който съдържа 32 разредния линеен базов адрес на таблицата и 16 разреден лимит.

LDT (Local Descriptor Table) - локална дескрипторна таблица. В мултизадачна система за всяка задача може да се построи локална дескрипторна таблица. Тя определя сегментите, достъпни само за тази конкретна задача. За съхранение на таблиците LDT се използват сегменти, чиито дескриптори се съхраняват в таблицата GDT. За локализация на таблицата LDT е предназначен 16 разреден регистър LDTR, който съдържа селектора на дескриптора на LDT на текущата активна задача. С LDTR е свързан разширителен регистър, наречен кеш-регистър, в който се съхранява дескриптора на LDT на текущата задача.

С регистрите на дескрипторните таблици оперират следните привилегировани инструкции на МП: ***lgdt m48*** и ***lidt m48*** зареждат регистрите GDTR и IDTR съответно, а инструкциите ***sgdt m48*** и ***sidt m48*** съхраняват съдържанието на тези регистри в паметта. Операндите на тези инструкции са 48 разредни данни в паметта, младшите 16 разреда на които задават лимита, а останалите 32 разреда - линейния базов адрес на съответната таблица. Инструкциите ***lldt r16/m16*** и ***sldt r16/mem16*** служат съответно за зареждане и съхраняване на *ldtr*. Операндът е 16 разреден регистър или 16 разредни данни в паметта, задаващи валиден или празен (0000h) селектор.

### 3.1.3. Сегментни селектори

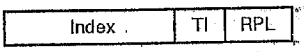
Достъпът до дескрипторите се осъществява чрез 16 разредните сегментни регистри на МП. Съдържанието на сегментен регистър се нарича селектор (фиг. 5.4 а). Двата младши разреда на селектора определят заявеното ниво на привилегия (RPL), което се използва в механизма на защита. Битът TI (Table Indicator) определя таблицата, в която е разположен дескрипторът: TI=0 указва таблицата GDT, а TI=1 - таблицата LDT. Старшите 13 разреда на селектора (поле Index) определят нужния дескриптор в дескрипторната таблица.

### 3.1.4. Сегментно преобразуване на адреса

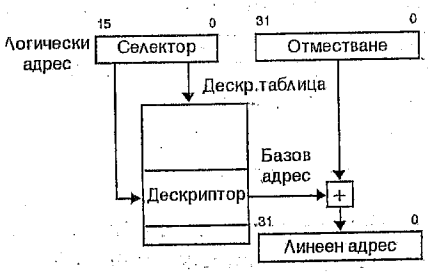
Преобразуването на адреса преминава през следната последователност от действия:

- изчисляване на ефективния (логически) адрес;
- определяне на дескрипторната таблица и номера на дескриптора от селектора в съответния сегментен регистър;
- извличане на дескриптора на сегмент от адрес базов_адрес(GDT/LDT)+Index*$2^3$;
- получаване на линейния адрес чрез сумиране на ефективния адрес и базовия адрес на сегмента от неговия дескриптор.

За да се избегнат обръщенията към паметта за извличане на дескриптора на сегмента при следващите обръщения към него, се използва кеширане на дескрипторите. За целта с всеки сегментен регистър, както и с регистъра LDTR, е асоциран 64 разреден кеш-регистър, наречен още регистър-сянка. Кеш-регистрите са невидими и недостъпни за програмите. При зареждане на селектор в сегментен регистър МП автоматично извлича и зарежда в неговия кеш-регистър съответния дескриптор.




а) формат на селектор




б) формиране на линеен адрес

Фиг. 4.4. Сегментно преобразуване

### 3.1.5. Инструкции за работа с дескриптори

Значенията на селекторите в сегментните регистри са достъпни на програмата, а чрез тях стават достъпни и дескрипторите на съответните сегменти. За опростяване на анализа на атрибутите на достъпните за дадена програма сегменти са предвидени специални инструкции:

`lar r16/32, r16/m16` - зарежда в регистъра-получател правата на достъп на сегмента (разреди 47-40 и битовете G, D, X U на дескриптора), указван от адресирания с операнда-източник селектор.

`lsl r16/32, r16/m16` - зарежда в регистъра-получател лимита на сегмента, указван от адресирания с операнда-източник селектор.

`verr r16/32` - проверява достъпността за четене на сегмента, указан от адресирания селектор.

`verw r16/32` - проверява достъпността за запис на сегмента, указан от адресирания селектор.

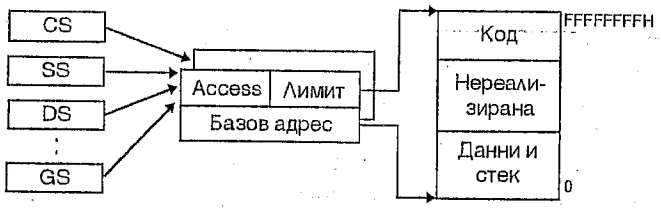
Инструкциите `verr` и `verw` вдигат флага ZF=1, ако е разрешено съответното право на достъп.

### 3.1.6. Сегментни модели

Основните сегментни модели за организация на паметта са:

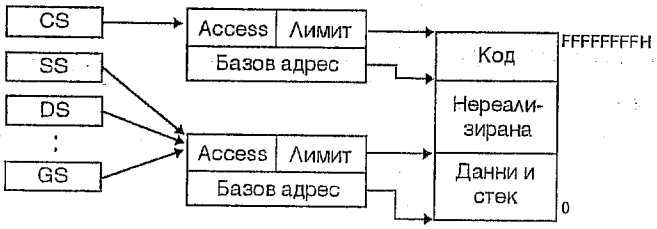
**Базов плоск модел.** Това е най-прост за реализация модел, при който системните и приложни програми имат достъп до непрекъснато, несегментирано адресно пространство. Моделът скрива

механизма на сегментиране, както за приложните, така и за системните програмисти. За реализацията му са необходими поне два сегментни дескриптора - за достъп до кодов сегмент и за достъп до данков сегмент. И двата сегмента обхващат цялото линейно пространство, т.е. имат една и съща стойност за базов адрес - 0 и лимит - 4GB (фиг. 5.5). Не се осъществява апаратна защита при непопадане на адрес в реална физически достъпна област памет.



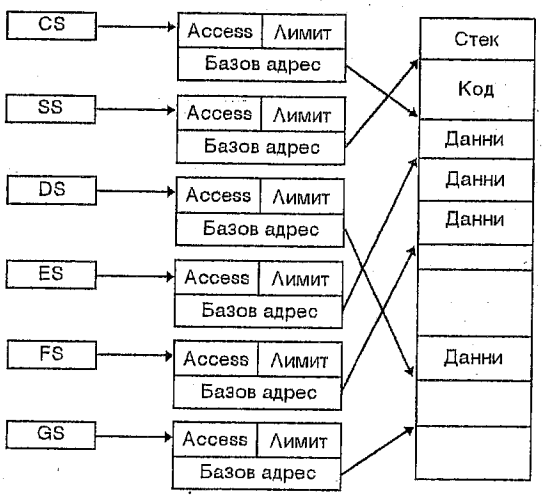

Фиг. 5.5. Базов плосък модел

**Защитен плосък модел.** Подобен е на базовия, но в сегментите се включват само области от адреси, реално съществуващи във физическата памет (фиг. 5.6). При попадение в несъществуваща област се генерира изключение #GP (общо нарушение на защитата). Моделът може да бъде усложнен за осигуряване на по-добри възможности за защита. Например чрез механизма на страницираме може да се осигури изолиране на системното от потребителското адресно пространство. Чрез добавяне на различни структури от страници за различните задачи се постига защита на приложенията едно от друго. Такъв подход се използва в редица популярни операционни системи (например Unix).




Фиг. 5.6. Защитен плосък модел

**Многосегментен модел.** При него се използват пълните възможности на механизма на сегментиране за осигуряване на апаратна защита на код, данкови структури и задачи. Всяка програма (задача) има на разположение собствена таблица от сегментни дескриптори (LDT) и свои собствени сегменти. Достъпът до всички сегменти се контролира апаратно.




Фиг. 5.7. Многосегментен модел

### 3.2. Странична организация на паметта

В защитен режим процесорите поддържат странично преобразуване на адреса, което внася второ ниво на преобразуване на логическия адрес във физически. Основното приложение на страничното преобразуване е свързано с организацията на виртуална памет, която позволява достъп до по-голямо пространство памет от реално реализираната физическа памет посредством използване на дисково пространство.

За разлика от сегментирането, при което базовият обект памет е блок с произволен размер, наречен сегмент, страницирането използва като базов обект памет блок с фиксиран размер, наречен страница. Ако се използва само сегментиране, структурите от данни, представени във физическата памет трябва да съдържат всичките си елементи в паметта. Ако се използва и страницираме, структурите могат да бъдат частично в паметта и частично върху диска. Страницирането е напълно невидим за приложните програми механизъм. За страниците също се прилагат защитни механизми: две нива на привилегия и защита от запис.

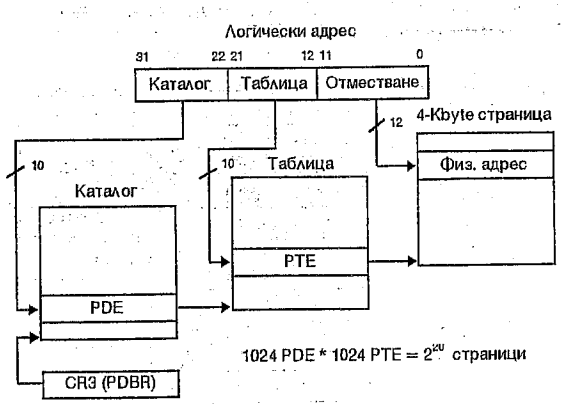
При разрешено странично преобразуване линейното адресно пространство на МП (4GB виртуална памет) се разделя на 1M страници по 4KB. Физическата памет също се разделя на страници със същия размер, наречени странични кадри. Виртуалната памет е много по-голяма от физически реализираната памет. Следователно във всеки един момент само част от всички страници на виртуалната памет присъстват във физическата памет. Всяка виртуална страница може да бъде заредена във всяка физическа страница. Процесорът автоматично генерира изключение #PF (странично нарушение) при обръщение към страница неприсъстваща във физическата памет. При обработката на това изключение ОС зарежда нужната страница от външната памет. Страницата става достъпна посредством базовия адрес на зареждането ѝ в паметта.

#### 3.2.1. Информационни структури

Информацията, която МП използва, за да преобразува линейните адреси във физически е включена в даннови структури на две йерархични нива (фиг. 5.8):

- каталог на страниците, представляващ масив (таблица) от 1024 32-разредни елемента - PDE (Page Directory Entry). Всеки PDE адресира подчинена таблица на страници. Каталогът на страниците е достъпен чрез управляващия регистър CR3, наречен PDBR (Page Directory Base Register). BDBR съдържа 20-разредния базов физически адрес на каталога.
- таблица на страници. Масив от 1024 32-разредни елемента - PTE (Page Table Entry), всеки от които адресира физическа страница в паметта.

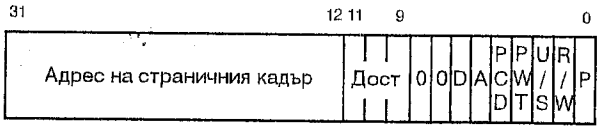
Каталогът и таблиците на страниците са с размер една 4KB страница.




Фиг. 5.8. Двуетапно преобразуване на линейния адрес във физически

Елементите PDE и PTE имат еднакъв формат (фиг. 4.9) и задават следните атрибути на страниците:

- присъствие в оперативната памет - бит P (Present). При P=1 страницата е в паметта и е валиден форматът на PTE/PDE от фиг.5.9, а при P=0 страницата не е в паметта и останалата част от PTE/PDE е достъпна за ОС за съхранение на системна информация. В случай на обръщение към PTE/PDE, съдържащ P=0, се генерира изключение #PF;




Фиг. 5.9. Формат на PDE и PTE

• защита. Бит R/W служи за защита от запис. При R/W=0 страницата е достъпна само за четене, а при R/W=1 за четене и запис. Бит U/S определя нивото на привилегия на страницата: U/S=0 (supervisor) - за ОС и други системни програми, U/S=1 (user) - потребителско ниво за приложни програми и данни;

• управление на кеширането. Битовете PCD (забрана на кеширането) и PWT (кеширане с едновременен запис) се използват за управление на кеширането на ниво страници;

• обръщение. Битът A (Accessed) автоматично се вдига в 1 от МП при обръщение към страницата и таблицата на страници за запис или четене. ОС използва този бит за реализация на стратегията за замяна на страници;

• модификация. Битът D в PTE автоматично се установява в 1 от МП при запис в страницата. Той служи за определяне на съответствието на съдържанието на страницата в паметта и копието на същата страница върху диска. В PDE битът D не се използва;

• базов адрес. Разряди 12-31 задават базовия физически адрес на страницата или таблицата в паметта;

• системна информация. Битове 9-11 са предоставени на системния програмист за специфична системна информация за страницата.

### 3.2.2. Преобразуване на линейния адрес във физически

Преобразуването на линейния адрес се състои от следните действия:

• старшите 10 разреда на линейния адрес (разреди 31-22), допълнени с две младши нули, се използват за индекс в каталога на страниците. Избраният чрез тях PDE определя старшите 20 разреда на базовия адрес на таблицата на страниците, които се разширяват с 12 младши нули;

• средните 10 разреда на линейния адрес (разреди 21-12), допълнени с две младши нули индексират таблицата на страниците. Избраният чрез тях PTE определя старшите 20 разреда на базовия адрес на страницата във физическата памет, които се разширяват с 12 младши нули;

• към базовия адрес на страницата се прибавя отместването от линейния адрес (разреди 11-0), в резултат на което се получава пълният 32-разреден физически адрес, по който се извършва обръщението към паметта.

При обръщение към PDE и PTE се извършват съответните проверки за присъствие (бит P), привилегия (бит U/S) и право на достъп (бит R/W).

За минимизиране на броя цикли по шината, необходими за извличане на PDE и PTE за преобразуване на адреса, в МП е включена вътрешна кеш памет, наречена асоциативен буфер за преобразуване - TLB (Translation Lookaside Buffer). В TLB се кешират използваните елементи от каталога на страниците и от таблиците на страници. TLB удовлетворява болшинството от заявките за четене на PDE и PTE. Допълнителни цикли по шината се изискват само при обръщение към нова страница. Информацията в TLB е достъпна само за програмите от нулево ниво на привилегия. Pentium и P6 имат отделни TLB буфери за вътрешните кеш за данни и инструкции. Елементите в TLB автоматично се обявяват за невалидни при превключване на задача и при зареждане на CR3 с нов адрес на каталог на страници. От Pentium Pro в PTE е въведен флаг G (Global), определящ страницата като глобална. Ако този флаг е вдигнат (G=1) и глобалните страници са разрешени с флага PDE на управляващия регистър CR4, кешираният за съответната страница PTE не се унищожава при превключване на задачи или при зареждане на CR3 с нов адрес на каталог.

### 3.2.3. Режими на страницираме

Описаната по-горе схема на страницираме е основна за всички МП, започвайки от I386. От Pentium е въведена възможност за увеличаване на размера на страницата. Допустими са страници с размер 4MB и 2MB.

В съвременната Intel архитектура страницирането се управлява от три флага:

• PG (Paging) от управляващия регистър CR0 (CR0[31]) разрешава механизма на страницираме. Въведен е от I386;

• PSE (Page Size Extension) в управляващия регистър CR4 (CR4[4]) разрешава използване на страници с размер 4MB и 2MB. Въведен е от Pentium;

• PAE (Physical Address Extension) в управляващия регистър CR4 (CR4[5]) разрешава разширение физическия адрес от 32 на 36 разреда. Въведен е от Pentium Pro.

В режим на 32-разряден адрес са допустими страници с размер 4KB и 4MB, а в режим на разширена адресация - 4KB и 2MB. Кой от двата размера е използван за дадена страница се указва от бита PS (Page size) на PDE: PS=0 - задава таблица от 4KB-ви страници, а PS=1 - задава 4MB-ва (при 32-разреден адрес) или 2MB-ва (при 36 разреден адрес) страница.

При разширена адресация преобразуването на виртуалния адрес се изпълнява на три нива, като се въвежда допълнителна структура, наречена таблица от указатели към каталози на страници (Page_Directory_Pointer_Table - PDPT). Каталозите и таблиците на страниците съдържат 512 64-разредни елементи. Регистърът CR3 съдържа старшите 27 разреда на PDPT. Старшите 2 разреда на линейният адрес (31-30) задават отместването на един от 4-те елемента в PDPT. Този елемент задава базов физически адрес на каталог на страници. Разряди 29-21 на линейния адрес избират елемент от каталога на страниците, който осигурява базовия адрес на таблица на страници, а разреди 20-12 индексират елемент в тази таблица.

В схемите за странично преобразуване за страниците с размер 4MB и 2MB отсъстват таблици на страници. Избраният от каталога PDE директно задава базовия адрес на страницата, а разреди 20-0 на линейния адрес определят отместването в нея.

**Въпроси и задачи:**

1. Стартирайте симулатора sim486. Заредете в него програма за изпълнение. В произволна точка от изпълнението превключете МП в защитен режим като заредите от прозореца на системните регистри подходящо значение в тях. Каква е реакцията на МП при изпълнение на следващата инструкция? На кои регистри трябва да бъде променено значението и как, за да се превключи МП в защитен режим?

2. Съдържанието на кои регистри трябва да бъде променено, за да продължи правилно изпълнението на заредената програма след превключване в защитен режим? Направете нужните промени в регистрите и евентуално в паметта от съответните прозорци на симулатора. Проверете правилността на решението.

3. Как се изчисляват адресите на операндите и инструкциите в защитен режим? Изчислете адресите на операндите в инструкциите на вашата програма и направете проверка с изчислените адреси от симулатора.

4. Каква ще бъде реакцията на процесора, ако в произволна точка от изпълнението на програмата разрешите механизма на страницираме? Кои регистри трябва да бъдат променени и как, за да се разреши този механизъм? Опитайте да създадете необходимите структури в регистрите и паметта, за да продължи изпълнението на програмата с разрешено страницираме.

5. Как при разрешен механизъм на страницираме се определят физическите адреси на операндите и инструкциите в паметта? Обяснете влиянието на съдържанието на буферите на устройството за предварително извличане на правилността на изпълнение на програмата след разрешаване на страницирането. А на кеш паметта?

# 6. Превключване от режим с реална адресация в защитен режим

Целта на упражнението е демонстрация на последователността от действия за установяване на микропроцесора в основния му защитен режим на работа, състоящи се в построяване на необходимите структури от данни и установяване на необходимото съдържание на управляващите и сегментните регистри. Изясняват се и се изследват съпътстващите защитни механизми. Познаването на възможностите на защитения режим дава добра основа за разработване както на системен софтуер и компилиращи програми, така и на ефективни приложения.

### 1. Превключване от реален в защитен режим

След включване на захранването и начално установяване (reset) МП се установява в реален режим на работа. Превключването в защитен режим се извършва програмно. За целта превключващата програма трябва да зареди в паметта минималното количество от данновите структури на защитения режим и необходимите програмни модули, за да се осигури надеждната работа на процесора: IDT, GDT, TSS, LDT (ако е необходима), поне един каталог и една таблица на страници (ако се използва страницираме), един или повече програмни модули с необходимите процедури за обработка на прекъсвания и изключения. Превключващата програма трябва да зареди с необходимите значения системните регистри на МП - GDTR, LDTR, CR0-CR4, MTRR. Съдържанието на системните структури от данни за Р-режима зависи от типа на организацията на паметта, която ще се поддържа: плоск модел на паметта, плоск модел със страницираме, сегментен модел, сегментен модел със страницираме. Когато необходимите данновн структури, програмни модули и системни регистри бъдат заредени, МП може да се превключи в защитен режим чрез установяване в единица на флага PE (CR0[0]). Основните действия, които трябва да се изпълнят са следните:

1. забрана на прекъсванията (*cli*). По време на превключването не трябва да постъпват прекъсвания и да се генерират изключения;
2. зареждане с инструкцията *lgdt* на регистъра GDTR с базовия адрес на таблицата GDT.
3. с инструкцията ***mov cr0*** се вдига в единица флага PE, разрешаващ защитения режим;
4. веднага след вдигане на флага PE трябва да се изпълни инструкция за преход - ***far jmp*** или ***far call***. Тези инструкции изчистват конвейера и буферите на устройството за предварително извличане и зареждат селектор в регистъра CS.
5. актуализация на сегментните регистри по един от следните два начина:
   - зареждане на правилни селектори в сегментните регистри DS, SS, ES, FS и GS. Ако регистрите ES, FS и GS не се използват в тях трябва да се зареди нулев селектор;
   - изпълнение на *jmp* или *call* към задача, при което сегментните регистри автоматично се зареждат от съответния сегмент за състояние на задачата;
6. зареждане с инструкцията *lidt* на регистъра IDTR с адреса и размера на таблицата IDT за защитения режим;
7. разрешаване на прекъсванията с инструкцията *sti*.

### 2. Превключване от защитен в реален режим

Микропроцесорът се превключва обратно в режим на реална адресация чрез нулиране на флага PE в регистъра CR0 с инструкцията ***mov cr0***. Процедурата, която възстановява R-режим, трябва да изпълни следните действия:

1. Забрана на прекъсванията. С инструкцията *cli* се забраняват маскируемите апаратни прекъсвания, а немаскируемите могат да бъдат забранени с външна схема.
2. Ако е разрешено страницирането се изпълняват следните операции:
   - прехвърляне на управлението в област, за която линейните и физически адреси съвпадат;
   - GDT и LDT трябва да бъдат в област със съвпадащи линейни и физически адреси;
   - нулиране на флага PG в CR0;
   - нулиране на регистъра CR3, с което се изчистват буферите TLB.
3. Управлението се предава в достъпен за четене сегмент с лимит 64KB, като при това в CS се зарежда съответният селектор.

4. Зареждане на сегментните регистри SS, DS, ES, FS и GS със селектор към дескриптор, подходящ за реален режим:

- лимит = 64 KB (0FFFFh);
- байтова грануларност (G=0);
- даннов сегмент, достъпен за запис;
- подходящ базов адрес.

Ако не се презаредят, сегментните регистри ще останат със съдържанието си от защитения режим.

5. С инструкцията *lidt* в регистъра IDTR се зарежда указател към таблицата за прекъсване на реалния режим в 1MB адресно пространство.

6. Нулира се флага PE в CR0 за превключване в реален режим.

7. С инструкцията *far jmp* управлението се предава на програмата в реален режим. Тази операция изчиства конвейера, опашката на устройството за предварително извличане и зарежда необходимия базов адрес в CS.

8. Зареждане на останалите сегментни регистри според нуждите на кода, изпълняван в реалния режим. В неизползваните сегментни регистри се записва 0h.

9. Разрешаване на прекъсванията.

## 3. Примерна програма

Приведеният пример демонстрира стъпките, които трябва да се изпълнят за превключване на МП от реален в защитен режим и обратно. В средата на защитения режим могат да се изпълняват различни програми и процедури. В показания пример се изпълнява единично извикване на процедура с име main и се демонстрира обработка на програмно прекъсване, предизвикано от инструкцията *int 0*. Пълният код на програмата е приведен в Приложение 1.

### 3.1. Необходими структури от данни

#### 3.1.1. Сегментен дескриптор

В асемблерска програма сегментният дескриптор може да бъде дефиниран със следната структура (съгласно формата от фиг.4.1):

```
struc  segment_descriptor
   seg_length0_15     dw ?    ; младша дума на дължината на сегмента
   base_addr0_15      dw ?    ; младша дума на базов адрес
   base_addr16_23     db ?    ; младши байт на старшата дума на базовия адрес
   flags              db ?    ; тип на сегмента, привилегия и флаг за присъствие P
   access             db ?    ; старша част на дължината на сегмента
                              ; флагове за грануларност (G), размер (D) и AVL
   base_addr24_31     db ?    ; старши байт на базов адрес
ends  segment_descriptor
```

Описанието на конкретен сегмент се извършва чрез запис на съответни значения в полетата на дескриптора. Например даннов сегмент (S=1, тип=001B ), даващ достъп за четене и запис до цялото линейно адресно пространство (с базов адрес 00000000h и лимит 4GB, получаван само при G=1), с ниво на привилегия 0 (DPL=0) и размер по премълчаване 32 разреда (D=1) се описва с дескриптор със следното значение на полетата от горната структура: 0ffffh,0,0,92h,0cfh,0.

Значение на дескриптор 0ffffh,0,0,9ah,0,0 определя кодов сегмент, с базов адрес 00h, лимит 64K (G=0), размер по премълчаване 16 разряда (D=0) и ниво на привилегия 0.

#### 3.1.2. Глобална дескрипторна таблица

Програмата трябва да съдържа като минимум два сегмента - сегмент с програмата и данните в реален режим (16 битов сегмент) и сегмент за програмата и данните в защитен режим (32 битов сегмент).

```
segment code16 para public use16      ;16-битов сегмент за код и данни
   assume   cs:code16, ds:code16
   .....
ends        code16
```

```
segment code32 para public use32      ;32-битов сегмент за код и данни
   assume   cs:code32, ds:code32
   .....
ends        code32
```

За да може от тези сегменти да се изпълняват инструкции и едновременно с това да се четат и записват операнди, достъпът до тях трябва да се осъществява с по два синонимни дескриптора:

- кодов - даващ достъп до сегмента за четене и изпълнение на инструкции;
- даннов - даващ достъп за четене и запис на операнди.

Следователно GDT трябва да съдържа като минимум 5 дескриптора: нулев дескриптор (dummy_descriptor), който не описва реален сегмент и задължително е първи елемент в GDT, дескриптор на 32 разряден кодов сегмент (code32_deskriptor), дескриптор на 32 разряден даннов сегмент (gama32_deskriptor), дескриптор на 16 разряден кодов сегмент (code16_deskriptor) и дескриптор на 16 разряден даннов сегмент (code16_deskriptor). Избран е плосък модел на сегментация като най-прост за реализация. В дадения пример не е дефиниран отделен сегмент за стек. Вместо това за стек се заделят области от данновите сегменти (редове 40-41 и 127-128 в листинга на програмата).

Следва да се отбележи, че на стадий компилация не са известни базовите адреси на зареждане на сегментите в паметта, тъй като асемблерът генерира обектен код в относителни адреси (по отношение на нулевия адрес). Те стават известни след свързването на програмата. Затова първоначално в полетата за базови адреси се записват нули и дескрипторите обхващат цялото линейно пространство - базов адрес 00h и лимит 4GB за 32 разряните сегменти и базов адрес 00h и лимит 64K за 16 разрядните сегменти. Записване на конкретни стойности в полетата за базов адрес се извършва от инициализиращата програма - редове 61-89 в листинга на програмата.

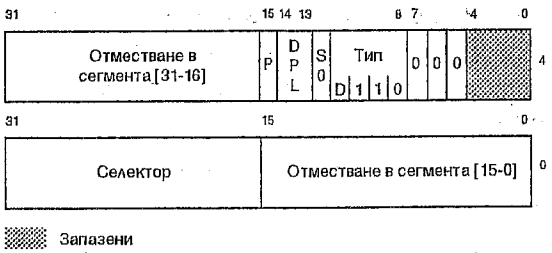
Дефиницията на GDT в програмата заема редове 135-146.

Таблицата GDT е достъпна за МП посредством регистъра GDTR. Инструкцията за неговото зареждане ***lgdt*** изисква 48 разряден операнд (ред 134), състоящ се от две части - лимитна база. 16 разрядният лимит (gdt_size) заема младшата дума, а 32 разрядният базов адрес - старшите думи.

```
; GDT data
gdt_reg        dw    gdt_size,0,0;  операнд за зареждане на GDTR
dummy_dscriptor     segment_descriptor   <0,0,0,0,0>
code32_dscriptor    segment_descriptor   <0ffffh,0,0,9ah,0cfh,0>
data32_dscriptor    segment_descriptor   <0ffffh,0,0,92h,0cfh,0>
code16_dscriptor    segment_descriptor   <0ffffh,0,0,9ah,0,0>
data16_dscriptor    segment_descriptor   <0ffffh,0,0,92h,0,0>
gdt_size=$-(offset  dummy_dscriptor)
```

### 3.1.3. Дескриптор на шлюз към прекъсване

Елементите в таблицата с дескриптори на прекъсване (IDT) определят програмните обекти, обработващи изключенията и прекъсванията. Те могат да бъдат: шлюз към извикване, шлюз към прекъсване и шлюз към задача. Шлюз към изключение или прекъсване задава косвено процедура за неговата обработка, която се изпълнява в контекста на текущата задача, а шлюз към задача косвено указва обработваща задача.




Фиг. 6.1. Формат на шлюз към прекъсване и изключение

На фиг.6.1 е показан форматът на шлюз към изключение и прекъсване. Селекторът в шлюза сочи дескриптор на кодов сегмент в GDT или LDT. Полето за отместване сочи входната точка на обработващата процедура. Шлюзовете към изключение и прекъсване се различават по значението

на полето тип и въздействието върху флага IF. При активиране на шлюз към прекъсване флагът IF се нулира, като по този начин се маскират възможни следващи прекъсвания. Шлюзът към изключение не въздейства върху флага IF.

В асемблерска програма дескрипторите на шлюзове в IDT могат да бъдат дефинирани със следната структура (съгласно формата от фиг.6.1):

```
struc interrupt_descriptor
  offset0_15       dw  ?   ; младша дума на указателя на отместването
  selector0_15     dw  ?   ; сегмент на селектора
  zero_byte        db  0   ; неизползван в този дескриптор формат
  flags            db  ?   ; байт флагове
  offset16_31      dw  ?   ; старша дума на отместването
ends interrupt_descriptor
```

Описанието на конкретен шлюз се извършва чрез запис на съответни значения в полетата на дескриптора.

### 3.1.4. Таблица на прекъсванията

За простота в таблицата на прекъсване за защитения режим е дефиниран само един дескриптор на шлюз към изключение с номер 0 (ред 159), който определя процедурата demo_int в 32 разрядния кодов сегмент чрез селектора code32_idx.

Таблицата IDT е достъпна за МП посредством регистъра IDTR. Инструкцията за неговото зареждане ***lidt*** изисква 48 разряден операнд (ред 158), състоящ се от две части - лимит и база. 16 разрядният лимит (idt_size) заема младшата дума, а 32 разрядният базов адрес - старшите думи. Конкретната стойност на базовия адрес се записва на етапа изпълнение преди превключването в защитен режим (редове 85-89 от листинга на програмата).

По премълчаване таблицата на прекъсване за реалния режим се разполага от адрес 00h и е с размер 3ffh. Особеност на реалния режим по сравнение с реалната адресация в I8086 е използването на регистъра IDTR за достъп до таблицата на прекъсване. Следователно, в реален режим IDTR трябва да е зареден с действителния базов адрес и лимита на таблицата. Операндът за зареждане на IDTR в реален режим е дефиниран в ред 44 на програмата и има значение 3ffh,0,0.

### 3.2. Превключване от реален в защитен режим

Следвайки стъпките от т. 1. редове 91-100 и 167-173 реализират превключването от реален в защитен режим и установяването на контекста на защитения режим.

След установяване на флага PE в CR0 трябва да се изпълни инструкцията *far jmp* по следния начин:

```
db    0eah
dw    offset start32, code32_idx
```

където 0eah е кодът на операция за инструкцията *far jmp*. Ако в програмата директно се напише инструкцията *far jmp*, асемблерът (TASM) се опитва да оптимизира прехода, което може да не доведе до желания резултат: изчистване на конвейера и опашката за предварително извличане и зареждане в CS и неговия кеш-регистър на селектора и дескриптора на сегмента за P-режима.

След това МП се установява в защитен режим и започва изпълнение на инструкциите от етикет ***start32***, които зареждат останалите сегментни регистри.

### 3.3. Работа в защитен режим

В защитения режим е възможно използване на всички заложени в микропроцесора възможности. В примера се извиква процедурата main, разположена в същия кодов сегмент, която от своя страна програмно генерира изключение 0.

### 3.4. Превключване от защитен в реален режим

Следвайки стъпките от т. 2. редове 177-178 и 103-116 реализират превключването от защитен в реален режим и установяване на контекста на реалния режим.

### 4. Използвани особености на асемблера TASM

Асемблерът TASM автоматично групира сегментите и разрешава обръщения към адреси (имена), разположени или декларирани в други сегменти. Например при инициализацията на GDT от сегмента в реален режим е използвано обръщение към адрес в 32-разрядния сегмент:

```
mov     [ds:code16_descriptor.base0_15],ax
```

Префиксът ds заставя TASM да разреши обръщението към адреса на code16_descriptor.base0_15, който е в сегмент code32 и към него не е възможно непосредствено обръщение от сегмента code16. Определянето и поставянето на действителните адреси на разрешените обръщения се извършва от свързващия редактор TLINK. За тази цел извикването на асемблера и свързващия редактор трябва да има следния формат:

```
tasm -zi -m2    име
tlink -3 -x     име
```

където опциите задават съответно:

- **zi** - включване на пълна диагностична информация;
- **m#** - задава броя на необходимите пасове на асемблера за разрешаване на обръщенията напред и междусегментните обръщения. В дадения случай са достатъчни 2 паса;
- **3** - разрешава 32 разрядно изпълнение;
- **x** - не се генерира файл с обръщенията.

**Забележка:**

Примерната програма може да бъде стартирана само от DOS, работещ в реален режим, а не в виртуален режим (V86). Не трябва да са активни менажерите на разширената памет, като EMM386, QEMM и др., тъй като те винаги превключват МП в режим V86.

**Въпроси и задачи:**

1. Напишете значението на полетата на дескриптор за:

• 32 разряден кодов сегмент, изпълняван на привилегировано ниво 3, с базов адрес 100h, limit 8K, достъпен за изпълнение и четене;

• 16 разряден данков сегмент, достъпен само за четене, обхващащ цялото линейно адресно пространство.

2. Компилирайте и изпълнете със симулатора за I486 програмата от Приложение 1. В какво състояние се установяват управляващите и сегментните регистри на МП след reset? Сравнете началното състояние със състоянието след превключване в защитен режим. Отбележете моментите в програмата, в които се формират нужните за работата на МП структури от данни. Кои са те? Какво е тяхното състояние в реален и защитен режим? Проследете и опишете действието на инструкцията *far jmp* (междусегментен преход - код 0eah).

3. Изследвайте защитните механизми в защитен режим:

• каква е реакцията на процесора при опит за обръщение извън границите на сегмент? За целта сменете лимита в дескриптора на данновия сегмент и включете в програмата инструкция, която да чете или записва операнд по адрес, надхвърлящ лимита на сегмента;

• каква е реакцията на МП при нарушаване правата на достъп до сегмент: запис в сегмент, разрешен само за четене, запис в кодов сегмент, четене от кодов сегмент, разрешен само за изпълнение? За целта променете типа на сегмента в неговия дескриптор и включете в програмата в защитен режим на инструкции, нарушаващи правата за достъп;

• сменете нивото на привилегия само на кодовия сегмент, изпълняван в защитен режим. Какво влияние ще окаже смяната на привилегията на изпълнението на програмата? Включете и анализирайте повторното изпълнение на инструкциите *lgdt* и *lidt*.

4. Изпълнете програмата на работния си компютър, като включите извеждане върху екрана на съобщения, показващи режима на МП.

**Упътване:**

В реален режим за тази цел може да се използва *int 21*. В защитен режим таблицата с векторите на прекъсване за реалния режим е недостъпна, поради което не могат да се използват прекъсванията

към DOS. За извеждане на съобщение в Р-режим може да се използва директен запис във видео паметта. Ако се използва текстов видеорежим с разрешаваща способност 80x25 символа и начало на видеопаметта 0B8000h, то адресът на един символ с координати X - номер на колона и Y - номер на ред се изчислява по формулата:

$$VideoAddr = 0B8000h + 2 * (80 * Y + X)$$

За всеки символ във видеопаметта се записват по два байта: байт с ASCII кода на символа, който се изобразява и байт за атрибути на символа. Съдържанието на байта за атрибути е следното:

| бит | значение |
|---|---|
| 0-2 | цвят на символа |
| 3 | интензивност |
| 4-6 | цвят на фона на символа |
| 7 | мигане/интензитет на фона |

Възможни са 8 различни цвята за символ и фон на символа, а с добавяне на интензитета - 16:

| код | цвят | код | цвят |
|---|---|---|---|
| 0000 | черен | 1000 | тъмно сив |
| 0001 | син | 1001 | ярко син |
| 0010 | зелен | 1010 | ярко зелен |
| 0011 | циан | 1011 | ярко циан |
| 0100 | червен | 1100 | ярко червен |
| 0101 | виолетов | 1101 | ярко лилав |
| 0110 | кафяв | 1110 | жълт |
| 0111 | светло сив | 1111 | ярко бял |

Примерна процедура за извеждане на примерен текст от вида

```
pm_msg  db   0,0,1,0,1fh,'Now in Protected Mode',0
```

Преди извикване на процедурата указател към началото на извеждания низ трябва да бъде зареден в двойката регистри DS:ESI.

```
proc    write_msg
   push ax esi edi es
   mov  ax,core32_idx     ; в Р-режим трябва да се определи сегмент
                          ; обхващащ видеопаметта с примерен селектор
                          ; core32_idx
   mov  es,ax
   movzx  di,[esi+2]      ; взема позицията Y
   imul edi,160
   add  di,[esi]          ; добавя позицията X
   add  di,[esi]
   add  edi,0b8000h       ; добавя началния адрес на текстовия екран
   mov  ah,[esi+4]        ; взема байта с атрибути
   add  esi,5
write_loop:
   mov  al,[esi]
   or   al,al             ; край на извеждания символен низ?
   jz   loop_end
   inc  esi
   mov  [es:edi],ax
   inc  edi
   inc  edi
   jmp  write_loop
loop_end:
   pop  es edi esi ax
   ret
endp    write_msg
```

5. Разширете примерната програма, като добавите ваша процедура по избор, дефинирана в отделен сегмент. Осъществете извикване на тази процедура след превключване в Р-режим (непосредствено преди или след main). Как ще протече изпълнението, ако сегментът с тази процедура има по-ниско ниво на привилегия?

# 7. Многозадачност. Защита по привилегии

Целта на упражнението е разглеждане на механизма за поддържане на многозадачна работа в микропроцесорите от фамилията Intel, състоящ се от специални структури от данни за съхраняване и възстановяване на контекста на задачите и осигуряване на защита по привилегии.

## 1. Защита по привилегии

В МП са дефинирани 4 привилегировани нива, номерирани от 0 до 3, като най-привилегировано е ниво 0, а най-ниско привилегировано е ниво 3. Задачите, в зависимост от присвоеното им привилегировано ниво, имат различни права над ресурсите и инструкциите на МП. МП в Р-режим постоянно контролира дали текущата програма има достатъчна привилегия за: изпълнение на определени инструкции, обръщение към данните на други програми, предаване на управлението към друг код на програма. За реализиране на този контрол МП разпознава следните типове нива на привилегии:

- текущо ниво на привилегия (CPL) - ниво на привилегия на текущо изпълняваната задача или програма. Съхранява се в битове 0-1 на селектора в кодовия сегментен регистър CS. Нормално CPL е равно на нивото на привилегия на кодовия сегмент, от който се извличат инструкциите за изпълнение, задавано в полето DPL на неговия дескриптор. Процесорът променя CPL при предаване на управлението към кодов сегмент с различно ниво на привилегия;
- дескрипторно ниво на привилегия (DPL) - ниво на привилегия на сегмент или шлюз. Съхранява се в полето DPL на дескриптора на сегмента или шлюза. Интерпретира се в зависимост от типа на дескриптора: DPL на дескриптора на данков сегмент определя минималното ниво, което трябва да притежават програми или задачи, за да се обръщат към него;
- заявено ниво на привилегия (RPL). Всеки селектор определя само един дескриптор, но даден дескриптор може да се идентифицира чрез няколко селектора. Младшите два бита на селектора съдържат полето RPL, което показва нивото на привилегия на източника на селектора.

### 1.1. Защита на достъпа до данни

МП контролира по привилегия обръщенията към данните, като не разрешава обръщение към по-привилегировани данни от изпълняваната програма. Основното правило за достъп до данни може да се изрази чрез **CPL <= DPL**. Проверката на привилегиите се извършва при зареждане на селектор в един от сегментните регистри за данни - DS, ES, FS и GS. Ако значението на DPL на указвания от селектора сегмент е числено по-малко от CPL, МП не зарежда селектора и генерира изключение общо нарушение на защитата (#GP). Даже ако програмата има достатъчни привилегии за достъп до данните, достъпът се отменя в случай, че RPL на селектора е числено по-голям от CPL (**RPL>CPL**). По този начин се забранява привилегирована програма да осъществи достъп до данни от името на по-ниско привилегирован код. Пълното право на достъп до данков сегмент може да се определи като: **max(CPL,RPL) <= DPL**. Следователно полето RPL действа и се отчита само, ако числено е по-голямо от CPL.

Зареждане на селектор в стековия сегментен регистър SS е разрешено само, ако значението на DPL на стековия сегмент е точно равно на CPL. За да не се нарушава това правило при смяна на нивото на привилегия, МП автоматично превключва и стека.

### 1.2. Защита на кодови сегменти

Управлението на МП от един поток инструкции към друг се предава посредством инструкциите *jmp*, *call*, *ret*, *iret*, *int n*. Сегментният селектор за новия поток инструкции се зарежда в кодовия сегментен регистър CS. При това МП извършва проверки на сегментния дескриптор по отношение на лимит, тип и привилегии. Предаване на управлението с *jmp* или *call* може да бъде директно или индиректно през т.н. шлюзове (gates).

#### 1.2.1. Директно предаване на управлението

При директното предаване преходът може да бъде в рамките на текущия кодов сегмент (близък преход) или към друг сегмент (междусегментен преход). В първия случай МП не прави проверка на

привилегиите. В случай на междусегментен преход операндът на инструкциите *call* и *jmp* съдържа селектор към дескриптора на кодовия сегмент, към който се предава управлението. Програмите и задачите могат да се обръщат директно и да предават управлението само към код на същото привилегировано ниво, т.е. CPL трябва да бъде равно на DPL на дескриптора на целевия кодов сегмент, (**CPL = DPL**). Това ограничение може да бъде разчупено посредством подчинени кодови сегменти. При обръщение към подчинен кодов сегмент CPL на извикващата програма или задача трябва да бъде числено по-голямо (по-ниска привилегия) или равно на DPL в дескриптора на подчинения сегмент (**CPL>=DPL**). В този случай DPL на подчинения кодов сегмент задава максималното ниво на привилегия, което трябва да притежава програма, за да предаде управлението към него. Извикването на подчинен кодов сегмент не предизвиква смяна на CPL, т.е. подчиненият код се изпълнява в контекста на извикващата програма. Подчинените кодови сегменти могат да бъдат използвани за оформяне на програмни модули като библиотеки, сервизни програми на ОС и др., които могат да бъдат извиквани от различни нива на привилегия.

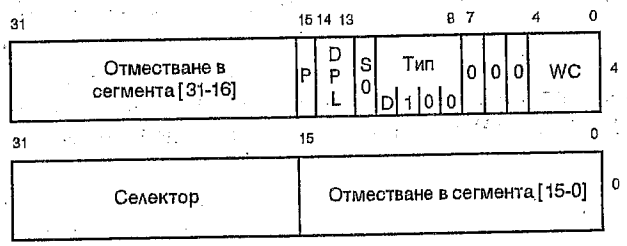
### 1.2.2. Косвено предаване на управлението

Косвеното предаване на управление се реализира с помощта на специални системни обекти, наречени шлюзове (gates). Шлюзовете, както и сегментите, се описват с дескриптори. Различават се четири типа шлюза:

- шлюз на извикване;
- шлюз към прекъсване;
- шлюз към изключение;
- шлюз към задача.

Шлюзът на извикване идентифицира разрешена входна точка в процедура от по-привилегирован код, към която може да бъде предадено управлението. Той е единственото средство за смяна на нивото на привилегия. Дескрипторът на шлюза към извикване (фиг.7.1) може да се включва в GDT или LDT и е достъпен само чрез инструкцията *call*. Операнд в инструкцията *call* е далечен указател, чийто селектор задава дескриптора на шлюза. Дескрипторът определя пълния указател за предаване на управлението - *селектор:отместване*, като чрез селектора се определя нужният сегмент, а чрез отместването - точката, в която се предава управлението. Полето DPL определя минималното ниво на привилегия, което трябва да притежава извикващата програма, а полето WC задава броя двойни думи, които автоматично се копират от текущия стек в стека на новото привилегировано ниво. Полето WC е предназначено за предаване на параметри към извикваната процедура. Правилото, разрешаващо достъпа през шлюза, има вид:

***DPL на целевия сегмент <= max(RPL,CPL) <= DPL на шлюза***




Фиг.7.1. Дескриптор на шлюз към извикване

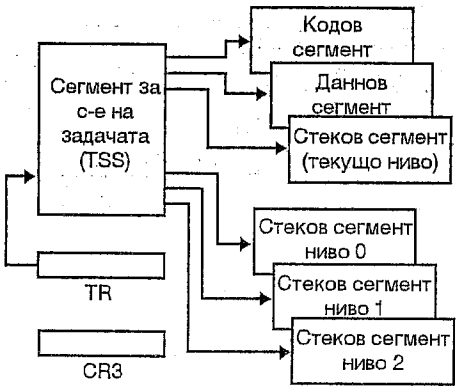
### 2. Структура на задача

Механизмът за многозадачна работа е въведен от I286 и има за цел да осигури ефективна апаратна среда за реализация на многозадачни операционни системи в техните основни функции за управление, синхронизация и комуникация на задачите (процесите) и разпределяне и защита на ресурсите. Мултизадачната среда се осигурява само при работа на МП в защитен режим. МП не използва специални инструкции за управление на задачите. Вместо това нормалните инструкции за предаване на управлението се изпълняват по различен начин, ако техните операнди са специални структури от данни. Структурите от данни за реализация на мултизадачност са: сегмент за

състояние на задачата (TSS - Task State Segment); дескриптор на TSS; дескриптор на шлюз към задача; селектор към дескриптор на TSS; регистър на задачата (TR - Task Register), задаващ селектора на TSS за текущата задача; флагът за вложена задача NT в регистъра EFLAGS.

Задачата е единица работа, която микропроцесорът може да планира, изпълни и преустанови. Основните компоненти в структурата на задачата са област за изпълнение и област за съхранение на състоянието - TSS (фиг.7.2). Областта за изпълнение се състои от кодов сегмент, стеков сегмент и един или няколко сегмента данни. Тя може да съдържа също и отделни стекове за всяко ниво на привилегия. Сегментът за състояние на задачата задава сегментите на областта за изпълнение, съхранява информация за състоянието (контекста) на задачата и осигурява механизъм за свързване на задачите. Задачата се идентифицира чрез селектора на своя TSS. Когато задача се активира за изпълнение в МП, селекторът на нейния TSS се зарежда в регистъра TR, а дескрипторът на TSS - в неговия кеш-регистър.

Задача може да бъде извиквана за изпълнение директно чрез селектор към нейния TSS или косвено посредством шлюз на задача, съдържащ селектор към TSS. При превключване на задача се извършва автоматична смяна на контекста на МП. Контекстът на текущо изпълняваната задача се съхранява в нейния TSS и изпълнението ѝ се прекратява. В регистрите на процесора се зарежда контекстът на извикваната задача и изпълнението продължава от инструкцията, зададена с новозаредения EIP.




Фиг. 7.2. Структура на задача

### 2.1. Сегмент за състояние на задачата

Цялата информация, необходима на МП да управлява една задача, се намира в сегмента за състояние на задачата TSS. Форматът на TSS е показан на фиг.7.3. Информацията, съхранявана в TSS се групира в две множества от полета - динамично и статично. Информацията от динамичното множество се обновява при всяко превключване на задачата и се състои от:

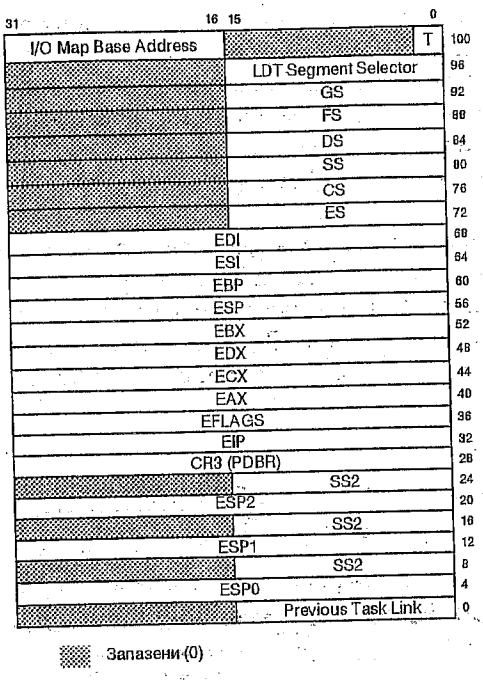
- регистрите с общо предназначение EAX, ECX, EDX, EBX, ESP, EBP, ESI, EDI;
- сегментните регистри ES, CS, SS, DS, FS, GS;
- регистъра на флаговете EFLAGS;
- указателя на инструкциите EIP;
- поле за обратна връзка (Previous Task Link), съдържащо селектора на предходната задача. Използва се при вложено извикване на задачи за автоматично връщане към предходната задача.

Микропроцесорът може да чете информацията от статичното множество, но не може да я променя. Статичното множество включва:

- селектора на LDT на задачата;
- регистъра PDBR, съдържащ базовия адрес на каталога на страниците за задачата. Използва се само при разрешено странициране;
- указатели към стековете на привилегировани нива 0-2 - SS0:ESP0, SS1:ESP1, SS2:ESP2;
- указателя към двоичната карта за вход/изход (I/O Map Base Address), използвана за управление на достъпа на задачата към конкретни адреси от входно-изходното адресно пространство;
- флаг T (Trap) - флаг за тестване. При T=1 МП генерира изключение при всяко активиране на задачата;

• информация, определяна от ОС.

TSS може да бъде разположен в произволно място в паметта. Минималният размер на TSS е 103 (67h) байта.



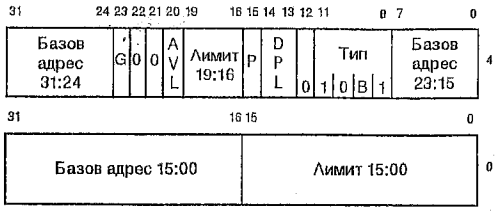

Фиг. 7.3. Формат на TSS

### 2.2. Дескриптор на TSS

Дескрипторите на TSS могат да се разполагат само в дескрипторната таблица GDT. Форматът им е аналогичен на формата на обикновения сегментен дескриптор и отделните полета се интерпретират по същия начин (фиг.7.4). Различават се от обикновените дескриптори по значението на бита S (S=0 - системен дескриптор) и полето за тип на сегмента. Битът B в него се използва за индикация на заета задача, като код 9 в полето за тип (флаг B=0) индицира незаета (свободна) задача, а код 11 (флаг B=1) означава, че задачата или е в процес на изпълнение или е осъществила вложено обръщение към друга задача. Битът B се използва за защита от повторно обръщение към заета задача, като в този случай МП генерира изключение. Всяко изменение на флага B се извършва при заключена шина (активен сигнал LOCK). В мултипроцесорна конфигурация заключването предотвратява два или повече процесора да активират едновременно една и съща задача.

Полето DPL на дескриптора на TSS определя привилегията на задачата. Всяка програма или процедура с текуща привилегия CPL има право на достъп до дадена задача и може да я активира с инструкциите ***call*** или ***jmp*** само, ако е с по-голямо или равно ниво на привилегия, т.е. ако числено е спазено неравенството **CPL <= DPL**.

Програмни запис или четене от сегмент за състояние на задача са забранени. При опит за зареждане на селектор към дескриптор на TSS в кой да е от сегментните регистри се предизвиква изключение. Информацията в TSS може обаче програмно да бъде променяна посредством синонимен данков сегмент. Дадена задача може да има само един дескриптор на TSS.




Фиг. 7.4. Дескриптор на TSS

### 2.3. Регистър на задача

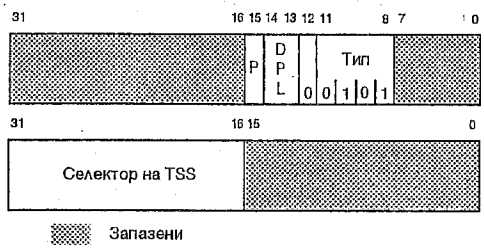
Регистърът на задача TR идентифицира текущо изпълняваната задача, като съдържа във видимата си част селектора, а в програмно невидим кеш регистър - дескриптора на нейния TSS. Видимата част на TR може да се чете и променя програмно чрез инструкциите *str* (store task register) и *ltr* (load task register):

`str r16/m16` записва видимата част на TR в регистър с общо предназначение или в дума от паметта;

`ltr r16/m16` зарежда TR с операнд селектор, по който от GDT се избира дескриптор на TSS и се зарежда в кеш регистъра. Инструкцията не предизвиква превключване на задача. Тя е привилегирована инструкция, изпълнява се само от ниво на привилегия 0 и основно се използва при инициализация на системата за задаване на първоначалната стойност на TR. След това съдържанието на TR се променя автоматично при всяко превключване на задача.

### 2.4. Дескриптор на шлюз към задача

Дескрипторът на шлюз към задача предоставя косвен, защитен достъп до нейния TSS. Той съдържа селектор към дескриптор на TSS (фиг.7.5). Основното му предназначение е да разреши обръщение към дадената задача от по-ниско привилегировани задачи посредством полето DPL. Програма или процедура с текущо ниво на привилегия CPL може да превключи дадена задача, ако е изпълнено неравенството **max(CPL, RPL) <= DPL**, т.е. ако CPL и RPL (заявено ниво на привилегия в селектора към шлюза) са числено по-малки или равни на стойността на полето DPL на шлюза. В този случай полето DPL в дескриптора на TSS, както и полето RPL на съдържащия се в него селектор, не се използват.




Фиг. 7.5. Шлюз към задача

Шлюзовете към задачи предоставят допълнително следните възможности:

• дадена задача може да има само един дескриптор на TSS, но може да се избира през няколко шлюза;

• предоставяне на селективен достъп до задачите. За разлика от дескриптора на TSS, шлюзовете към задачи могат да се разполагат в LDT. Програма или процедура, която няма привилегия за директен достъп до дескриптор на TSS на дадена задача, може да получи достъп до задачата чрез шлюз с по-голяма стойност на DPL. Това дава по-голяма гъвкавост на ОС при определяне на достъпа до определени задачи;

• прекъсване или изключение може да предизвика превключване, обръщайки се към шлюз към задача в IDT. По този начин всички задачи могат да бъдат изолирани от задачите, обработващи прекъсванията.

### 3. Превключване на задачи

Превключването на задача може да се инициира посредством един от следните способи:

- изпълнение от текущата задача на инструкциите *call* или *jmp* към дескриптор на TSS;
- изпълнение от текущата задача на инструкциите *call* или *jmp* към шлюз към задача;
- възникване на прекъсване или изключение, сочещо към шлюз към задача в IDT;
- изпълнение на инструкцията *iret* при вдигнат флаг NT в регистъра EFLAGS.

Когато дадена задача е стартирана с инструкцията *call*, тя се смята вложена в извикващата я задача. Ситуацията е подобна на извикване на подпрограма или обработка на прекъсване. Флагът NT в регистъра EFLAGS е индикатор, че текущата задача е вложена. Когато вложената задача изпълни инструкцията *iret* при вдигнат флаг NT, МП автоматично възстановява изпълнението на предходната задача. Дълбочината на вложеност е неограничена. За влагане на задачите е необходим еквивалент на стек. Той се строи на базата на полето за обратна връзка в TSS, в което се съхранява указател (селектор) към TSS на извикващата задача.

Превключването от една задача към друга задача се състои от следните апаратни действия:

1. Проверка на привилегиите и правото на текущата задача да извика новата задача.

2. Проверка за валидност на TSS на новата задача: достъпност в паметта (P=1); валиден лимит; незаета задача (при превключване с *call*, *jmp*, прекъсване или изключение) или заета - при превключване с *iret*.

3. Актуализация на флаговете B и NT за текущата задача. При превключване с *jmp* или *iret* МП нулира флага B в дескриптора на TSS на текущата задача, обявявайки я за незаета. Флагът B остава вдигнат в останалите случаи. Ако превключването е инициирано с *iret*, МП изчиства флага NT в регистъра EFLAGS на текущата задача. В останалите случаи NT не се променя.

4. Съхраняване на състоянието на текущата задача в нейния TSS. По базовия адрес от кеш регистъра на TR микропроцесорът адресира текущия TSS и записва в него значението на регистрите, съставящи динамичното множество.

Ако всички проверки и съхранявания до този момент са успешни, МП продължава процеса на превключване на задачата. В противен случай превключването се преустановява и МП се връща в състоянието си непосредствено преди инициралата превключването инструкция, т.е. управлението остава в контекста на текущата задача.

Ако се открие грешка в следващите действия, МП завършва превключването и генерира подходящо изключение преди започване на изпълнението на новата задача. Това изключение се обслужва в контекста на новата задача.

5. Зареждане на TR със сегментния селектор и дескриптора на новия TSS, като го маркира зает (B=1).

6. Зареждане на регистрите на МП със значения от TSS на новата задача. Всички грешки, свързани със зареждането и проверката на сегментните дескриптори на тази стъпка, възникват в контекста на новата задача. За новата задача се установява ниво на привилегия, указано от полето RPL на селектора в кодовия селекторен регистър CS.

7. Актуализация на флага NT за новата задача. При инициране на превключването с *call*, прекъсване или изключение флагът NT се установява в единица, при *jmp* NT остава непроменен, а при *iret* се възстановява от съхраненото в TSS копие на EFLAGS. Флагът TS в управляващия регистър CR0 се вдига.

8. Стартиране на изпълнението на новата задача от инструкцията, указана с новото съдържание на регистъра EIP.

### 4. Взаимодействие между задачи

В многозадачна среда е необходимо управлявано взаимодействие между задачите, което се постига посредством разделяеми код и данни, т.е. общи области от паметта. Общите области са достъпни чрез припокриване на адресните пространства на задачите. Могат да бъдат реализирани следните възможности за припокриване:

- напълно изолирани адресни пространства. Всички използвани от задачите сегменти се задават чрез локалните им дескрипторни таблици;
- пълно припокриване на адресните пространства. Всички сегменти се задават или чрез GDT, откъдето са достъпни за всички задачи, или са описани в локалните таблици на всички задачи;
- частично припокриване.

Задачите имат еднакви права над общите сегменти, описани в GDT. Различните задачи могат да имат различни права над общите сегменти, описани чрез синонимни дескриптори в отделните LDT.

### 5. Примерна програма за превключване на задачи

В Приложение 2 е приведен листинг на програма, разширяваща програмата от Приложение 1 с мултизадачна среда след превключване в защитен режим. За организиране на мултизадачна среда трябва да се добавят необходимите структури от данни, осигуряващи превключването на задачите, съхраняване и възстановяване на техния контекст. Към структурите от данни, дефинирани в програмата от Приложение 1, се добавят сегменти за състояние на всяка задача и техните дескриптори. В асемблерска програма TSS може да се дефинира чрез следната структура, съгласно формата от фиг.7.3:

```
struc    task_state_segment
  back      dw  ?     ;селектор към TSS на предходната задача
  back_h    dw  ?
  esp0      dd  ?     ; размер на стека за DPL 0
  ss0       dw  ?     ; индекс на дескриптора на стека в GDT
  ss0_h     dw  ?
  esp1      dd  ?     ; размер на стека за DPL 1
  ss1       dw  ?     ; индекс на дескриптора на стека в GDT
  ss1_h     dw  ?
  esp2      dd  ?     ; размер на стека за DPL 2
  ss2       dw  ?     ; индекс на дескриптора на стека в GDT
  ss2_h     dw  ?
  tcr3      dd  ?     ; базов адрес на каталога на страниците
  teip      dd  ?     ; указател на инструкциите
  teflags   dd  ?     ; флагов регистър
  teax      dd  ?     ; регистри на задачата
  tecx      dd  ?
  tedx      dd  ?
  tebx      dd  ?
  tesp      dd  ?     ; размер на стека
  tebp      dd  ?
  tesi      dd  ?
  tedi      dd  ?
  tes       dw  ?     ; селектор на ES
  tes_h     dw  ?
  tcs       dw  ?     ; селектор на CS
  tcs_h     dw  ?
  tss       dw  ?     ; селектор на SS
  tss_h     dw  ?
  tds       dw  ?     ; селектор на DS
  tds_h     dw  ?
  tfs       dw  ?     ; селектор на FS
  tfs_h     dw  ?
  tgs       dw  ?     ; селектор на GS
  tgs_h     dw  ?
  tldt      dw  ?     ; указател на LDT към задачата
  tldt_h    dw  ?
  trap      dw  ?
  iomap     dw  ?     ; карта на I/O
ends  task_state_segment
```

Ако трябва да се установи мултипрограмен режим или възможност за превключване на нивата на привилегия, инициализиращата програма трябва да зареди поне един TSS и съответстващия му дескриптор. Дескрипторите на TSS се помещават в GDT и първоначално не трябва да са в състояние "зает", т.е флагът B се установява в 0. След превключване на МП в защитен режим се използва инструкцията ***ltr*** за зареждане в TR на селектора на TSS на текущата задача (в примера ред 331). Тази инструкция маркира TSS като зает, но не осъществява превключване на задачи. Селекторът на TSS трябва да бъде зареден, преди програмата да изпълни първото превключване на задача, тъй като при превключване контекстът на текущата задача трябва да бъде копиран в TSS. МП може вече да използва TSS, за да определи местоположението в паметта на стековете за привилегировани нива 0, 1 и 2 (стековите сегменти за привилегированите нива и указателите към тях се съдържат в TSS).

Следващи операции с TR ще предизвикват превключване на задачи. TSS и техните дескриптори могат да бъдат динамично създавани и разполагани в паметта при възникване на необходимост от това.

В приведения пример са дефинирани две задачи: начална задача, в контекста на която МП попада веднага след превключване в защитен режим (за краткост в коментарите в листинга отбелязана като "стара задача") и задача, която вложено се извиква за изпълнение (за краткост в листинга отбелязана като "нова задача"). Целта на примера е проста демонстрация на превключване на задача и връщане от нея. Двете задачи се изпълняват на едно и също привилегировано ниво - 0. Тъй като не се извършва превключване между нивата на привилегия, е дефиниран стек само за това ниво (редове 482-488). Стекът за това ниво е дефиниран в отделен сегмент, за да бъде достъпен за всички задачи от това ниво, с дескриптор stack00_descriptor, включен в GDT (ред 290) и селектор 60h (ред 306). Селекторът и началното значение на стековия указател се записват в TSS на задачите при тяхното създаване (редове 497-498 и редове 563-564).

Работната област на началната задача се състои от сегментите code32 и data32 и техните дескриптори code32_descriptor и data32_deskriptor, определени, както в програмата от Приложение 2. Нейният TSS сегмент е дефиниран в редове 494-537, а дескрипторът му oldtask_descriptor (ред 292) е поместен в GDT и е достъпен чрез селектор 40h (ред 302). В него трябва да бъдат зададени само значенията на компонентите от статичното множество, тъй като динамичното множество (значението на регистрите на процесора) се формира от програмата при превключването в защитен режим. Текущото значение на регистрите ще се запише и съхрани в TSS при превключване на задачата. От статичното множество се записват селекторът и началното значение на стековия указател за стековете на привилегированите нива (в дадения случай само за привилегировано ниво 0).

Кодът и данните на извикваната задача са разположени в сегмент (редове 419-479), достъпен посредством два синонимни дескриптора: за код ecode32_descriptor (ред 278) със селектор 30h (ред 300) и за данни edata32_descriptor (ред 280) със селектор 38h (ред 301). TSS на извикваната задача е дефиниран с редове 557-603, а дескрипторът му newtask_descriptor (ред 284) е поместен в GDT и е достъпен чрез селектор 48h (ред 303). Тъй като задачата не е активна при превключване в защитен режим, а се извиква за изпълнение, то в TSS първоначално трябва да бъде зададено подходящо значение за някои компоненти на динамичното множество. Като минимум това трябва да бъдат координатите в паметта на точката за стартиране или възобновяване на изпълнението на задачата, т.е. значението на CS (ред 589) и EIP (ред 573). За конкретната задача това е адресът на етикета estart32 (ред 437).

**Въпроси и задачи:**

1. Разучете програмата от Приложение 2.

2. Изследвайте защитните механизми, действащи по отношение на сегмента за състояние на задача. В кои случаи се генерира прекъсване и какъв е неговият тип? За целта променете различни компоненти в дескриптора на TSS на извикваната задача, като лимит, тип и др., асемблирайте и изпълнете програмата.

3. Добавете в програмата трета задача, която да извежда характерно съобщение на екрана при своето извикване. За целта определете, разработете и добавете в нужните места в програмата необходимите структури. Осъществете вложено извикване на трите задачи.

4. Изследвайте защитните механизми на МП по привилегия. Променете нивото на привилегия на една от извикваните задачи. Каква е реакцията на процесора? Осъществете обръщение към указаната задача чрез шлюз към задача. Възможно ли е това? Обосновете и проверете правилността на отговора си.

# 8. Организация на шината на микропроцесора

Цел на упражнението е разглеждане на основните принципи на организация на взаимодействието на процесора с останалите компоненти на микропроцесорната система. Разглеждат се както общите за микропроцесорите на фамилията Intel характеристики на шината, така и подходите за увеличаване на нейната производителност в едно- и мултипроцесорни конфигурации, отговарящи на развитието на архитектурата. Като база е избрана шината на I486 и Pentium.

## 1. Шинен интерфейс

Микропроцесорите се съединяват с останалите компоненти на системата (памет и периферни устройства) чрез набор сигнални линии, наречен шина. Отделните компоненти се подсъединяват към шината посредством съгласуваща логика, наречена интерфейс. В шината се отделят групи сигнали по функционален признак - адреси и данни, управление на шината и арбитраж, определение на циклите на шината, управление на кеш паметта и регистриране на грешките.

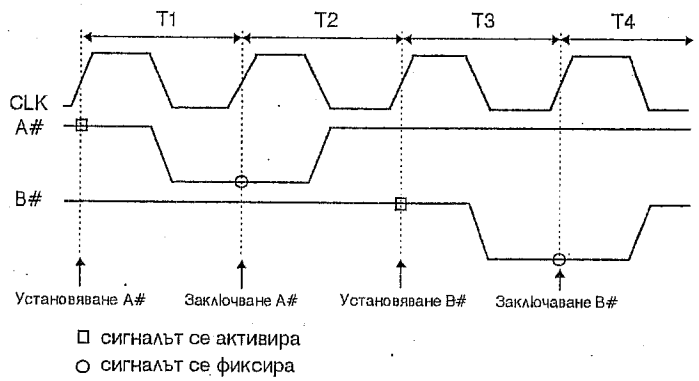
## 2. Функционални групи сигнали при i486 и Pentium

### 2.1. Особености на шината

Сред особеностите на шината на i486 и Pentium следва да се отбележат:

- немултиплексирани шини за адрес (32 разреда) и данни (32 разреда за i486 и 32/64 разреда за Pentium);
- еднотактова синхронизация (1x). Всички операции на шината са привързани към нарастващия фронт на тактовия сигнал CLK. Асинхронни по отношение на CLK са само сигналите за немаскируемо прекъсване NMI и игнориране на грешката от устройството с плаваща запетая IGNNE#;
- пакетно предаване (до 16 байта);
- кеширано предаване;
- блокиране и псевдоблокиране на шината;
- съгласуване на вътрешната и външна кеш памет;
- ограничаване на адресното пространство до 1 MB за съвместимост с микропроцесора 8086;
- управление размера на шина данни с цел поддържане на 8/16 разредни периферни устройства;
- контрол на данните по четност.

Шината изисква всеки входен сигнал да се тества през валиден интервал на нарастващия фронт на тактовия импулс CLK и неговият ефект да се появи не по-рано от следващия фронт на CLK (фиг.8.1). Това предполага един пълен такт за комуникация между подателя и получателя на сигнала и най-малко един пълен такт за подготовка на отговора. На фиг.8.1 сигналът A# се активира в такт T1, но не се възприема до T2. Приемащата схема използва такта T2, за да определи отговора си и активира сигнала B# в T3. Останалите компоненти към шината възприемат B# като активен в T4. Символът "#" след името на сигнал означава, че той е активен с ниското си ниво.




Фиг. 8.1 Пример за шинен протокол

### 2.2. Цикли на шината

Взаимодействието на процесора с външните компоненти на системата се реализира посредством цикли на шината. Циклите на шината образуват две основни групи: предаване на данни и специални цикли. Към циклите за предаване на данни спадат: предварително извличане на инструкции; четене на данни от паметта или от входно-изходни устройства; запис на данни в паметта или във входно-изходните устройства. Към специалните цикли се отнасят: потвърждаване на прекъсване; спиране; изчистване на кеш паметта; отключване. Типът на цикъла се задава посредством сигналите M/IO#, D/C#, W/R#.

Циклите за предаване на данни са два типа - пакетни и непакетни. В непакетните цикли се предават до четири (осем при Pentium) байта с максимална скорост два такта на един елемент данни (байт, дума, двойна дума или четворна дума). Когато се предава един елемент данни, цикълът се нарича единичен. Последователно повтарящи се единични цикли образуват многоциклова последователност.

Пакетните цикли осигуряват максимална скорост за предаване. В пакетен цикъл се предават до 16 байта (32 за Pentium) със скорост един елемент за такт. Те са въведени с цел кеширано четене, при което се осъществява запис на един ред във вътрешната кеш памет, но се използват и за четене на числа с плаваща запетая, четене на дескриптори от дескрипторните таблици и др. От непакетните цикли могат да се кешират само многоциклови последователности.

### 2.3. Сигнали на микропроцесорите I486 и Pentium

#### 2.3.1. Шини за адрес и данни

Адресната шина се състои от сигналите за адрес A2-A31, като A4-A31 са двупосочни. Като входове A4-A31 се използват за определяне на адресите на вътрешната кеш памет, когато се обявяват за недостоверни от страна на външните схеми. От младшите разреди на адреса A0-A1 се формират сигналите BE0#-BE3#, които се използват за определяне на валидните байтове на данновата шина. Входният сигнал A20M# осигурява затваряне на адресното пространство на 1MB.

По двупосочната шина данни може да се предава двойна дума (четворна за Pentium), като младшият байт се предава по линиите D0-D7, а старшият по линиите D24-D31 (D56-D63 за Pentium). Всеки байт данни се придружава с контролен бит по четност. Контролните битове се предават по двупосочните линии DP0-DP3 (DP0-DP7 за Pentium) съответно. Изходният сигнал PCHK# регистрира наличието на грешка по четност в един или няколко байта. Размерът на шина данни може да се управлява и от външните схеми чрез сигналите BS8# и BS16#. Тези сигнали установяват размера на шината за текущия цикъл съответно 8 или 16 разреда и се използват при работа с 8/16 разредни устройства. Те оказват влияние на броя цикли на шината, които трябва да се изпълнят, за да се предаде елемент данни с необходимия размер.

#### 2.3.2. Сигнали за определяне на цикъла и управление

Сигналите за определяне на цикъла и управление определят типа и посоката на предаване на данните, моментите от време, когато данните стават достоверни или недостоверни и възможността за кеширане.

Началото на всеки цикъл се отбелязва с активиране на сигнала ADS#, показващ, че на шината на процесора се намират валидни (достоверни) сигнали за адрес и тип на цикъла. При непакетните цикли всеки адрес се отбелязва с активиране на ADS# и единично предаване на данни. В пакетните цикли с ADS# се отбелязва само началото на последователност от адреси и съответстващи предавания на данни.

Типът на цикъла на шината се определя от двоичната комбинация на сигналите M/IO#, D/C# и W/R#. Сигналът M/IO# (памет/вход-изход) разграничава адресното пространство на паметта от входно-изходното адресно пространство. Сигналът D/C# (данни/управление) разграничава циклите по предаване на данни от останалите цикли, а W/R# (запис/четене) определя посоката на предаване на данните. Таблица 8.1. съдържа валидните двоични комбинации на тези сигнали и съответстващите типове цикли.

Завършването на текущия цикъл за предаване на данни се осъществява при получаване на входния сигнал RDY# (готовност). С този сигнал външните схеми (памет или входно-изходни устройства)

показват, че са поставили достоверни данни върху шина данни или че са възприели данните от процесора. Този сигнал се възприема в края на втория такт на цикъла. Бавните устройства могат да формират сигнала RDY# след въвеждане на състояния (тактове) на изчакване до стабилизация на данните. Сигналът RDY# винаги завършва текущия цикъл на шината, даже ако е издаден по средата на пакетен цикъл.

В пакетните предавания се използва сигналът BRDY# (пакетна готовност). Той е аналогичен на сигнала RDY#, но не прекратява текущия пакетен цикъл. Процесорът реагира на BRDY#, очаквайки в следващия такт ново предаване на данни. Издаването на сигнала BLAST# прекратява пакета. Сигналът BLAST# (последен в пакета) показва последното предаване във всеки цикъл (пакетен или непакетен) от страната на процесора. Когато е издаден BLAST# подаденият към процесора BRDY# действа като RDY#. Ако BLAST# не е активиран, то за завършване на цикъла е необходимо допълнително предаване.

Входният сигнал KEN# (разрешение на кеш) разрешава вътрешната кеш памет. Ако този сигнал е подаден, процесорът ще изпълни толкова последователни цикли, колкото са необходими за запълване на 16 байтовия ред на кеш паметта.

**Таблица 8.1**

| M/IO | D/C# | W/R# | Вид на предаването |
|---|---|---|---|
| 0 | 0 | 0 | Потвърждение на прекъсването |
| 0 | 0 | 1 | Специални цикли на шината |
| 0 | 1 | 0 | Четене на данни от входа/изхода |
| 0 | 1 | 1 | Запис на данни към входа/изхода |
| 1 | 0 | 0 | Превантелно извличане на инструкции от паметта |
| 1 | 0 | 1 | (резервиран) |
| 1 | 1 | 0 | Четене на данни от паметта |
| 1 | 1 | 1 | Запис за данни в паметта |

### 2.3.3. Управление на шината

Сигналите за управление на шината, цикълът за потвърждаване на прекъсване и специалните цикли на шината влияят на временната диаграма на предаване и защитните действия на шината на процесора. Част от тях се използват във всички системи, а други - в системи с няколко водещи на шината. Сигналите, които се използват във всички системи са: RESET, привеждащ системата в начално състояние; INTR - сигнал за маскируемо прекъсване, NMI - сигнал за немаскируемо прекъсване.

Сигнали, използвани в системи с няколко водещи на шината:

BREQ (заявка за шината) - изходен сигнал, с който процесорът иска достъп до шината или индицира, че в дадения момент я използва. Този сигнал може да се подаде на външна арбитрираща схема, която да управлява достъпа до шината между няколко водещи (например в мултипроцесорна система).

HOLD - входен сигнал, заставящ процесора да освободи шината. В отговор процесорът след завършване на текущия цикъл привежда болшинството от сигналите си в плаващо състояние и издава подтвърждаващия сигнал AHOLD.

AHOLD - разрешение за заемане на шината в отговор на HOLD.

LOCK - позволява на процесора да да изпълни няколко цикъла на шината без прекъсване от сигнала HOLD. Сигналът LOCK се формира в операциите четене-модификация-запис, в циклите за потвърждаване на прекъсването и зареждане на дескрипторите на сегментите.

PLOCK - аналогично на LOCK позволява на процесора да изпълни няколко шинни цикъла без

прекъсване от HOLD. Той се издава във всички многоциклови последователности, в които предаваните данни са изравнени на границата на четворни думи (предаване на 64 разредни операнди с плаваща запетая и запълване на ред в кеш паметта).

BOFF# - входен сигнал за бързо преотстъпване на шината. Показва, че на друг водещ е нужно да завърши цикъл на шината преди завършване на текущия цикъл на процесора. Той се използва за предотвратяване на взаимна блокировка, при която нито процесорът, нито другият водещ, могат да завършат своите операции, тъй като всеки очаква някакво действие от другия. Реакцията на BOFF# е аналогична на HOLD, но по-бърза, защото не се изчаква завършване на цикъла, а шината се освобождава в следващия такт и потвърждение не се издава. След снемане на BOFF# процесорът надеждно осъществява рестартиране на цикъла, който е прекъснал.

### 2.3.4. Управление на кеш паметта

Управлението на кеш паметта осигурява съгласуване между вътрешната кеш памет, основната памет и външната кеш памет. Данните при четене могат да се кешират (запишат във вътрешната кеш памет) при активен входен сигнал KEN#. Всяка 4K страница в основната памет може да бъде определена като кешируема или не, и да й се зададе принципа на кеширане с обратен или едновременен запис. Сигналът PCD показва дали текущата страница е кешируема, а сигналът PWT определя принципа на кеширане.

Входният сигнал AHOLD (адресен HOLD) заставя процесора да приведе в следващия такт адресната си шина в плаващо състояние. Това позволява на външна схема да предаде адрес в процесора за обявяване за недостоверен ред във вътрешната кеш памет. Адресът се придружава от входния сигнал EADS# (достоверен външен адрес).

Входният сигнал FLUSH# (изчистване на вътрешната кеш памет) заставя процесора да изчисти съдържанието на цялата кеш памет.

В системата инструкции са включени две инструкции за работа с кеш паметта:

`invd` - стартира цикъл изчистване на кеш паметта. В този цикъл се изпълняват две действия: превръща в недействително съдържанието на вътрешната кеш и уведомява външната кеш да обяви за недостоверно цялото си съдържание;

`wbinvd` - обявява като недостоверно съдържанието на вътрешната кеш памет, заставя външната кеш памет да запише цялото си съдържание в основната памет, след което да го обяви за невалидно.

### 2.3.5. Управление на грешките с плаваща запетая

Механизмът за управление на грешките с плаваща запетая се използва за съвместимост с процесора 8086 и операционната система DOS.

Входният сигнал FERR# (грешка от плаваща запетая) показва, че е възникнала немаскируема грешка при действие с плаваща запетая и е аналогичен на сигнала ERROR# на копроцесорите 80287 и 80387. За съвместимост с DOS този сигнал трябва да се подаде на входа за немаскируемо прекъсване NMI. Входният сигнал IGNNE# служи за игнориране на грешките от плаваща запетая.

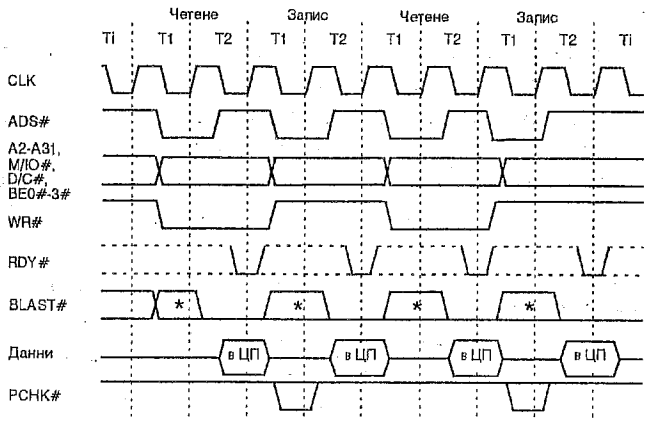
## 3. Цикли за запис и четене

Циклите на шината управляват предаванията на данните чрез последователна смяна на сигналите на шината. Началото на циклите на шината се отбелязва с подаване на сигнала за състояние на адреса ADS#. Минималният цикъл за единично предаване (четене или запис) заема два такта и се нарича цикъл "2-2". Ако външната схема не може да реагира в течение на втория такт, то се въвежда такт на изчакване и цикълът се превръща в цикъл "3-3" и т.н.

### 3.1. Единични цикли

На фиг.8.2 е показана времедиаграмата при непакетирани цикли "2-2". Сигналите, чието обозначение завършва със знака "#", са активни при ниско ниво. Знакът "*" означава недостоверен сигнал, а пунктирните линии показват, че значението на сигнала е безразлично. Цикълът започва с появата на стабилен адрес и сигналите за определяне на цикъла (A2-A31, M/IO#, D/C#, BE0#-BE3#). Активирането от процесора на сигнала ADS# показва, че тези сигнали са стабилни. Първият такт се използва само за предаване на адреса и определяне типа на цикъла. В случай на цикъл за четене

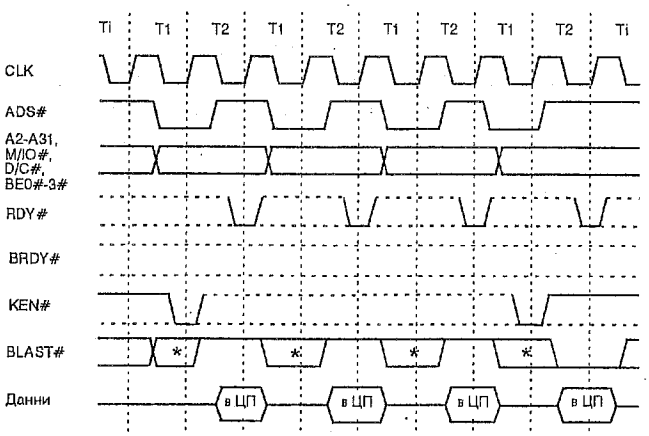
външната схема дешифрира адреса и подготвя данните. През втория такт ADS# се снема, но адресът и другите сигнали за определяне на цикъла остават стабилни. За завършване на предаването външната схема трябва да предаде сигнала за готовност RDY# преди края на втория такт. Процесорът проверява този сигнал по нарастващия фронт на CLK в третия такт. Сигналът RDY# показва, че външната схема може да приеме данните при запис или да предостави данните при четене. Освен това RDY# разрешава подаването на следващ сигнал ADS#, ако са необходими още предавания на данни. Ако външната схема не може да отреагира навреме, тя не трябва да издава RDY# до формиране на правилната реакция. МП издава сигнала BLAST# в последния (втори) такт на цикъла. МП издава този сигнал във всички цикли на шината, показвайки кога очаква завършване на предаването. В циклите за четене се осъществява контрол по четност на разрешените байтове. Изходният сигнал PCHK# показва грешката от контрола за предидущия такт. Той е достоверен един такт след RDY#.




Фиг. 8.2. Непакетен цикъл

### 3.2. Многоциклови последователности

На фиг.8.3. е показана времедиаграма на многоциклова последователност. Многоцикловата последователност представлява серия единични предавания. Процесорът например може да инициира многоциклова последователност при четене на 128 разредни инструкции, 64 разредни операнди с плаваща запетая или неизравнена двойна дума. Външната схема може да предизвика такава последователност, когато чете или записва 32 разредни данни по 8 или 16 разредна шина данни.



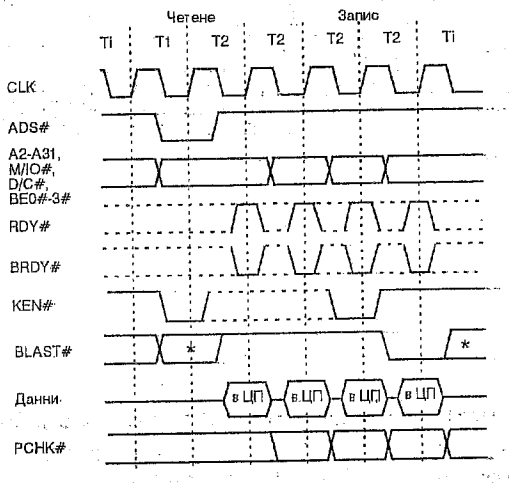

Фиг. 8.3. Кеширана многоциклова последователност

Многоцикловите последователности за четене могат да се кешират за запълване на ред от вътрешната кеш памет, в случай че страницата, от която се чете, е разрешена за кеширане.

Многоцикловата последователност започва по същия начин, както едноцикловото предаване - с активиране на ADS#. След това процесорът определя многоциклова последователност, снемайки сигнала BLAST# в края на втория такт на първия цикъл на шината. Външната схема подава сигнала за готовност RDY#, завършвайки първия цикъл на шината. Всеки следващ цикъл в последователността започва с издаване на ADS# и завършва с връщане на RDY# . Последователността завършва, когато се подаде сигналът RDY# при активен BLAST#. За да бъде кеширана последователността, външната схема трябва да подаде сигнал KEN# в края на първия такт преди RDY#. Получавайки KEN#, процесорът изпълнява от 4 до 16 последователни цикъла на шината в зависимост от значението на сигналите BS16# и BS8#, за да прочете един 16 байтов ред за кеш паметта. Сигналът KEN# трябва да бъде активиран също един такт преди връщане на сигнала RDY# за последното предаване. Второто подаване на KEN# предизвиква фактическото поместване на данните във вътрешната кеш памет.

### 3.3. Пакетни цикли

Пакетните цикли са най-бързият способ за предаване на данните. В един пакет могат да се предадат до 16 байта (32 за Pentium). Най-бързият пакет изисква изисква два такта за първото предаване и по един такт за всяко следващо предаване. В пакет могат да се предават само изравнени данни от последователни адреси. Пакетните цикли могат да бъдат кешираните и некешираните. Кешираните пакетни цикли осигуряват най-бързо запълване на редовете на вътрешната кеш памет.




Фиг. 8.4. Кеширан пакетен цикъл

На фиг.8.4. е показана времедиаграмата на кеширан пакетен цикъл, който запълва кеш паметта по 32-разредна шина данни и предава 4 двойни думи. За да се осъществи кеширано пакетно предаване трябва да се удовлетворят следните условия за пакетиране и кеширане:

• входният сигнал за разрешение на кеширането KEN# трябва да бъде подаден в началото и края на запълване на реда;

* вместо сигнала RDY# трябва да се подава входния сигнал за пакетна готовност BRDY#;

• пакетират се четения от паметта и входно-изходните устройства, които изискват повече от едно предаване;

• кешират се само четения от паметта и предварително извличане на инструкции;

• 16 разредна изравнена област от паметта винаги се чете като ред на кеш паметта без значение от началния адрес в 16 разредната област;

• страницата трябва да бъде разрешена за кеширане;

• кеш паметта трябва да бъде разрешена в регистъра за състояние на машината.

Пакетът започва с подаване на адреса, сигналите за тип на цикъла и сигнала ADS#, както и в непакетните цикли. Външната схема издава сигнала KEN# в края на първия такт и го снема след началото на втория такт. В отговор на изменението на KEN# процесорът снема BLAST#, показвайки, че за завършване на цикъла са необходими допълнителни предавания. В края на втория такт външната схема поставя на шина данни първата двойна дума и заявява своята готовност като активира

BRDY# и снема RDY#. Останалите двойни думи се въвеждат без издаване на ADS# от последователни адреси, които се определят от външната схема в зависимост от първия подаден адрес. Сигналът BRDY# се тества във всеки следващ такт и ако той е издаден, данните се въвеждат в процесора. В последното предаване на цикъла процесорът винаги активира BLAST#, показвайки, че очаква края на предаването. В края на пакетния цикъл отново трябва да бъде издаден KEN# един такт преди издаването на BLAST#. Получаването на BRDY# при издаден BLAST# завършва пакетния цикъл. Ако са издадени и двата сигнала BRDY# и RDY#, BRDY# се игнорира. В този случай пакетният цикъл се отменя преждевременно и започва нов цикъл, ако за запълването на реда на кеш паметта са необходими допълнителни цикли. Контролът по четност се осъществява както в непакетните цикли. В пакетните цикли също могат да се въвеждат тактове на изчакване.

**4. Организация на шината при I80386**

За шината на I80386 са характерни следните основни отличия по сравнение с шината на I486:

- двутактова синхронизация 2x;
- не поддържа пакетно предаване;
- конвейеризация на адреса. Тя се активира чрез специален сигнал NA# и предизвиква появата на следващия адрес и данни за вида на цикъла, преди да е завършил текущия цикъл;
- поради отсъствието на вътрешна кеш памет липсват сигнали за нейното управление, както и кешируеми цикли;
- няма контрол на данните по четност;
- включени са сигнали за връзка с външния за процесора копроцесор: PEREQ (означава, че процесорът трябва да прехвърли данни от/към копроцесора), BUSY# (копроцесорът е зает с изпълнението на инструкция и не може да получи нова) и ERROR# (грешка на копроцесора).

Останалите сигнали: INTR, NMI, RESET, HOLD, HOLDA, BE0#-BE3#, W/R#, D/C#, M/IO#, LOCK#, ADS# имат аналогично предназначение, както в I486 и Pentium. Сигналът READY# е аналогичен на RDY#. Организацията на циклите при предаване на данни е подобна на единичните цикли при I486 и Pentium с указаните различия.

**5. Организация на шината при I80286**

Различията между сигналите на I80386 и I80286 са свързани с размера и начина на адресиране и със сигналите за управление на шината. Адресната шина е 24 разредна, а шина данни - 16. За определяне на валидния байт върху данновата шина се използват сигналите BHE# (валиден старши байт) и A0. Типът на цикъла се задава чрез сигналите S1#, S0#, COD/INTA# и M/IO#. S1# и S0# се използват и за задаване на началото на цикъла. Сигналите за връзка с копроцесор са с един повече - сигнал PEACK#. Сигналът PEACK# е изходен и когато е активен означава, че в момента се прехвърля необходимият операнд. Останалите сигнали: INTR, NMI, RESET, HOLD, HOLDA, LOCK#, READY#, PEREQ, BUSY#, ERROR# имат същото предназначение както в I80386.

Принципът на работа на шината е сходен с този на I80386. Началото на цикъла се отбелязва с активиране на един от сигналите S1# или S0#, които изпълняват ролята на ADS# при I80386, I486 и Pentium, като освен това допълнително спомагат за определяне типа на цикъла заедно с COD/INTA# и M/IO#. Заедно с тези сигнали в състояние T1 на шината се подава и адресът. От състояние T1 шината задължително преминава в състояние T2, в което се очаква потвърждение от адресираното устройство чрез сигнала READY#. След получаване на READY# приключва текущия цикъл и започва нов, ако има заявка от процесора или се влиза в състояние на очакване TI. Няма възможност за конвейеризация на адресите.

**6. Особености на шината при фамилия P6**

Шината на P6 е проектирана за ефективна работа в мултипроцесорна конфигурация. За тази цел тя притежава следните особености:

- поддържа синхронен протокол с фиксация на сигналите;
- симетрично разпределено арбитриране между процесорите;
- сигнали и логика (наречена слушаща - snoop) за поддържане на кохерентността на вътрешните кеш памети на процесорите;
- сигнали за привеждане в режим на ниска консумация;

• шина за свързване и предаване на съобщения между вградените в процесорите вътрешни интелигентни контролери за прекъсване;

• развити средства за контрол на предаваните адреси, данни и кодове за управление, като данните се контролират чрез код, коригиращ единични и откриващ двойни грешки;

• сигнали за диагностика и управление на производителността;

### 6.1. Предаване на данни

Предаванията по шината, наречени транзакции, се състоят от две фази: адресна фаза или заявка и даннова фаза или отговор. Началото на предаването (заявката) се определя, както и при по-младшите представители на фамилията Intel, с активиране на сигнала ADS#. Този сигнал определя валидността на сигналите REQ[4:0] (тип на предаването), A[35:3] (адрес) и AP[1:0] (контролни разреди на адреса). Младшите адресни разреди A[2:0] формират сигналите за разрешение на байт, като са възможни от 1 до 8 байтови предавания. Активира се и RP#, който контролира по четност всички управляващи сигнали на заявката - REQ[4:0]#, AP[1:0]# и ADS#.

Във фаза отговор адресираното устройство трябва да завърши предаването. При запис устройството отговаря със сигнала TRDY#, индицирайки, че е готово да приеме данните. При четене удостоверява достоверността на данните с активиране на сигнала DRDY#. DRDY# се поставя във всеки такт, в който могат да бъдат предадени валидни данни и се дезактивира, за да се въведат състояния на изчакване. При многотактови предавания, при които се предават последователно няколко елемента данни, се използва сигналът DBSY#, с който шината се задържа до завършване на предаването, т.е до последния DRDY#. Не е необходимо да се активира DBSY# при еднотактово предаване, ако не са необходими тактове на изчакване. Данните върху данновата шина D[63:0] се съпровождат с 8 разреден контролен код DEP[7:0], коригиращ единична и откриващ двойна грешка.

### 6.2. Арбитриране на шината

До три устройства (агенти) могат едновременно да се състезават (арбитрират) за системната шина при PentiumII - до два процесора (симетрични агенти) и един приоритетен агент (входно-изходна подсистема или памет). Симетричните агенти заявяват владеене на шината чрез сигналите BREQ[1:0]#, а приоритетният агент чрез сигнала BPRI#. Владеенето на шината е необходимо условие за започване на предаване по нея. Симетричните агенти се арбитрират за шината по ротационна приоритетна схема, което гарантира справедливост и симетричност. Схемата за арбитраж е разпределена, т.е. определянето на следващия водещ се взема едновременно във всички агенти. Решението е на базата на информацията за последния водещ, който е известен на другите агенти, и активирания BREQ[1:0]#. Издалият заявка за шината приоритетен агент е винаги следващият водещ, т.е неговият приоритет е винаги по-голям от приоритета на симетричните агенти. Арбитражът може да бъде временно преустановен чрез сигналите BNR# и LOCK#.

Сигналът BNR# може да бъде активиран от всеки агент, за да блокира изпълнение на следващи предавания по шината. Той се активира, когато някои системни ресурси, като адресни или даннови буфери, могат да останат временно заети и възпрепятствани да съхранят данни за други предавания.

Сигналът LOCK# се активира от агент, който изпълнява неделима последователност от предавания по шината, които не трябва да бъдат прекъсвани. Използват се за реализация на семафорни операции.

### 6.3. Осигуряване на кохерентност на кеш паметите

При мултипроцесорни системи съществува проблем при съвместната работа с общи данни на процесите, изпълнявани в различните процесори. Всяка промяна на общите данни трябва да бъде видима за всички процеси. Когато общите данни попаднат във вътрешните кеш памети модификацията им може да остане невидима за останалите процеси и те да работят с неактуални техни копия, т.е. нарушена е кохерентността на данните. За решаването на този проблем в процесорите към шината е включена специална логика, наречена "слушаща" логика. Тази логика следи адресите по шината и ги проверява за съвпадение във вътрешната кеш памет. При откриване на съвпадение активира или сигнала HIT# или HITM# в зависимост от това дали данните след зареждането им в кеш са модифицирани (HITM#) или не (HIT#). Ако не може веднага да се определи

състоянието на съответния ред в кеш-паметта сигналите HIT# и HITM# се активират едновременно и текущото предаване се удължава. При активиране на HITM#, агентът стартирал предаването, го преустановява. Агентът, издал HITM#, записва модифицирания ред обратно в паметта.

### 6.3. Управление на изпълнението

В процесорите от P6 е предвидена възможност за преминаване в режим на ниска консумация на мощност. За тази цел се използват сигналите STPCLK# (спиране на тактовия сигнал) и SLP# (sleep). При активиране на STPCLK# процесорът преминава в т.н. състояние Stop-Grant, като спира подаването на тактовия импулс към вътрешните си схеми, но продължава да следи предаванията по шината. След дезактивиране на STPCLK# процесорът възстановява нормалната си работа. При активиране на SLP# процесорът преминава в друг режим на ниска консумация - SLEEP, като престава да следи и предаванията по шината.

Процесорите от P6 се характеризират и с вградени интелигентни контролери за прекъсване, т.н. APIC. Между APIC контролерите се поддържа специален протокол с помощта на сигналите PICCLK (тактов сигнал за вътрешните APIC) и PICD[1:0]# (двупосочно последователно предаване на съобщения между APIC). Сигналите за прекъсване постъпват по линиите LINT[1:0] в APIC. При забранен APIC сигналът LINT0 се възприема като INTR, а LINT1 - като NMI. Двата сигнала са асинхронни.

Към групата на сигналите за управление спадат: INIT# и RESET# - сигнали за инициализация; FLUSH# - предизвиква запис на всички редове на вътрешните кеш памети обратно в паметта и обявяването им за невалидни.

### 6.4. Индикация на грешки

Сигналите BINIT# (ненадеждна работа на шината), BERR# (грешка по шината), IERR# (вътрешна грешка в процесора, нямаща отношение към операциите по шината), FRCERRR (грешка от контрола на данните), AERR# (грешка от контрола на адреса) сигнализират за различни неправилни ситуации по шината.

Сигналът AERR# се активира при откриване на грешка по четност на адреса от някой от агентите на шината. В отговор на AERR# предаването се преустановява. Заявяващият агент може да се опита да го повтори по-късно лимитиран брой пъти, след което се установява твърда грешка. Сигналът AERR# може да бъде забранен при конфигуриране. В този случай грешките по четност на адреса се игнорират. Сигналът BINIT# се използва за сигнализиране на условия, които препятстват надеждността на операциите по шината.

**Въпроси и задачи:**

1. Кои са общите характеристики на шината за разгледаните микропроцесори?
2. Наблюдавайте с логически анализатор времедиаграмите на шината на 80286 при изпълнение на последователност инструкции с различни методи на адресация. По какво се различават циклите за четене на инструкции и четене на операнди? Какво влияние оказват различните методи на адресация върху последователността от цикли върху шината?
3. Сравнете получените от т.2 времедиаграми с времедиаграмите за изпълнение на същата последователност от инструкции за I486. В какво се изразява разликата?
4. Проектирайте структурната схема и алгоритъма на работа на интерфейса на контролер за динамична памет, подключван към шината I486. Контролерът трябва да реализира едноциклово, многоциклово и пакетно предаване на данни.

# 9. Обработка на прекъсвания и изключения. Контролер за прекъсване

Целта на упражнението е изучаване на особеностите на системата за възприемане и обработка на прекъсвания и изключения в фамилията микропроцесори на Intel.

## 1. Видове прекъсвания и изключения

Реакцията на процесора на външни и вътрешни събития се реализира посредством механизма на прекъсване. Вътрешните събития се наричат още изключения и възникват в резултат на програмни или апаратни особености на системата. Те са синхронни по отношение изпълнението на програмата. По отношение момента на откриването им и точката на връщане в прекъснатата програма те могат да бъдат:

• грешки - откриват се преди или по време на изпълнение на инструкцията, която след обработка на изключението се рестартира при възстановено състояние на процесора;

• капани - откриват се на границата между две инструкции и възстановяването е от следващата инструкция;

• аборти (критични грешки) - предизвикват се от програмни и апаратни проблеми, при които не е възможно да се продължи изпълнението на програмата.

Външните събития предизвикват прекъсвания. Те са асинхронни по отношение на работата на процесора и се възприемат чрез входовете за прекъсване. Прекъсванията могат да бъдат маскируеми (вход INTR) или немаскируеми (вход NMI). Особен вид прекъсване е началното установяване (вход RESET).

Прекъсванията и изключенията се разпознават по техния номер (табл.9.1), който може да бъде от 0 до 255. Номерата от 0 до 31 съответстват на точно определено изключение, а номера от 32 до 255 се използват за външни маскируеми прекъсвания. Номерът се издава от прекъсващото устройство през цикъл на шината за потвърждаване на прекъсването.

**Таблица 9.1.**

| Номер | Причина | Тип | Рестарт | Код на грешка |
|---|---|---|---|---|
| 0 | Грешка при деление | Нарушение | Да | Няма |
| 1 | Тестване | Нарушение/капан | Да | Няма |
| 2 | Немаскируемо прекъсване | Капан | Да (*) | Няма |
| 3 | Контролна точка | Капан | Да (*) | Няма |
| 4 | Препълване | Капан | Да (*) | Няма |
| 5 | Нарушаване границите на масив | Нарушение | Да | Няма |
| 6 | Недействителен код на операция | Нарушение | Да | Няма |
| 7 | Устройството не е достъпно | Нарушение | Да | Няма |
| 8 | Двойно нарушение | Авария | Не | Да (0) |
| 9 | Не се използва | | | |
| 10 | Недействителен TSS | Нарушение | Да | Да |
| 11 | Неприсъствие на сегмент | Нарушение | Да | Да |
| 12 | Нарушение на стека | Нарушение | Да | Да |
| 13 | Нарушение на общата защита | Нарушение/капан | Да | Да |
| 14 | Странично нарушение | Нарушение | Да | Да |
| 15 | Резервирано | | | |
| 16 | Грешка с плаваща запетая | Нарушение | Да | Няма |
| 17 | Контрол на изравняване | Нарушение | Да | Да (0) |
| 18 - 31 | Резервирани | | | |

(*) - рестарт от следващата инструкция

Едно прекъсване може да не бъде разпознато в два случая - когато е маскирано и когато момента се обработва друго прекъсване с по-висок приоритет. Маскирането на външните прекъсвания се осъществява чрез флага IF на регистъра EFLAGS. При IF=0 прекъсванията по вхо

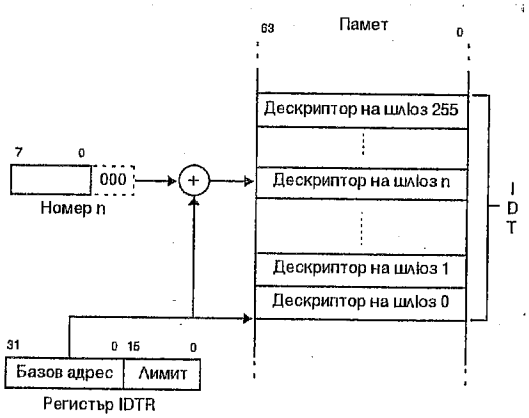
INTR# са маскирани, а при IF=1 тези прекъсвания са разрешени. Немаскируемото прекъсване не се разпознава само, когато пристигне в момент на обработка на друго немаскируемо прекъсване. Маскиране на изключение 1 (изключение при настройка) се извършва с флага RF в регистъра EFLAGS.

Едновременно възникнали изключения и прекъсвания се обработват съгласно следния приоритет в намаляващ ред: всички грешки, с изключение на номер 1; капани; изключение 1 за текущата инструкция; изключение 1 за следващата инструкция; немаскируеми прекъсвания по вход NMI; маскируеми прекъсвания по вход INTR, като по-малък номер означава по-висок приоритет. Процесорът обработва най-напред прекъсване или изключение от класа с най-висок приоритет. Изключенията с по-нисък приоритет се отхвърлят, а прекъсванията с по-нисък приоритет остават висящи.

## 2. Алгоритъм на процесора при обработка на прекъсвания и изключения

Прекъсванията и изключенията се обработват по следния начин. Първо в стека се запазва съдържанието на регистрите EFLAGS, CS и EIP. Второ, идентифицира се прекъсването, за да се получи неговият номер. Трето, чрез получения номер се индексира съответен елемент в таблицата с дескриптори на прекъсвания (IDT - Interrupt Descriptor Table). Таблицата IDT асоциира с всеки номер на прекъсване или изключение описание (дескриптор) на адреса и начина на извикване на програмата, която го обслужва. Следва изпълнение на програмата за обслужване на прекъсването или изключението, която трябва да завършва с инструкцията *iret*. При изпълнение на *iret* се възстановяват регистрите EFLAGS, CS и EIP от стека и управлението се предава в точката, където е възникнало прекъсването или изключението.

Таблицата IDT може да бъде разположена на произволно място в паметта, като местоположението и размерът ѝ се задават чрез специален регистър - IDTR. Всеки дескриптор в IDT е с размер 8 байта и номерът му съответства на номера на прекъсването или изключението, което той обслужва. Дескрипторите могат да бъдат три вида: шлюзове към задачи; шлюзове към прекъсване и шлюзове към капани (фиг.9.1). Шлюз към прекъсване или капан сочи косвено към процедура, която се изпълнява в контекста на текущата задача. Селекторът на шлюза указва дескриптор на кодов сегмент от глобалната (GDT) или локалната (LDT) дескрипторни таблици. Полето за отместване задава началото на процедурата, обработваща прекъсването или изключението. Единствената разлика между шлюз към капан и шлюз към прекъсване е, че при използване на шлюз към прекъсване автоматично се нулира флагът IF, като по този начин се маскират следващите прекъсвания. Шлюз към задача предизвиква превключване към задача за обслужване на даденото изключение или прекъсване. Обслужването на прекъсване е подчинено на общите правила за защита, характерни за защитения режим на работа на микропроцесорите Ix86.




Фиг. 9.1. Предаване на управлението за обработка на прекъсване

Определен тип изключения предизвикват също и записване на код за грешка в стека, който може да се използва за определяне на причината, предизвикала изключението. Форматът на кода на грешка е подобен на формата на селектор с изключение значението на трите младши бита: най-

младшият бит (бит 0) показва дали изключението е предизвикано от външно за програмата събитие (стойност 1), а бит 1 се установява в 1, ако индексиращата част от кода на грешката (битове 3-15) се отнася до шлюз в IDT. Ако бит 1 не е установен, тогава следващият бит 2 показва дали кодът на грешката се отнася до GDT (стойност 0) или LDT (стойност 1). Останалите 14 бита са селектор към необходимия сегмент в съответната таблица. По този начин едно прекъсване може да бъде обслужено от няколко различни програми в зависимост от кода за грешка.

Прекъсвания и изключения могат да бъдат генерирани програмно посредством инструкцията ***int n***, където ***n*** е произволен номер на прекъсване в интервала 0-255. Следва да се има пред вид, че с тази инструкция могат да се изпълнят процедурите за обработка, както на маскируеми, така и на немаскируеми прекъсвания, но без характерната за тези случаи апаратна реакция. Например прекъсванията, генерирани с инструкцията ***int n*** не могат да бъдат маскирани с флага IF. В стека не се записва и код за грешка, така че съществува опасност да не се възстанови правилната стойност на EIP след връщане от обработващата процедура.

Инструкциите за работа с флага IF (***sti*** - установяване в 1 и ***cli*** - нулиране) могат да бъдат изпълнявани само при текущо привилигировано ниво CPL по-малко или равно на значението на флаговете IOPL в EFLAGS. Неявно върху съдържанието на EFLAGS и следователно върху флага IF, въздействат инструкциите, възстановяващи EFLAGS от стека - ***popf***, ***iret***, обръщение към шлюз на прекъсване и превключване на задачи.

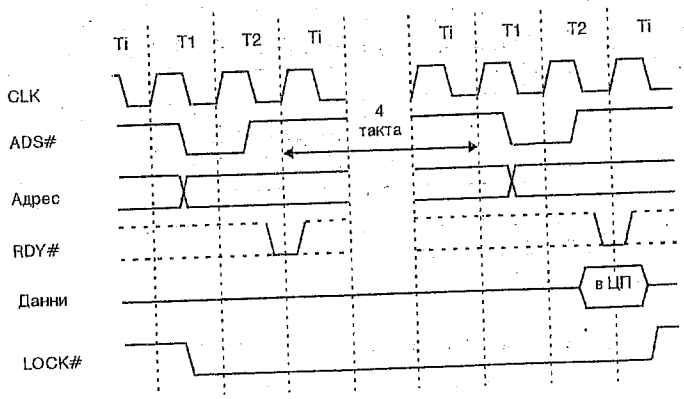
Маскиране на изключения и прекъсвания е необходимо и при превключване на стекови сегменти. Обикновено за тази цел се използват две инструкции, например:

```
mov  SS, AX          ; зареждане селектора на новия стеков сегмент
mov  ESP, StackTop   ; установяване на стековия указател на върха на стека
```

Ако се регистрира прекъсване или изключение между двете инструкции, то двете части на логическия адрес към стека (SS:ESP) няма да си съответстват за обработващата прекъсването процедура. За да се предотврати такава ситуация автоматично се забраняват прекъсвания и изключения за тестване след инструкциите, изменящи регистъра SS (***mov*** в SS и ***pop*** в SS), докато не се изпълни следващата ги инструкция. Проблемът не съществува, ако за зареждане в SS се използва инструкцията ***lss***.

### 3. Потвърждаване на външно прекъсване

Когато процесорът получи маскируемо прекъсване чрез входа си INTR, той завършва текущата инструкция и ако обслужването на това прекъсване е разрешено, стартира цикъл по шината, наречен цикъл за потвърждаване на прекъсване (фиг.9.2). Чрез този цикъл процесорът получава от прекъсващото устройство номера на прекъсването. Външното устройство, което генерира прекъсването, най-често е контролерът I8259A, но неговата роля може да се изпълнява и от специализирана за целта логика. Цикълът за потвърждаване на прекъсване е въведен за удовлетворяване на изискванията на контролера I8259A.




Фиг. 9.2. Цикъл за потвърждаване на прекъсване

Процесорът потвърждава прекъсването, получено по вход INTR, изпълнявайки два блокирани цикъла за четене, разделени с 4 празни такта. В отговор на сигнала INTR, който се проверява в началото на всяка инструкция, микропроцесорът активира сигнала ADS# заедно със сигналите M/IO#, D/C# и W/R# в комбинация 000 (тип на цикъл за потвърждаване на прекъсване). Адресните изводи с изключение на A2 и BE0# се установяват в състояние 0. Сигналът A2 е с високо ниво през първия цикъл (адрес 0..04h) и с ниско ниво през втория (адрес 0..00h). По време на първия цикъл четене се изчаква RDY#, но данните се игнорират. През втория цикъл при пристигане на RDY# от младшия байт на данновата шина (D0-D7) се приема номерът на прекъсването. Сигналът LOCK# е активен през цялото време до завършване на втория цикъл, за да осигури неделимост на двата цикъла.

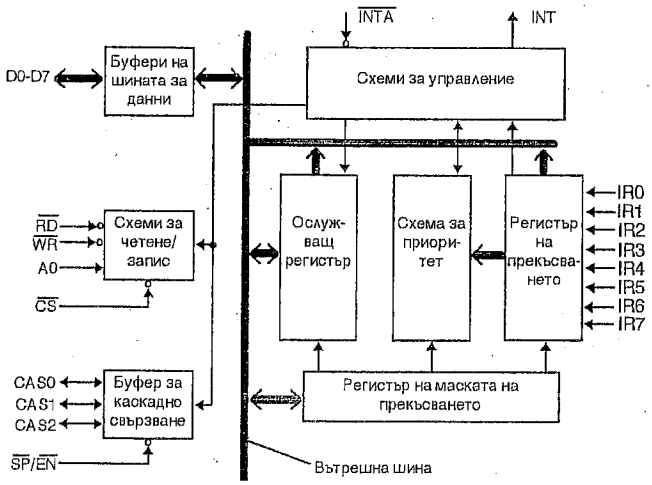
### 4. Контролер на прекъсванията

Определяне на приоритета на външните прекъсвания, както и генериране на техния номер се извършва от външна схема - контролер на прекъсванията I8259A. За повишаване на гъвкавостта за приложение в различни системи функционалните възможности на контролера се управляват програмно, а именно:

- динамична промяна на приоритета на прекъсванията;
- начин на въздействие на входните сигнали за прекъсване - по фронт или ниво;
- настройка на номерата на множеството източници на прекъсване;
- определяне на различни режими на работа - каскадно свързване, вложеност на прекъсванията.

### 4.1. Структура на контролера I8259A

Сигналите за прекъсване от външните източници постъпват асинхронно на входовете IR0 - IR7 на контролера и се запомнят в регистъра на заявките IRR (фиг. 9.3). В регистъра ISR се записват и съхраняват всички заявки, чието обслужване от МП е започнало. Регистърът на маските IMR се зарежда програмно и съхранява индивидуалните маски на заявките за прекъсване. Приоритетът за обслужване на постъпили заявки се определя от приоритетната логика на базата на съдържанието на регистрите IRR, ISR и IMR. Адресният компаратор се използва при каскадно свързване на няколко контролера, осигуряващи по-голям брой входни линии за прекъсване. Един от контролерите се дефинира като главен (на вход SP/EN на неговия компаратор се подава напрежение 5V), а останалите като подчинени (вход SP/EN на съответните компаратори се подключва към маса). Сигналът INT от подчинен контролер се подава на някой от входовете IR на главния контролер. Адресният компаратор на главния контролер адресира подчинените контролери като в цикъла за разпознаване на прекъсване извежда на адресните линии CAS0-CAS2 номера на подчинения контролер, чиято заявка е приета. Адресацията от страна на МП на програмно достъпните регистри на контролера се извършва чрез сигналите CS# и A0, а посоката на предаване се задава с RD# (четене) и WR# (запис). Достъпът до тези регистри се управлява от логиката за четене и запис.




Фиг. 9.3. Структура на контролера на прекъсванията I8259A.

### 4.2. Алгоритъм на работа

След като логиката за приоритет определи най-приоритетната постъпила заявка за прекъсване от регистъра IRR, се активира сигналът INT, който постъпва на входа INTR# на микропроцесора. Контролерът изчаква две последователни активирания на сигнала INTA# от страна на МП. При втория INTA# поставя на шина данни номера на прекъсването, след което вдига съответстващия на възприетото прекъсване бит в регистъра ISR, маскирайки по този начин повторна заявка от същия източник. Този бит се нулира програмно от процесора в края на обработката на прекъсването. Нова заявка се приема от контролера в случай, че не е маскирана и има по-висок приоритет от обслужваната в момента. Приоритетът на заявките за прекъсване се определя по два начина:

- статичен, при който всеки вход IR0-IR7 има твърдо зададен приоритет, като най-висок е приоритетът на заявката, постъпваща по вход IR0, а най-нисък - заявката от вход IR7;
- динамичен, при който приоритетът на входовете IR0-IR7 се мени по ротационен принцип.

### 4.3. Програмиране на контролера

Действието на контролера се програмира чрез два типа управляваща информация, предавана от микропроцесора - 8 разредни инициализиращи командни думи (ICW) и операционни командни думи (OCW), които се съхраняват в съответни вътрешни регистри. При начално установяване на контролера се предават последователно 4 инициализиращи думи ICW1 - ICW4. Чрез ICW1 се задават основните режими на работа:

- начин на приемане на заявките за прекъсване - по фронт или ниво на сигналите IR0 - IR7;
- режим на каскадно свързани контролери;
- режим на единичен контролер;
- ротация на приоритета;
- режим на вложени прекъсвания;
- следват или не ICW3 и ICW4.

С ICW2 се определя старшата част на номера на прекъсване. Младшата част се определя от номера i на входната линия IRi. ICW3 се използва при програмиране на главен контролер и определя към коя негова входна линия е свързан подчинен контролер, а ICW4 задава режим на вложена обработка на прекъсванията.

Операционните командни думи се използват за оперативно управление на работата на контролера и за изменение на неговите режими. Те не изискват подреждане в определена последователност. Чрез OSW1 се управлява маскирането по отделните линии за прекъсване IRi. С OSW2 се отбелязва завършване на обслужването на дадено прекъсване, като се нулира съответния бит в регистъра ISR. Тъй като няма достатъчно входове за адресиране на всички вътрешни регистри, OSW3 се използва за избор за следващо четене или на регистър IRR или на регистър ISR.

Разпознаването на управляващата информация и адресирането на вътрешните регистри се извършва по значението на сигналите A0, D4, D3, RD#, WR#, CS# по следната схема:

| A0 | D4 | D3 | RD# | WR# | CS# | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | запис в OSW2 |
| 0 | 0 | 1 | 1 | 0 | 0 | запис в OSW3 |
| 0 | 1 | x | 1 | 0 | 0 | запис в ICW1 |
| 1 | x | x | 1 | 0 | 0 | запис в OCW1, ICW2, ICW3, ICW4 |
| 0 | | | 0 | 1 | 0 | четене от IRR или ISR |
| 1 | | | 0 | 1 | 0 | четене от IMR |

### 5. Особености на обработката на прекъсванията при фамилията P6

В набора изключения, на които реагират процесорите от семейството P6, е включено изключение с номер 18, предизвиквано от схемите за машинен контрол. При възникване на това изключение автоматично се генерира и код за конкретната грешка. Суперскаларността на конвейера и спекулативното изпълнение на инструкциите не оказват влияние на възприемането на прекъсванията и изключенията. Те се възприемат на границата на две инструкции по време на завършващата фаза в завършващото устройство и се обработват по реда на потока инструкции в програмата.

За възприемане и обработка на външни прекъсвания в процесорите е вграден локален

програмируем интелигентен контролер за прекъсвания (Advanced Programmable Interrupt Controller - APIC). Сигналите за външни прекъсвания се получават или чрез входовете LINT[1-0], свързани с локалния APIC или чрез неговия сериен интерфейс - входове PICD[1-0]. Локалният APIC може да бъде свързан към общ за системата APIC за вход-изход (I82489DX) чрез серийния си интерфейс или да работи съвместноо с контролер от типа на I8259A. По този начин входно-изходният контролер на прекъсванията (I82489DX или I8259A) определя номера на прекъсването и го изпраща към локалния APIC. Доставените чрез APIC външни прекъсвания могат да бъдат маскирани посредством флага IF на регистъра EFLAGS.

**Въпроси и задачи:**

1. Допълнете програмата от Приложение 1 с процедури за обработка на изключенията от 1-15 и попълнете таблицата IDT със шлюзове към тях, като ползвате шаблона за изключение 0. За простота процедурите могат само да извеждат на екрана съобщение за номера на възникналото изключение.

2. Изпълнете програмата в симулатора sim486, като от защитен режим програмно генерирате различни изключения. Отбележете промените в стека и вътрешните регистри преди и след всяко от тях.

3. Изпълнете програмата на работния компютър, като след превключване в защитен режим установите програмно режим на трасировка и програмирате действията при възникване на съответното изключение.

4. Снемете с логически анализатор времедиаграмата на шината на МП и сигналите на контролера I8259A при възникване на прекъсване от таймера на дънна платка на IBM PC/AT на базата на I286. Сигналът от таймера постъпва на вход IRQ0 на I8259A. Изводите и съответните сигнали за МП и контролера са описани в предоставения на лабораторното упражнение справочен файл chips.doc.

5. Наблюдавайте с логически анализатор началното програмиране на контролера I8259A. Определете режима на работа, в който той се установява.

6. В програмата от Приложение 2 добавете задача, която да се стартира при прекъсване от таймера. Добавете необходимите структури и дескриптори. Задачата да извежда на екрана текущия брой прекъсвания от таймера.

# 10. Изграждане на мултипроцесорни системи

Развитието и усъвършенстването на архитектурата Intel цели осигуряване на средства и механизми за изграждане и увеличаване на производителността на паралелни мултипроцесорни системи на базата на обща системна шина. На тяхна основа може да се поддържа едновременната работа, както на множество еднотипни процесори, използващи обща памет и входно-изходни устройства (симетрични мултипроцесорни системи), така и на несиметрични системи, състоящи се от Intel процесор и процесор със специално предназначение (наричан обикновено копроцесор). Тези средства и механизми са апаратни и програмни и включват:

- заключване на шината;
- сериализиращи (подреждащи) средства;
- вграден в процесора програмируем интелигентен контролер за прекъсване (APIC);
- средства за поддържане консистентност на кеш паметите;

Основната цел на изброените механизми е да осигурят съгласуваност на отделните процесори и изпълняваните от тях задачи при работа с общата памет и данни.

### 1. Заключване на шината

В мултипроцесорните системи се извършва мултипроцесорна обработка, при която няколко програми или програмни модули се изпълняват едновременно върху различни процесори, за които трябва да се осигури надеждна комуникация с общата памет. Например няколко процесора могат съвместно да използват бит в паметта като индикатор, че даден ресурс (например периферно устройство) е свободен. Процесорът, който иска да използва ресурса, трябва да тества този бит и ако ресурсът е свободен, да реверсира състоянието му. Новото състояние на бита ще сигнализира останалите процесори, че ресурсът е зает. Проблем възниква обаче, ако друг процесор прочете индикатора преди първият процесор да е върнал променената му стойност в паметта. Тогава и двата процесора ще решат, че ресурсът е свободен и ще се опитат да го използват, което може да доведе до неправилни резултати. За да се предотврати тази ситуация в процесорите са въведени заключени цикли на шината, по време на които заявките на друг процесор за използване на шината се игнорират.

Заключването на шината е механизъм, който е вграден във всички модели микропроцесори на Intel. Той осигурява неделимостта на някои операции с паметта посредством активиране на изходния сигнал LOCK#. Типично тези операции се използват за манипулиране с разделяеми даннови структури (семафори, сегментни дескриптори, таблици на страниците и др.). Когато LOCK# е активен, заявките за ползване на общата шина от други процесори се блокират. На фиг.10.1 е показана времедиаграма на заключен цикъл на шината.

Сигналът LOCK# може да се активира автоматично или програмно. Автоматично LOCK# се активира в следните случаи:

- при изпълнение на инструкцията *xchg* с операнд в паметта;
- при установяване на флага B (Busy) на TSS дескриптор. Тъй като този флаг се вдига при активиране на задачата, то по този начин се забранява активиране на една и съща задача от два или повече процесора;
- при промяна на сегментен дескриптор. Докато се копира един дескриптор от дескрипторна таблица в сегментен регистър процесорът активира сигнала LOCK#, така че да не може да бъде модифициран от друг процесор, докато се зарежда. За ефективност процедурите на операционната система, които променят дескриптори, трябва да изпълнят следните стъпки, за да гарантират неговата цялост: със заключено обръщение към байта за право на достъп да маркират дескиптора като неприсъстващ; да променят полетата на дескриптора (това може да изиска няколко обръщения към паметта, поради което не може да се използва префикс *lock*); да използват заключено обръщение към байта за право на достъп, за да маркират дескриптора като присъстващ;
- при изменение на елемент от каталога на страниците или от таблица на страници;
- потвърждаване на прекъсване. Шината е заключена докато контролерът на прекъсване не изпрати номера на прекъсването.

Програмно сигналът LOCK# се активира чрез префикс *lock*, за да заключи шината за следващата инструкция, ако тя се обръща към паметта за модификация на операнд. Една заключена инструкция гарантира заключването само на областта от паметта, определена от операнда-назначение.

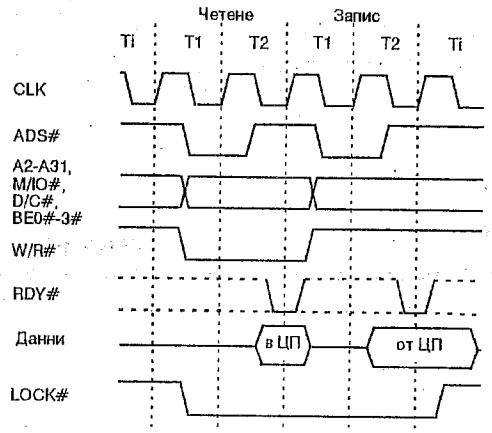
Префиксът ***lock*** е допустим само за следните инструкции:

- инструкции за тестване и модификация на битове: ***bts***, ***btr*** и ***bts***;
- еднооперандни аритметични и логически операции: ***inc***, ***dec***, ***not*** и ***neg***;
- двуоперандни аритметични и логически операции: ***add***, ***adc***, ***sub***, ***sbb***, ***and***, ***or*** и ***xor***;

Когато префиксът ***lock*** се използва пред други инструкции или пред инструкция, която не модифицира операнд в паметта, се генерира изключение "невалиден код на операция".

При процесорите от фамилията P6 префиксът ***lock*** има сериализиращ ефект, т.е. заставя инструкцията да изчака завършване на всички предидущи обръщения към паметта, което означава, че тя ще се изпълни по реда си в програмата.

Механизмът на заключване действа и когато операндът се намира в кеш паметта. При I486 и Pentium кеш паметта е с едновременен запис в основната памет и сигналът LOCK# се появява върху шината по обичайния начин. При P6, ако заключваният операнд е в кеш паметта и тя е с обратен запис, процесорът може да не вдига сигнала LOCK#. Вместо това той модифицира данните в нея. В този случай правилността от изполване на модифицираните общи данни се гарантира от механизма за консистентност на кеш паметите, който се разглежда в т. 5.




Фиг. 10.1. Заключен цикъл на шината

В I486 се гарантира непрекъсваемост при запис и четене на операнди с размер по-голям от 32 бита чрез т.н. псевдозаключване на шината, което се постига с активиране на сигнал PLOCK#. Сигналът PLOCK# се използва само за еднопосочни предавания и означава, че текущият и следващите цикли, докато е активен PLOCK#, трябва да се третират като неделимо предаване. Чрез псевддозаключване се предават: ред на кеш, операндите с плаваща запетая и двойни думи по 8 или 16 разредна шина.

Сигналите LOCK# и PLOCK# блокират арбитрирането на общата шина, при мултипроцесорни системи, изградени на базата на арбитър или участват в изработването на сигналите за готовност (RDY# и BRDY#) за останалите процесори, като предизвикват въвеждане на тактове на изчакване, при системи неизползващи шинен арбитър.

### 2. Сериализиращи средства

#### 2.1. Подреждане на обръщенията към паметта

По съответствието на реда на изпълнение на обръщенията към паметта по шината и реда им в потока инструкции на програмата се различават строго подреждане и процесорно подреждане. При строгото подреждане обръщенията към паметта се изпълняват по шината по реда на срещане в потока инструкции. Това подреждане е характерно за процесорите до I386 включително. С цел повишаване на производителността при процесорното подреждане се разрешава четенията от паметта да изпреварват записите, които се буферират. В I486 и Pentium обикновено се използва строго подреждане с изключение на несполучили в кеш операции четене. На тях се разрешава да изпреварят буферираните записи, когато последните са попаднали в кеш паметта и следователно не са насочени към същите адреси, поради което не съществува опасност от четене на некоректни данни.

В P6 основно се използва процесорно подреждане, произтичащо от характерното за неговата суперскаларност спекулативно изпълнение на инструкциите. За процесорното подреждане е характерно:

• операциите четене могат да се изпълняват спекулативно не по реда си в програмата и могат да изпреварват буферираните записи;

• записите могат да се буферират и винаги се изпълняват съгласно реда си в програмата;

• операциите четене и запис не могат да изпреварват операциите за вход-изход, заключените операции и сериализиращите инструкции;

• входно-изходните и заключените инструкции обезателно изчакват завършване на всички предидущи операции, т.е. те се изпълняват строго по реда си и могат да се използват като синхронизиращ механизъм в мултипроцесорните системи.

За операциите към отделни области от паметта може да се дефинира строго или слабо подреждане чрез задаване на тип посредством регистрите MTRR (Memory Type Range Registers). Тези регистри управляват кеширането на избраните области от паметта, които могат да бъдат от следния тип: некешируема (UC), защитена от запис (WP), с едновременен запис (WT), с обратен запис (WB). Типът на областта задава и възможния тип на подреждане на операциите към нея. Например некешируемите области памет изискват строго подреждане на операциите към тях, а областите, кешируеми с обратен запис (WB) допускат слабо подреждане и спекулативно изпълнение. Механизмът, базиран на MTRR, позволява дефиниране до 96 области във физическата памет.

За Pentium и I486 подобно, но не идентично управление, може да се постигне със сигналите KEN# и WB/WT#. Те разрешават външно апаратно управление на метода за кеширане, използван за дадени области от паметта.

### 2.2. Сериализиращи инструкции

В архитектурата Intel са дефинирани няколко сериализиращи инструкции. Те заставят процесора да завърши всички модификации на флагове, регистри и клетки в паметта, започнати от предидущите инструкции и да изпълни буферираните операции запис преди следващата инструкция да бъде извлечена и изпълнена. Например след зареждане с инструкцията ***mov*** на управляващия регистър CR0 за разрешаване на защитен режим процесорът трябва да изпълни сериализираща операция. Това гарантира, че всички стартирани в реален режим инструкции ще завършат преди да се извърши превключването в защитен режим.

Сериализиращите инструкции са въведени в Pentium за поддържане на паралелната работа на двата конвейера, а в P6 за ограничаване на спекулативното изпълнение. Те могат да бъдат:

• привилегировани (допустими само от привилегировано ниво 0) - ***mov*** в управляващ регистър, ***mov*** в регистър за тестване, ***wrmsr***, ***invd***, ***invlpg***, ***wbinvd***, ***lgdt***, ***lldt***, ***lidt*** и ***ltr***;

• непривилегировани - ***cpuid***, ***iret*** и ***rsm***.

Никоя инструкция не може да пререди сериализираща инструкция и сериализираща инструкция не може да пререди никоя друга инструкция. При използване на сериализиращи инструкции следва да се има пред вид, че динамичното предсказване на преходите предизвиква извличане на инструкциите от посоката на прехода преди инструкцията за преход да бъде в действителност изпълнена. От това следва, че изпълнението на сериализиращи инструкции след изпълнението на преход няма да бъде детерминирано.

## 3. Програмируем интелигентен контролер на прекъсвания (APIC)

### 3.1. Мултипроцесорни системи на базата на APIC контролери

Един от вариантите за построяване на ефективни симетрични мултипроцесорни системи е базиран на използване на APIC контролери. APIC се включва към всеки процесорен модул в системата и към общите входно-изходни устройства. APIC контролерите служат за реализиране на симетричност и по отношение на входно-изходните устройства и обработката на прекъсвания, което позволява да се повиши еднородността на системата и се елиминират тесните места при изпълнение на входно-изходните операции. Това се постига благодарение на следните съображения:

• при настъпване на прекъсване най-желателно е да се прекъсне този процесор, който изпълнява задача с най-нисък приоритет в системата;

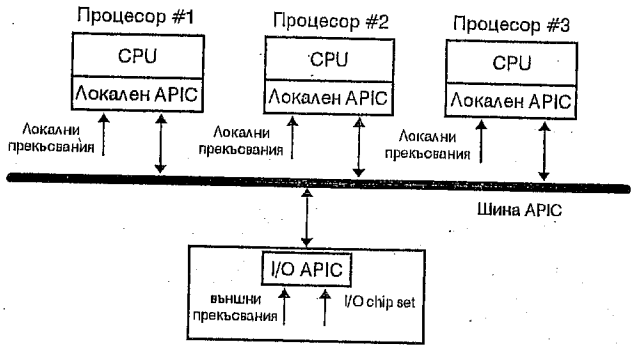
• прекъсване от входно-изходните устройства може да се получи от процесор, различен от

този, който е стартирал входно-изходна операция;

• по-голяма ефективност се постига чрез изпълнение на прекъсване от процесора, който последен е изпълнил прекъсване от същия източник, тъй като той още съхранява кода на обработващата програма в своята кеш памет;

• възможност един процесор да прекъсва при необходимост друг процесор.

В системата всички APIC контролери работят заедно като едно устройство, комуникирайки помежду си чрез собствен последователен интерфейс. В системи, изградени на базата на I486 и Pentium, се използват APIC контролери в отделни интегрални схеми I82489DX, а при P6 APIC контролерът е вграден в процесора. На фиг.10.2. е показана примерна структура на мултипроцесорна система.




Фиг. 10.2 Мултипроцесорна система на базата на APIC контролери

Локалният APIC контролер изпълнява две основни функции:

• обработва локалните за процесора прекъсвания, получени по неговите входове LINT[0-1] или генерирани от програмите;

• в мултипроцесорна конфигурация получава по шината APIC предназначените за него съобщения за прекъсване и ги обработва.

Локалният APIC управлява диспечеризацията на прекъсванията към собствения си процесор, като осигурява средства за маскиране и вложеност на обработката. Вграденият в него таймер позволява локално генериране на прекъсвания.

APIC контролерът за вход-изход получава сигнали за настъпили външни събития на прекъсване и съобщения за междупроцесорни прекъсвания от процесорите на системната шина и ги разпределя между локалните APIC контролери. Менажирането се извършва по статична или динамична схема, която позволява рутиране на прекъсванията към най-ниско приоритетния процесор. APIC контролерите се свързват чрез обща серийна шина (известна като ICC шина) и комуникират помежду си чрез съобщения, които са програмно невидими.

Те могат да поддържат 240 различни вектора на прекъсване (с номера от 16 до 255), групирани в 15 приоритетни нива с нарастване на приоритета от 1 към 15. Приоритетът се определя по формулата *vector / 16*.

### 3.2. Структура и функциониране на локален APIC

#### 3.2.1. Източници за прекъсване в APIC

APIC контролерът може да възприема прекъсвания от следните източници:

**Локални, генерирани от средата на собствения процесор на контролера:**

• входовете за прекъсване на процесора LINT[0-1], на които постъпват прекъсвания от локално подключените входно-изходни устройства;

• вградения таймер;

• броячите за следене на производителността, включени в процесорите P6;

• грешки;

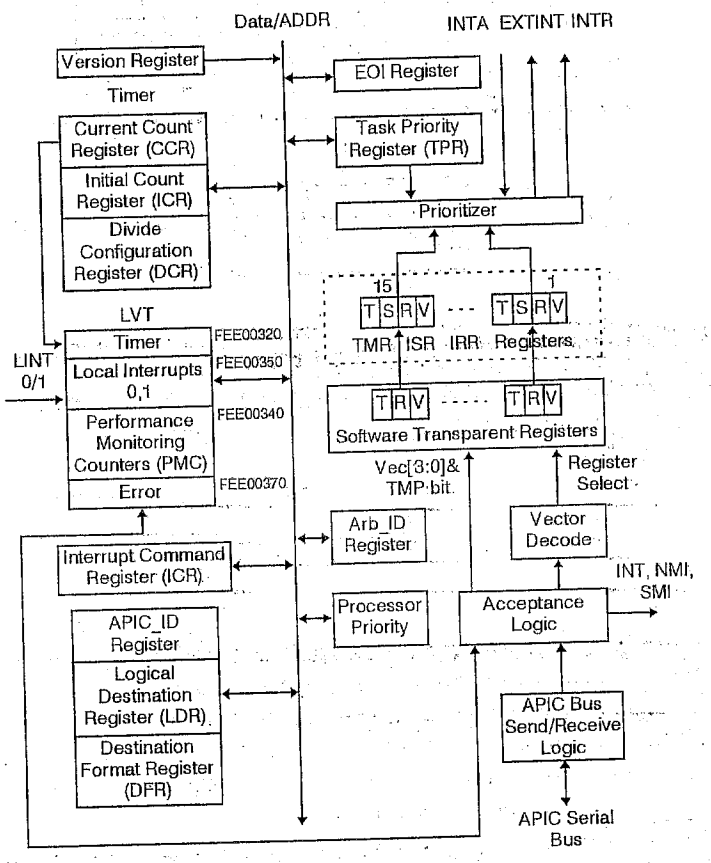
• програмни прекъсвания от собствения процесор;

**Системни, генерирани от други MII и общи компоненти на мултипроцесорната система:**

• съобщения от входно-изходния APIC контролер;
• съобщения от друг локален APIC, доставящи междупроцесорни прекъсвания.

### 3.2.2. Структурна схема

На фиг.10.3 е показана вътрешната структура на APIC контролер. Съдържащите се в нея регистри са програмно достъпни и са разположени на постоянни адреси в адресното пространство на процесора, като заемат 4K от адрес FEE00000h. При четене и запис по тези адреси при P6, за разлика от Pentium, не се активират цикли по шината, тъй като APIC е в кристала на процесора. Чрез зареждане на подходящо значение на регистъра APIC_BASE_MSR на МП може да се зададе алтернативен базов адрес на APIC регистрите..




Фиг. 10.3. Структура на APIC контролер

APIC контролерът се състои от следните функционални възли:

#### Схема за възприемане на прекъсвания от локалните източници

Всеки от локалните източници на прекъсване е индивидуално програмируем чрез съответенетен 32-разреден регистър, включен в локалната векторна таблица LVT. Всеки регистър на LVT задава за съответното прекъсване: вектор, начин на приемане - по ниво или по фронт, флаг за маскиране, режим на възприемане, състояние, отдалечен IRR бит. Векторът се използва като вход в таблицата IDT за обработка на прекъсването. Режимът на възприемане се задава с триразредно поле и може да бъде:

• фиксиран - прекъсването се възприема като прекъсване, доставено чрез сигнала INTR# на локалния процесор, но не се стартира цикъл по шината за получаване на неговия вектор. Използва се

векторът от полето вектор на съответния регистър;

• NMI - възприема се като прекъсване, доставено чрез сигнала NMI# на локалния процесор. Полето за вектор се игнорира, тъй като прекъсванията от тип NMI се задават със стандартен вектор - 2. За разлика от прекъсванията NMI обаче това прекъсване може да бъде маскирано с бита за маска в регистъра;

• ExtINTA - възприема се като сигнал INTR#, получен от външно подключен контролер тип I8259A. Предизвиква стартиране на цикъл по шината за получаване на вектора от външния контролер. Само един вход в APIC контролера може да бъде дефиниран като ExtINTA.

Състоянието на прекъсването се задава с едноразредно поле и може да бъде "празно", т.е. по съответния вход няма активно прекъсване, и "задържано", т.е. прекъсването е прието, но все още не е подадено в процесора за обработка. Полето за състояние се използва от програмите като флаг за готовност.

Отдалеченият IRR бит се отчита само, ако прекъсването се приема по ниво и се установява в 1 при неговото постъпване. Този бит се снема апаратно при получаване на команда EOI (край на обработката) от процесора и дублира съответния разред на регистъра IRR в контролера.

**Схема за изпращане на съобщения за прекъсване**

Процесор може да генерира междупроцесорно прекъсване посредством запис в регистъра ICR (Interrupt Command Register). Запис в ICR предизвиква формиране и изпращане на съобщение за прекъсване към другите APIC контролери в системата по последователната шина APIC. Форматът на регистър ICR позволява на процесора да генерира произволно прекъсване, в това число да препрати възприето от него прекъсване към други процесори. ICR съдържа следните полета:

• вектор - идентифицира изпращаното прекъсване и служи за индексация в таблицата IDT на получаващия процесор;

• назначение - 32-разредно поле, идентифициращо получаващия процесор (приемник). Съдържанието му се интерпретира в зависимост от значението на други две полета - режим за назначение и съкратено назначение;

• режим за назначение - определя интерпретацията на полето за назначение и може да бъде физически или логически. При физически режим полето за назначение съдържа идентификатора на приемника, а при логически се интерпретира съгласно съдържанието на регистрите DFR и LDR на локалния APIC;

• съкратено назначение - индицира дали се използва съкратено задаване на назначението, което може да бъде: програмно самопрекъсване, при което текущият APIC е единственият приемник на прекъсването; прекъсване към всички и към себе си; прекъсване към всички без себе си. Ако се използва (различно е от 0) отменя действието на полето за назначение;

• режим на доставяне - конкретизира приемащия APIC и неговата реакция при получаване на съобщението:

*Фиксиран* - задава доставяне на съобщението до всички, зададени с полето за назначение контролери, и възприемане на прекъсването като прекъсване чрез сигнал INTR#;

*Най-нисък приоритет* - доставка на прекъсването като сигнал INTR# до процесора, изпълняващ задача с най-нисък приоритет сред причислените в полето за назначение процесори;

*NMI* - доставя прекъсването като прекъсване NMI до всички процесори, зададени в полето за назначение;

*INIT* - доставя прекъсването като сигнал INIT до всички процесори, зададени в полето за назначение.

*Start_Up* - изпраща специално съобщение на процесорите в мултипроцесорната система, полето вектор на което задава началния адрес на протокола за междупроцесорно зареждане;

*INIT Level De_assert* - синхронизиращо съобщение към всички APIC за установяване на началната стойност на своя арбитражен идентификатор.

• състояние на изпращане - "празно" - означава, че няма прекъсване за изпращане, "задържано" - започнало е предаване на прекъсването, но още не е завършило. Този бит е достъпен само за четене и се използва за проверка дали текущото прекъсване е вече изпратено и ICR е свободен за нов запис;

• начин на приемане - по фронт или по ниво;

**Схема за получаване на съобщения за прекъсване**

Всеки APIC съдържа три 256-разредни регистъра, които играят важна роля при приемане на прекъсванията за локалния процесор - регистър за заявки за прекъсване (IRR), регистър за обслужвани прекъсвания (ISR) и регистър за начин на приемане - по фронт или ниво (TMR). Всеки разред **i** на тези регистри се отнася за прекъсване с вектор **i**. Тъй като вектори от 0 - 16 са резервирани, то разредите от 0 до 16 на тези регистри не се използват. Регистрите IRR и ISR имат същото предназначение както едноименните регистри в контролера I8259A. IRR съдържа прекъсванията, приети от контролера, но все още не предадени в процесора за обслужване. Съответен бит в IRR се вдига, когато APIC приеме прекъсване, и се снема, когато се изпълнява цикъл INTA. Регистър ISS съдържа прекъсванията, които в момента се обслужват от процесора. Той отразява текущото състояние на опашката за прекъсвания към процесора. При цикъл INTA съответстващият на най-високо приоритетния бит в IRR разред на ISR се установява в 1. По време на цикъл EOI най-приоритетният разред на ISR се нулира и ако съответния бит в TMR е 1 се изпраща съобщение EOI (край на обработката) до всички входно-изходни APIC. При приемане на прекъсване съответният бит в TMR се нулира, ако прекъсването е по фронт и се установява в единица, ако прекъсването е по ниво. Регистрите IRR, ISR и TMR са програмно достъпни само за четене, като всеки от тях е достъпен като 4 отделни 32 разредни регистъра.

Следва да се отбележи, че за разлика от I8259A в APIC отсъства явен регистър на маските на прекъсванията (IMR). Процесорът маскира прекъсванията посредством запис в регистъра за приоритет на задачата (TPR).

APIC контролерът следи съобщенията по APIC шината, като сравнява 32 разредното поле за назначение във всяко съобщение със своя LDR или FID регистър според режима за адресация. Ако режимът за доставка е:

- фиксиран, то прекъсването безусловно се приема от всички адресирани APIC, като се вдига съответният бит в IRR.
- най-нисък приоритет - прекъсването е предназначено за МП с най-нисък приоритет. Всеки APIC в адресираната група проверява дали не е текущ фокус на прекъсването чрез тестване на съответния бит в IRR и ISR. Ако някой APIC открие един от тези битове в 1 за пристигащото прекъсване, то той го приема, независимо от приоритета си и сигнализира за това останалите APIC, за да абортират приемането по приоритет. Процесорът е фокус за дадено прекъсване, ако текущо го обслужва или съдържа вече запомнена информация за него.
- NMI или INIT - всички адресирани APIC безусловно приемат прекъсването, като активират съответните схеми на процесора. Регистрите IRR и ISR не се използват за тези типове прекъсвания, а векторът е неопределен.

**Приоритетна логика**

В APIC контролерите е реализиран приоритетен механизъм за прекъсване на процесора, изграден на базата на регистър за приоритет на задачата (TPR). Обслужват се само прекъсвания с приоритет, по-голям от зададения в TPR. Другите прекъсвания се обслужват веднага след като значението на TPR намалее достатъчно, за да позволи това. Превключването на задачите и смяната на техния приоритет са резултат на явни действия на операционната система. Използвайки TPR, ОС може временно да блокира специфични прекъсвания, за да не нарушават работата на по-високоприоритетните задачи. Приоритетният механизъм не работи за прекъсвания, доставени без вектор - ExtINT, NMI, INIT, Start_Up. Например прекъсване с вектор 123 има приоритет, изчислен като *123 / 16 = 7,* и може да бъде маскирано от всяка задача с приоритет по-голям или равен на 7. Тъй като максимално възможният приоритет е 15, то зареждане на 15 в регистъра TPR маскира всички прекъсвания.

**Край на обработка на прекъсване**

Преди връщане от процедурата за обработка на прекъсване трябва да се изпрати в APIC команда "Край на прекъсване" (EOI) посредством запис в неговия регистър EOI. По този начин се съобщава на APIC да нулира най-приоритетния бит в регистъра ISR, тъй като неговата обработка е завършила. След получаване на EOI, APIC определя следващото най-приоритетно прекъсване и го подава в процесора.

### 3.2.3. Арбитраж на APIC шината

Свързаните към общата APIC шина контролери се арбитрират, за да получат разрешение да изпратят съобщение по нея. Приоритетът на контролерите при арбитраж е уникален в диапазона 0-15. Арбитражният приоритет на всеки APIC се съхранява в неговия регистър Arb_ID и динамично се модифицира след всяко успешно предаване на съобщение, за да се гарантира справедливост. Всяко изпращане на съобщение се предшества от арбитражна фаза, при която сред всички APIC, заявили ползване на шината, се определя този с най-висок арбитражен приоритет и на него се разрешава да изпрати съобщение. Победилият APIC установява своя арбитражен приоритет на най-ниско ниво - 0, а несполучилите APIC увеличават арбитражните си приоритети с 1. При инициализация арбитражният приоритет се установява равен на съдържанието на регистъра APIC_ID.

### 3.2.4. Междупроцесорни прекъсвания. Механизми за разпределяне на прекъсванията

APIC контролер може да изпраща прекъсване към всички или към произволно подмножество от процесорите в системата, в това число и към себе си. APIC контролерите се различават в системата по своите идентификатори (APIC ID). Използват се два режима за задаване на назначението на прекъсването:

• физически. Процесорът получател се задава чрез неговия локален APIC ID. По този начин може да бъде адресиран единичен (ID от 0 до 14) APIC или всички APIC контролери (ID = 15). Идентификаторът на APIC се съхранява в регистъра APIC_ID. Съдържанието на този регистър се зарежда при включване на захранването чрез входовете на процесора A11#, A12#, BR0#-BR3# за P6 и BE0#-BE3# за Pentium. Идентификаторът може да бъде четен и променян програмно;

• логически. Процесорът получател се задава чрез 8 разреден адрес (MDA - Message Destination Address), който се сравнява с 8 разредните логически идентификатори на APIC. Логическият идентификатор на всеки APIC се съхранява в регистъра LDR (Logical Destination Register). Интерпретацията на MDA зависи от съдържанието на регистъра DFR (Destination Format Register), чието съдържание задава модела на получаване на съобщенията.

Моделът за получаване на съобщенията зависи от начина на свързване на APIC контролерите към APIC шината.. Дефинирани са два модела за получаване на прекъсванията - плоск и клъстерен (групов).

• Плоск модел. При плоския модел в системата могат да съществуват до 8 APIC контролера. Адресът MDA се интерпретира като декодиран адрес - единица в даден негов разред адресира съответния номер контролер. По този начин произволно подмножество контролери може да бъде адресирано посредством установяване на 1 в съответните битове на MDA. Логическият номер на даден APIC се програмира с единица в съответния бит на регистъра LDR. Съобщение към всички контролери се изпраща чрез установяване на 1 във всички разреди на MDA.

• Клъстерен модел. За клъстерния модел са допустими две основни комуникационни схеми - плоска и йерархична. При плоския клъстерен модел всички клъстери (групи от APIC контролери) са свързани към обща APIC шина. Той позволява да се адресират 15 клъстера с по 4 процесора. Йерархичният модел позволява да се построи произволна йерархична мрежа чрез съединение на различни плоски клъстери посредством независими APIC шини. Схемата изисква менажер във всеки клъстер, който отговаря за прехвърляне на съобщенията между отделните APIC шини.

В APIC контролерите са вградени два механизма за разпределяне на прекъсванията:

• статичен. Използва се за достъп до определен процесор в мрежата. Прекъсването безусловно се доставя до всички APIC, които отговарят на информацията за назначение в съобщението;

• динамичен. Пристигащите прекъсвания се предават на най-ниско приоритетния процесор. Това се програмира в LVT за доставка на локалните прекъсвания или в ICR за съобщенията по шината. Ако съществува процесор-фокус, той може да приеме прекъсването, независимо от приоритета.

## 4. Консистентност на кеш паметите

В мултипроцесорни системи съществено значение придобива проблемът за осигуряване на консистентност на вътрешните кеш памети. Един или повече процесора могат да вземат част от общите разделяеми данни от системната памет във вътрешната си кеш памет. Всяко следващо изменение на тези данни се извършва в кеш паметта и ако тя е с обратен запис ще остава невидимо за останалите процесори, работещи с тези данни. В това се състои проблемът за консистентност (съгласуваност, непротиворечивост) на кеш паметите и системната памет. За осигуряване на консистентност, започвайки от Pentium, в архитектурата Intel е въведен т.н. MESI (modified,

exclusive, shared, invalid) протокол. Протоколът е изграден на базата на вградена към вътрешните кеш памети логика за следене на предаванията по шината и флаговете за състояние на всеки ред в нея. Редовете могат да бъдат в следните състояния:

- *изключителен* (*exclusive*) - редът се намира в кеш паметта само на текущия процесор;
- *разделяем* (*shared*) - редът се намира в кеш паметта и на друг процесор;
- *модифициран* (*modified*) - съдържанието на реда е променено от момента на въвеждането му в кеш паметта;
- *невалиден* (*invalid*) - съдържанието на реда е невалидно.

Следящите логики на различните процесори обменят информация с помощта на сигналите:

- HIT# - съдържа немодифициран ред в кеш;
- HITM# - съдържа модифициран ред в кеш;
- DEFER# - ако не е активен индицира, че на текущото предаване се гарантира завършване по реда на започването му.

Следящата логика следи предаванията на другите процесори по шината, за да открие дали друг процесор се опитва да осъществи достъп до област, която е в неговата кеш. При това са възможни следните случаи:

- данните не са в кеш паметта на следящия процесор и той не се намесва в предаването по шината;
- данните от тази област са в кеш, но са в немодифицирано състояние. В този случай следящият процесор активира сигнала HIT# и променя състоянието на съответния ред на *разделяем*. Другият процесор завършва операцията към паметта и също отбелязва въведения ред като *разделяем*;
- данните от тази област са в кеш и са модифицирани, но още не са записани обратно в системната памет. Следящият процесор активира сигнала HITM#, индицирайки по този начин, че ще изпълни обратен запис на модифицираните общи данни. При активен HITM# контролерът на паметта не предоставя адресираните данни, като вместо това се настройва за изпълнение на обратен запис на ред от кеш. Обратният запис се предава директно към инициализиращия предаването процесор и се следи от контролера на паметта, който е отговорен за актуализацията на данните в системната памет;
- следящата логика не е готова да определи състоянието и се нуждае от задържане на предаването, като активира едновременно сигналите HIT# и HITN#.

В процесорите I486 е реализиран протокол с две състояния на редовете в кеш - валидно и невалидно. Обявяването на ред от вътрешната кеш за невалиден може да се извърши от външна схема чрез активиране на специален цикъл по шината със сигнала AHOLD#.

**Въпроси и задачи:**

1. Реализирайте на асемблер семафорните P и V операции с командата ***xchg***. Снемете с логически анализатор времедиаграмите на шината на МП при изпълнение на тези операции. Как се осигурява неделимостта на операциите? Как се променя времедиаграмата, ако операндът-назначение не е в паметта, а във вътрешен регистър или кеш паметта?

2. Решете същата задача като използвате префикс *lock*. Какви са предимствата и недостатъците на второто решение? Какво ще се получи, ако префиксът е пред инструкция, двата операнда на която не са в паметта?

3. Какви са новите функционални възможности на мултипроцесорните системи, построени на базата на APIC контролери? Постройте клъстърен модел на свързване на микропроцесорите към шината APIC. Какви са неговите особености и достойнства по отношение на плоския модел?

4. Обяснете необходимостта от сериализиращи инструкции? Защо привилегированите инструкции трябва да бъдат сериализиращи?

5. В какво се изразява проблемът за консистентността на кеш паметите в мултипроцесорните системи? Какви средства апаратни и програмни са включени в микропроцесорите за неговото решение?

6. Проектирайте арбитражна схема за работа с обща шина на два и повече процесора I486 или Pentium, като отчитате заключването на шината. Кои сигнали на процесора могат да се използват за арбитраж? При мултипроцесорни системи, построени на базата на обща шина, е необходима арбитражна логика (арбитър) за регламентиране на ползването на шината от отделните процесори. Предоставянето на шината се извършва на базата на приоритет, който може да бъде статичен или динамичен. В началото на всеки цикъл към системната шина МП издава заявка към арбитъра. Анализирайки получените заявки арбитърът определя най-приоритетната от тях и изпраща сигнал-разрешение към съответния МП.

# 11. Програмируем паралелен интерфейс. Таймери

Паралелният интерфейс служи за предаване между микропроцесора и периферни устройства на информация в паралелен формат. При програмно управлявано предаване на данните през него се изпълняват три основни операции:

Операции при извеждане на данни:

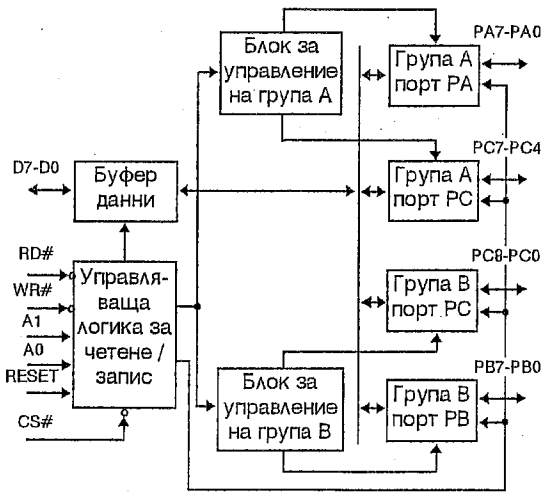
- подготовка на извежданите данни чрез записване в изходния порт;
- изчакване на сигнал, потвърждаващ готовността на ВУ за приемане на данните;
- към ВУ се изпраща сигнал за предаване на данните (сигнал за валидни данни).

Операции при въвеждане на данни:

- подготовка за въвеждане на данни - изчистване на входния регистър;
- изчакване на сигнал от ВУ за предаване на данните;
- четене на данните и предаване на сигнал към ВУ за завършване на четенето.

### 1. Програмируем паралелен интерфейс I8255A

Схемата I8255A осигурява програмируем паралелен интерфейс за различни ВУ с шината на микропроцесора. Свързването на I8255A към данновата шина се осъществява чрез двупосочните изводи D7-D0 и вътрешен двупосочен буфер (фиг.11.1). За връзка с ВУ в структурата на схемата са включени три 8 разредни порта - PA, PB и PC, всеки от които може да бъде отделно програмиран от МП за работа в различни режими и конфигурации. Портовете PA и PB се конфигурират или като 8 разредни изходни или като 8 разредни входни регистри. Порт PC се конфигурира като 8 разреден изходен регистър, като 8 разреден входен регистър или разделен на два 4 разредни порта. Всеки 4 разреден порт може да бъде използван за изходни управляващи сигнали или за входни сигнали на състояние към портовете PA и PB. Логиката за управление на конфигурирането и на режимите на работа на портовете е разделена в два блока: блок за управление на портовете в група A (PA и PC7-PC4) и блок за управление на портовете в група B (PB и PC3-PC0). Портовете могат да работят в три основни програмно задавани режима (фиг.11.2): режим 0 - основен вход-изход; режим 1 - стробиран вход-изход; режим 2 - двупосочна шина. Режимите за отделните портове могат да се комбинират. Например порт PB може да бъде програмиран в режим 0 за управление на ключове и индикация, а порт PA - в режим 1 за управление на клавиатура или друго ВУ на базата на прекъсване. В режим 1 и 2 всеки от разредите на порт PC може да бъде автономно установяван в състояние 1 или 0 с една инструкция запис. Изходните сигнали от порт PC, служещи за реализация на управлението на интерфейса с ВУ, могат да се използват и като сигнали за прекъсване.

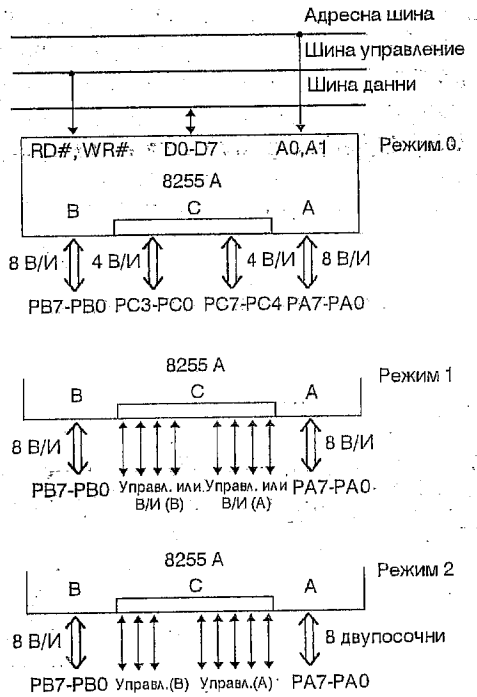



Фиг. 11.1. Структура на I8255A

Функционирането на портовете се програмира чрез запис на определена информация във вътрешен управляващ регистър, свързан с блоковете за управление. Портовете и управляващият регистър заемат 4 адреса от входно-изходното пространство на МП и се адресират чрез сигналите CS#, A1 и A0, а записът и четенето се задават съответно със сигналите WR# или RD#:

| A1 | A0 | WR# | RD# | CS# | операция |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | запис в PA |
| 0 | 1 | 0 | 1 | 0 | запис в PB |
| 1 | 0 | 0 | 1 | 0 | запис в PC |
| 1 | 1 | 0 | 1 | 0 | запис в управляващия регистър |
| 0 | 0 | 1 | 0 | 0 | четене от PA |
| 0 | 1 | 1 | 0 | 0 | четене от PB |
| 1 | 0 | 1 | 0 | 0 | четене от PC |

Останалите комбинации на сигналите не се използват, т.е. при неактивен CS# или липса на входно-изходна операция (WR#=1 и RD#=1) вътрешният буфер е в състояние на висок импеданс. Сигналът RESET нулира управляващия регистър и конфигурира всички портове като входове.




Фиг. 11.2. Основни режими

Управляващата информация, която се записва в управляващия регистър, има следния формат:

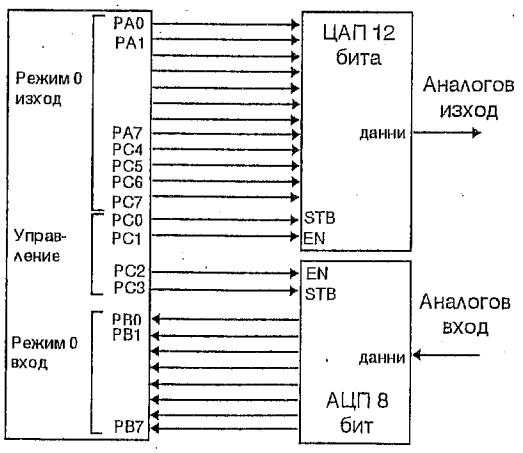
| бит | състояние | функция |
|---|---|---|
| 7 | 1 | Установяване режимите на портовете |
| | | Портове от група A |
| 0 | 0 | Конфигуриране на PC3-PC0 като изход |
| | 1 | Конфигуриране на PC3-PC0 като вход |
| 1 | 0 | Конфигуриране на PB като изход |
| | 1 | Конфигуриране на PB като вход |
| 2 | 0 | Установяване на режим 0 за PB и PC3-PC0 |
| | 1 | Установяване на режим 1 за PB и PC3-PC0 |

| бит | състояние | функция |
|---|---|---|
| | | Портове от група А |
| 3 | 0 | Конфигуриране на PC7-PC4 като изход |
| | 1 | Конфигуриране на PC7-PC4 като вход |
| 4 | 0 | Конфигуриране на PA като изход |
| | 1 | Конфигуриране на PA като вход |
| 6,5 | 00 | Установяване на режим 0 за PA и PC7-PC4 |
| | 01 | Установяване на режим 1 за PA и PC7-PC4 |
| | 1X | Установяване на режим 2 за PA и PC7-PC4 |
| 7 | 0 | Установяване/нулиране на отделните разреди на PC |
| 6,5,4 | XXX | без значение |
| 3,2,1 | 000 | Избиране на разред 0 на PC |
| | 001 | Избиране на разред 1 на PC |
| | ... | |
| | 111 | Избиране на разред 7 на PC |
| 0 | 0 | Нулиране на избрания разред на PC |
| | 1 | Установяване на избрания разред на PC |

## 2. Използване на I8255A

Схемата I8255A е удобно средство за изграждане на оптимален и достатъчно гъвкав интерфейс с ВУ без необходимост от допълнителна външна логика. С всяко ВУ в микропроцесорната система е свързана обслужваща програма (драйвер). Тази програма управлява интерфейса между МП и ВУ като съблюдава характеристиките на този интерфейс по предавани данни и временни съотношения на сигналите. На фиг.11.3 е показано примерно свързване на цифрово-аналогов (ЦАП) и аналого-цифров (АЦП) преобразователи към шината на МП посредством I8255A. Портовете на страна А осигуряват 12 разредни цифрови изводи към ЦАП, порт PB - 8 цифрови входа от АЦП. Сигналите PC0 и PC1 се използват като управляващи сигнали към ЦАП, като задават моментите на извеждане на данните (STB) и момента на разрешение на аналоговия сигнал на изхода на ЦАП (EN). Аналогично PC2 и PC3 служат за задаване на моментите на приемане на данните от АЦП (STB) и стартиране на преобразуването (EN).

В персоналните компютри от типа на IBM PC XT/AT на базата на 16 разредни микропроцесори схемата I8255A е включена в системната платка и се програмира за работа в режим 0, като осигурява два 8 разредни и два 4 разредни входно-изходни порта. Разположена е на базов адрес 60h, от който се изработва сигналът CS#. Порт PA се конфигурира като вход, на който постъпват преобразуваните в паралелен код данни от клавиатурата. Порт PB се конфигурира като изходен. Чрез него МП задава различни управляващи функции в системната платка. Порт PC е вход, на който постъпва информация за състоянието на превключвателите на системната платка.




Фиг. 11.3. Свързване на ЦАП и АЦП към I8255

В персоналните компютри на базата на 32 разредни микропроцесори паралелният интерфейс е включен в периферни чипове с висока степен на интеграция, например в моста PCI-ISA - PIIX3 или PIIX4 (82371AB), но с цел съвместимост на програмното осигуряване заема същите адреси във входно-изходното адресно пространство.

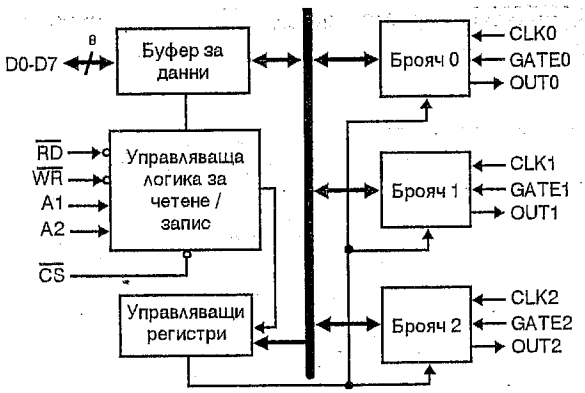
Следващият пример демонстрира фрагмент на програма за обработка на прекъсване от клавиатурата. При натискане на клавиш контролерът на клавиатурата изпраща кода на неговия знак в порт PA и издава сигнал за прекъсване чрез входа IRQ1 на контролера за прекъсване I8259. Програмата за обработка на прекъсването прочита стойността на клавиша от порт PA с адрес 60h. Уведомяването на контролера, че МП е приел текущия знак и може да получи следващ се извършва чрез промяна на сигнала от PB7 (адрес 61h). Преди завършването си процедурата за обработка на прекъсването изпраща към I8259 командата EOI. Тази команда нулира бит 1 на регистъра ISR на I8259. Регистърът ISR е разположен на адрес 20h.

```
keyboard_interrupt proc
   push AX          ;запомняне на рег. AX
   in   AL, 60H     ;четене стойността на текущия клавиш
   push AX          ;запомняне на стойността на клавиша
   in   AL, 61H     ;четене на стойността на порт PB
   mov  AH, AL      ;запомняне на стойността на порт PB
   or   AL, 80H     ;PB7 = 1
   out  61H, AL     ;сигнал, че е получен знак
   mov  AL, AH      ;
   out  61H, AL     ;възстановяване на старото състояние на PB
   pop  AX          ;възстановяване на стойността на клавиша
   . . .
   mov  AL, 20H     ;
   out  20H, AL     ;изпращане на команда EOI
   pop  AX          ;възстановяване на стойността на AX
   iret             ;връщане в прекъснатата програма
```

### 3. Таймери

В микропроцесорните системи и в персоналните компютри за измерване на различни интервали от време чрез програмно управление към системната шина се подключва модул таймер. Основното предназначение на таймера е:

- генериране на сигнали с еталонна честота с цел поддържане на системния часовник;
- генериране на сигнал за опресняване на динамичната памет;
- отчитане на квант от време в многозадачните ОС, базирани на времеделене;
- генериране на сигнали с програмируема продължителност и коефициент на запълване и др.




Фиг. 10.4. Структурна схема на I8253

В персонални компютри IBM PC XT/AT таймерът е изграден на базата на интегрални схеми I8253 или I8254. В P6 таймерът е включен в локалния APIC на процесора. В системи, изградени на базата на I486 и Pentium, таймерът се включва в интегрираните периферни чипове, например PIIX3 или PIIX4. Разликата между I8253 и I8254 е само в максималната честота на входния тактов сигнал. За I8253 тази честота е 2 MHz, а за I8254 - 10 MHz.

### 3.1. Структура

Схемата I8253 съдържа три независими интервални таймера, всеки от които може да бъде програмиран независимо за работа в различни режими (фиг.11.4). Таймерите са изградени от 16 разреден брояч и от 16 разреден управляващ регистър (CR). С всеки таймер са свързани независими входове за тактови сигнали CLK, входове за управляващи сигнали GATE, разрешаващи или забраняващи генерираните сигнали на изходите OUT.

Вътрешните регистри заемат 4 адреса от входно-изходното адресно пространство на МП и се адресират чрез сигналите CS#, A1 и A0, а записът и четенето се задават съответно със сигналите WR# или RD#:

| A1 | A0 | WR# | RD# | CS# | операция |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | зареждане на брояч 0 |
| 0 | 1 | 0 | 1 | 0 | зареждане на брояч 1 |
| 1 | 0 | 0 | 1 | 0 | зареждане на брояч 2 |
| 1 | 1 | 0 | 1 | 0 | запис в управляващия регистър |
| 0 | 0 | 1 | 0 | 0 | четене от брояч 0 |
| 0 | 1 | 1 | 0 | 0 | четене от брояч 1 |
| 1 | 0 | 1 | 0 | 0 | четене от брояч 2 |

Останалите комбинации на сигналите не се използват, т.е. при неактивен CS# или липса на входно-изходна операция (WR#=1 и RD#=1) вътрешният буфер е в състояние на висок импеданс.

Таймерите се програмират индивидуално чрез запис на определена информация в съответните броячи и управляващия регистър. Функцията на отделните битове на управляващия регистър е следната:

| бит | състояние | функция |
|---|---|---|
| 7,6 | | Избор на брояч |
| | 00 | избор на брояч 0 |
| | 01 | избор на брояч 1 |
| | 10 | избор на брояч 2 |
| | 11 | не се използва |
| 5,4 | | последователност на зареждане на броячите |
| | 00 | запомняне на текущата стойност на брояча |
| | 01 | четене/запис само на мл. байт на брояча |
| | 10 | четене/запис само на ст. байт на брояча |
| | 11 | четене/запис на мл. байт, след това на ст. байт на брояча |
| 3,2,1 | | Избор на режима на работа |
| | 000 | режим 0 - генериране на импулс |
| | 001 | режим 1 - чакащ мултивибратор |
| | X10 | режим 2 - генератор на правоъгълни импулси с програмируема честота F и коефициент на запълване |
| | X11 | режим 3 - генератор на правоъгълни импулси с коефициент на запълване 0.5 |
| | 100 | режим 4 - програмно управляем стробиращ импулс |
| | 101 | режим 5 - апаратно управляем стробиращ импулс |
| 0 | | Конфигурация на брояча |
| | 0 | двоичен 16 разреден брояч до 65536 |
| | 1 | десетичен брояч, 4 разреден до 10000 |

Всеки от броячите е реверсивен и може да бъде настроен като двоичен или десетичен в един от шест различни работни режима. Четенето на текущото състояние на броячите е възможно по два начина:

- посредством обикновена инструкция за четене от входно-изходното пространство (*in*), като адресът се определя от базовия адрес на I8253, по който се изработва CS# и номера на брояча от разредите A1 и A0. За да се гарантира достоверна информация е необходимо спиране на съответния таймер със сигнала GATE (GATE=0);
- четене без спиране на таймера. За тази цел в I8253 е предвиден специален вътрешен регистър, който приема съдържанието на зададения брояч и осигурява стабилни данни за четене по системната шина. Записът на текущата стойност на брояч във вътрешния регистър се задава чрез нулиране на

разреди 4 и 5 на управляващия регистър и определяне номера на брояча с разреди 7 и 6. Съдържанието на вътрешния регистър се прочита чрез инструкция за четене от избрания брояч.

### 3.2. Основни режими на работа на таймерите

Режим 0 се използва за генериране на положителен импулс след нулиране на брояча на съответния таймер. Зареждането на CR води до нулиране на неговия изход OUT, който преминава в 1 след N тактови импулса при условие GATE = 1. N е стойността, заредена в съответния брояч.

В режим 1 таймерът работи като чакащ мултивибратор. Изходът OUT приема ниско ниво през такта, следващ предния фронт на сигнала GATE и се вдига при нулирането на съответния брояч. Следователно на изхода OUT се генерира отрицателен импулс с продължителност N тактови импулса, където N е заредената стойност в брояча. Зареждането на нова стойност в брояча не влияе на продължителността на текущия импулс. Чакащият мултивибратор се запуска при всеки положителен фронт на GATE, дори ако импулсът не е завършил.

В режим 2 таймерът работи като делител на входната честота на N. Изходният сигнал OUT е с ниско ниво в продължение на един период на тактовия импулс. Периодът на повторение на изходните импулси се определя от предварително зареденото съдържание на брояча N. Ново зареждане на брояча в интервала от време между два изходни импулса не изменя текущия цикъл на броене, а се отразява в следващите цикли. Синхронизацията на броенето може да се извърши по два начина- програмно или чрез сигнала GATE. При програмната синхронизация GATE трябва да бъде с високо ниво, а броенето се стартира чрез зареждането на брояча. При втория начин изходният сигнал се поддържа с високо ниво, докато GATE е в ниско ниво. Стартирането на броенето се извършва чрез активиране на GATE (GATE=1).

В режим 3 таймерът работи като генератор на правоъгълни импулси с коефициент на запълване 0.5. Изходният сигнал е с високо ниво до отброяване на половината от първоначално заредената стойност в брояча (при четно число), след което до завършване на броенето се поддържа ниско ниво. При достигане на граничната стойност на брояча нивото на изходния сигнал се променя, броячът се зарежда отново с първоначалната стойност и цикълът се повтаря. Ако първоначалната стойност е нечетно число N, във всеки цикъл изходният сигнал ще бъде с високо ниво в продължение на (N+1)/2 такта, а с високо ниво - (N-1)/2 такта. Синхронизацията се осъществява както при режим 2.

Режим 4 позволява генерирането на програмно управляем стробиращ импулс с активно ниско ниво. Продължителността на строба, който се генерира при достигане на граничната стойност на брояча, е равна на продължителността на тактовия импулс.

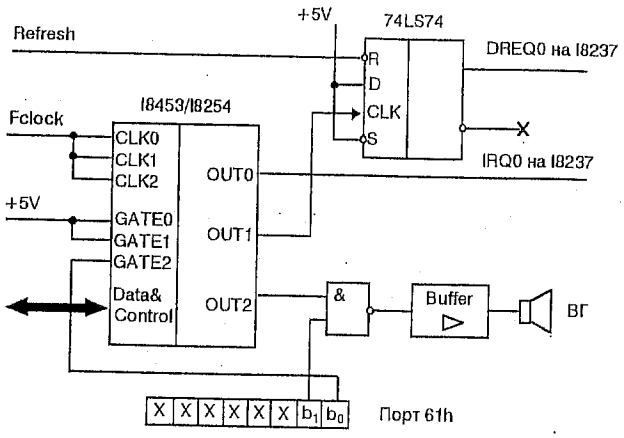
Режим 5 е аналогичен на режим 4, но при него е възможно ново запускане на таймера при всеки положителен фронт на GATE.

### 3.3. Използване на таймер в IBM PC XT/AT

В персоналните компютри схемата I8253/I8254 заема базов адрес 040h. За трите таймера се използва тактов сигнал с честота 1,193 MHz. Таймер 0 и таймер 1 се програмират за работа в режим 2. Таймер 0 е системен таймер с общо предназначение. Той осигурява изграждането на часовник за реално време и се използва от операционните системи. При всеки положителен фронт на изхода му OUT се подава заявка за прекъсване към линията IRQ0 на контролера на прекъсванията I8259A. Таймерът се настройва за генериране на заявка за прекъсване на всеки 54,925 ms (с честота 18,2 Hz). Програмата за обслужване на прекъсването обновява часовника за реално време и управлява спирането на двигателя на флопидисковото устройство.

Таймер 1 изработва сигнал с период 15,12 микросекунди, чрез който се генерира заявка към канал 0 на DMA I8237 за опресняване на динамичната памет.

Таймер 2 работи в режим 3 и се използва за управление на вградения високоговорител. Сигналът към високоговорителя се управлява чрез зареждане на подходяща информация в изходния регистър на порт PB на I8255A (адрес 61h). Разряд 0 на този регистър е свързан към управляващия вход на таймер 2, а разред 1 определя предаването или блокирането на изходния сигнал към високоговорителя (фиг.11.5). Благодарение на тези два разреда на порт PB е възможна модулация на сигнала към високоговорителя, с което става възможно управлението на силата на звука.




Фиг. 11.5. Свързване на таймера в IBM PC XT/AT

**4. Интегрирани таймери**

В APIC контролерите на микропроцесорите от фамилията P6 е включен 32-разреден програмируем таймер за използване в локалния процесор. Освен регистъра брояч (адрес FEE00390h), таймерът съдържа регистър за начална стойност (адрес FEE00380h) и конфигурационен регистър (адрес FEE003E0h). Предназначението на таймера е да генерира прекъсвания при нулиране на брояча. Таймерът може да работи като генератор на единични прекъсвания или в периодичен режим. Периодът се задава в брой тактове на броене N и се записва в регистъра за начална стойност. Прекъсванията от таймера могат да бъдат конфигурирани през локалната векторна таблица LVT на APIC с различен вектор. Честотата на броене е променлива и се задава чрез конфигурационния регистър. Тя се определя от честотата на тактовия сигнал на шината на МП, разделена на съдържанието на конфигурационния регистър C, което може да бъде: 1, 2, 4, 8, 16, 32, 64 и 128. Честотата на генериране на прекъсване се определя като $CLK / C / N$, където CLK е честотата на такта на шината. Таймерът в APIC може да се използва от системните програми за отработване на различни интервали от време.

В състава на PIX4 е включен таймерен блок, функционално съвместим с I8254, съдържащ три брояча. Те заемат същите адреси във входно изходното адресно пространство - 040h, 041h и 042h. Предназначението им е аналогично на разгледаното в т. 4.3. Включеният в таймерния блок управляващ регистър (адрес 043h) аналогично служи за избор на брояч, задаване на режим на работа, ред на обръщение към байтовете на брояча, формат на броене (двоичен или десетичен). Допълнително е въведена команда за обратно четене (Read Back Command - RBC), която се задава чрез запис на 11 в разреди 7 и 6 на управляващия регистър. Тази команда се използва за формиране на информация за избран брояч или броячи, включваща текущото значение на брояча, режима на работа, текущото значение на изхода OUT. Формираната информация се получава в МП чрез инструкция за четене от адреса на избрания брояч или броячи. Командата RBC променя интерпретацията на останалите разреди на управляващия регистър, а именно:

| бит | състояние | функция |
|---|---|---|
| 7,6 | 11 | команда RBC |
| 5 | 0 | фиксиране на текущото значение на избрания брояч |
| | 1 | без значение |
| 4 | 0 | фиксиране на състоянието на избрания брояч |
| | 1 | без значение |
| 3 | 1 | избор на брояч 2 |
| 2 | 1 | избор на брояч 1 |
| 1 | 1 | избор на брояч 0 |
| 0 | 0 | без значение |

**Пример:**

Примерът демонстрира фрагмент от програма за управление на високоговорителя на IBM PC. Високоговорителят може да бъде програмиран за възпроизвеждане на различни тонове и мелодии. За да се възпроизведе определен тон, входният сигнал трябва да има определена честота и продължителност, които в примерния фрагмент са зададени с **freq** и **time**.

```
    mov   al, 0b6h    ; препрограмиране на Timer 2
    out   43h, al     ;
    mov   eax, 1234DCh;
    div   freq        ; зарежда коефициента на деление
    out   42h, al     ;
    mov   al, ah      ;
    out   42h, al     ;

    in    al, 61h     ;
    mov   ah, al      ; разрешава високоговорителя
    or    al, 3 ;
    out   61h, al     ;

    mov   ecx, time   ;
lab:
    loop  lab         ; цикъл за изчакване

    mov   al, ah      ; изключване на високоговорителя
    out   61h, al     ;
```

За конфигуриране на таймер 2 към порт 043H (управляващ регистър на I8253) се изпраща стойността 0B6h (10110110B). Това задава режим 3 (делител на честота) за таймер 2 и последователност на обръщение към него - младши след това старши байт. Изчисленият коефициент на делене, равен на тактовата честота на брояча 1193180 (1234DCh) Hz разделена на необходимата честота на сигнала **freq**, се зарежда в брояча на таймер 2 по адрес 42h. Работата на таймер 2 и високоговорителя се разрешава чрез установяване в 1 на двата младши разреда на порт 61h. При това значението на останалите разреди на порт 61h не трябва да бъде променяно. Периодът от време, през който се разрешава работата на високоговорителя се изработва чрез програмен цикъл. В този случай времето за изпълнение на цикъла зависи от броя тактове за изпълнение на инструкциите в цикъла и тактовата честота на процесора.

**Въпроси и задачи:**

1. Програмирайте таймер 2 за генерация на правоъгълни импулси с честота 1 KHz и коефициент на запълване 0,5. Напишете програма за периодично излъчване на звук през 1 s. Наблюдавайте с логическия анализатор времедиаграмата при конфигуриране на таймера и сигнала на неговия изход.

2. Напишете програма за възпроизвеждане на нотите от първа октава, ако честотата им в Hz е следната: до - 261,7; ре - 293,7; ми - 329,6; фа - 349,2; сол - 392,0; ла - 440,0; си - 493,9.

3. Решете задача 1 като използвате таймер 0 като източник на еталонно времезакъснение.

# 12. Последователен програмируем интерфейс

В упражнението се разглеждат средства за реализация на опростен интерфейс с входно-изходни устройства, обменящи информация в последователна форма.

## 1. Предназначение на последователния интерфейс

Последователният интерфейс служи за обмен на данни между МП и външни устройства в последователен (сериен) формат: мишки, модеми, клавиатури, терминали и др. Основните функции, които трябва да се изпълняват при реализация на последователен интерфейс са: преобразуване на данните от паралелен в последователен формат и обратно; предаване и приемане; буфериране; разпознаване на състоянието на комуникационната линия (предаване или покой); разпознаване на битовете за начало (стартов бит) и край (стопови битове) на байт; контрол на данните (четност или нечетност); поддържане на различни честоти на линията. Предаването на данните по комуникационната линия може да бъде синхронно или асинхронно. За по-голяма гъвкавост и съвместимост с различни типове ВУ параметрите на интерфейса и режимите на изпълнение на неговите функции се настройват програмно.

МП обикновено обменя три типа информация с последователния интерфейс, за които се необходими три регистъра:

- регистър за данни, който буферира данните за предаване към или приемане от МП;
- регистър за състоянието, отразяващ готовността на регистъра за данни за предаване или за приемане на информация. През този регистър МП получава и друга информация за състоянието на интерфейса: грешки при обмен, състояние на линията и др.;
- управляващ регистър, чрез който МП задава режима на работа на интерфейса и параметрите на операциите с него и стартира обмена и др.

В персоналните компютри (ПК) последователен обмен с ВУ се изпълнява чрез токов кръг, стандартизиран асинхронен интерфейс RS-232C или стандартен интерфейс "универсална последователна шина" (Universal Serial Bus - USB). Управлението на интерфейса може да се реализира посредством специализирани интегрални схеми I8251A, I82510, I8250, I82450 и др., а за USB - многофункционални периферни схеми 82371AB (PIIX4), набори ИС (ShipSet) като I430TX, I440LX, микроконтролери 8x930Ax и др. Сигналните вериги на интерфейса RS-232C са:

- TxD - предавани данни;
- RxD - приемани данни;
- RTS# - заявка за предаване от ПК към ВУ;
- CTS #- сигнал към ПК, че приемникът е готов;
- DSR# - готовност на ПУ за предаване;
- DTR# - сигнал за установяване на връзка с ПУ;
- CD# - откриване на носещата честота от модема;
- RI# - използва се за работа по комутируема линия и е признак за повикване.

Сигналите CD# и RI# се използват само при работа с модем.

## 2. Програмируем последователен интерфейс I8251A

Интегралната схема I8251A служи за опростена реализация на последователен интерфейс между МП и външни устройства. Схемата може да се използва, както за синхронно, така и за асинхронно предаване. Чрез нея МП може да изпълнява програмно управляван обмен или обмен с прекъсване. Параметрите на обмена, като ширина на полето данни за представяне на един знак, скорост на обмен, протокол на опознаване и др., се задават програмно чрез запис в I8251A на два байта - байт за избор на режим и байт за управление.

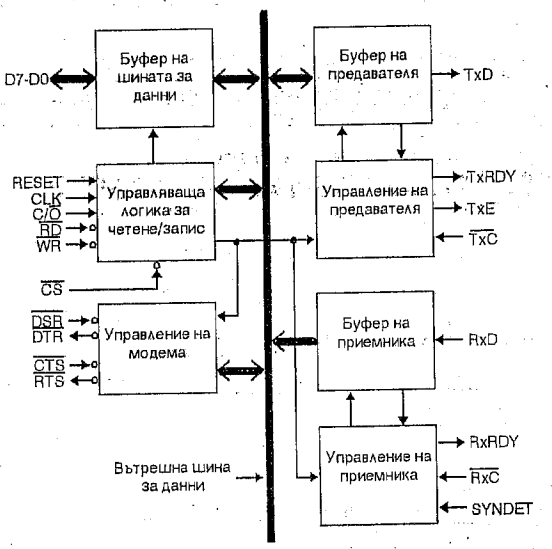
### 2.1. Вътрешна структура на I8251A

В структурата на интерфейса са включени пет 8 разредни регистъра: два буферни регистъра съответно на предаваните и приемани данни, управляващ регистър, регистър на състоянието и регистър на режима. Предаваните данни през данновите входове D7-D0 се помествят паралелно в буферния регистър за предаване и се извеждат в последователен вид през изхода TxD (фиг.12.1.).

Данните, пристигащи в последователна форма по вход RxD, се помествам в приемащ буфер, откъдето в паралелен вид се четат от изходите D7-D0. На вход TxC# се подават синхросигналите за предаване, а на RxC# - синхросигналите за приемане. Скоростта на предаване на данните може да бъде 1/16 или 1/64 от честота на синхросигналите, която се задава от режима на работа. На практика се използват стандартизираните честоти: 300, 600, 1200, 2400, 4800, 9600, 19200 бит/с.

Сигналът TxRDY се активира след предаването на последния бит от буфера по линията TxD, а сигналът RxRDY - след приемането на последния бит от текущия кадър по линията RxD. При високо ниво на тези сигнали МП може да осъществява съответно запис или четене на следващия байт данни от интерфейса. Когато обменът е с прекъсване, TxRDY и RxRDY се използват като сигнали за прекъсване на МП. При програмно управляемо предаване значенията на тези сигнали могат да се тестват от регистъра на състояние (битове 0 и 1 съответно) и да се интерпретират като флагове за готовност. Сигналът TxEMPTY означава, че предавателят е свободен (TxEMPTY=1) или зает с изпращане на данни (TxEMPTY=0).

Нормално състоянието на линията между предаванията на два кадъра е ниво логическа единица. Непрекъснато предаване на нули между два кадъра данни съответства на състояние на прекъсване (например натиснат клавиш BREAK) и се открива от схемата за управление на приемането, която активира сигнала BRKDET.




Фиг. 12.1. Структура на интерфейса за последователна връзка I8251A.

В последователния интерфейс е включена и схема за управление на модем. С тази схема са свързани 4 сигнала:DSR#, CTS#, DTR#, RTS#, които могат да се използват като едноименните сигнали на стандартния интерфейс RS232C. Значението на входните сигнали DSR# и CTS# се тества от МП чрез четене на регистъра на състоянието, а значението на изходните сигнали DTR# и RTS# може да бъде зададено чрез запис в управляващия регистър.

### 2.2. Управление на записа и четенето

Схемата за запис и четене управлява достъпа до вътрешните регистри на I8251A в зависимост от значението на сигналите CS#, C/D#, RD# и WR# съгласно следната схема:

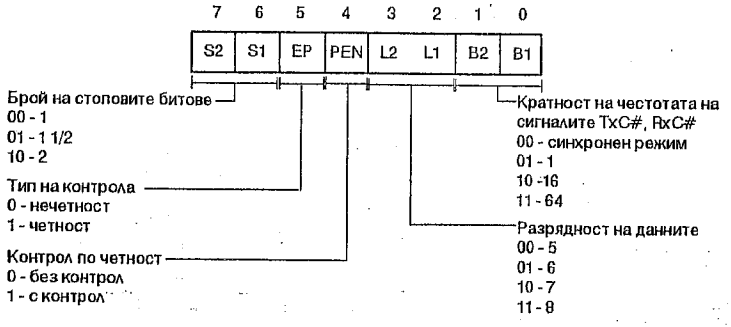
| CS# | C/D# | RD# | WR# | действие |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | четене на приетите данни |
| 0 | 0 | 1 | 0 | запис на предаваните данни |
| 0 | 1 | 0 | 1 | четене на регистъра на състояние |
| 0 | 1 | 1 | 0 | запис в регистрите за режим и управление |

Последната комбинация се приема като запис в регистъра за режим, само ако следва след сигнал RESET или след команда за начално установяване от управляващия регистър. В противен случай се

извършва запис в регистъра за управление.

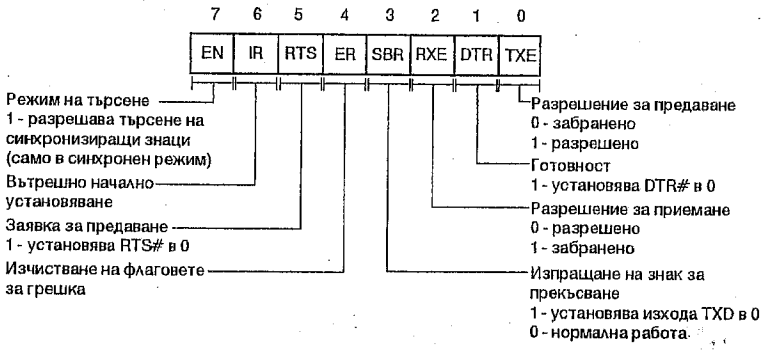
### 2.3. Програмиране

Регистърът за задаване на режима служи за задаване на параметрите на предавания кадър данни - размер на данните, брой стопови битове, наличие и тип на контролен бит, честота на синхросигналите и има структурата от фиг.12.2.




Фиг. 12.2. Регистър на режима

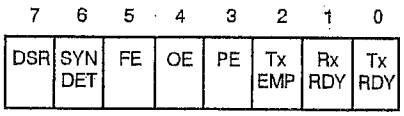
Регистърът за управление задава значение на изходите към модем, изчиства флаговете за грешка, задава подаване на сигнал BREAK. Структурата му е показана на фиг.12.3.




Фиг. 12.3. Регистър за управление.

След обмен на данни състоянието на I8251A се отразява в регистъра на състоянието (фиг.12.4). Освен състоянието на сигналите за готовност за предаване и приемане TxRDY и RxRDY, на входните сигнали от модема DSR# и CTS#, на сигнала TxEMPTY, в регистъра на състоянието се отразяват и следните състояния на грешка:

- PE - грешка по четност. Този флаг се установява, когато се открие грешка по четност. Той се изчиства чрез бита ER на управляващия регистър;
- OE - грешка от припокриване. Флагът се вдига, когато на входа на приемника постъпи знак, без предходният приет знак да е бил прочетен от процесора. Флагът OE също се изчиства чрез бита ER на управляващия регистър;
- FE - грешка на формата. Този флаг се вдига, когато не е открит валиден стопов бит в края на текущия кадър. Изчиства се чрез бита ER на управляващия регистър;
- SYNDET - индицира открит синхронизиращ знак. Използва се само в синхронен режим.

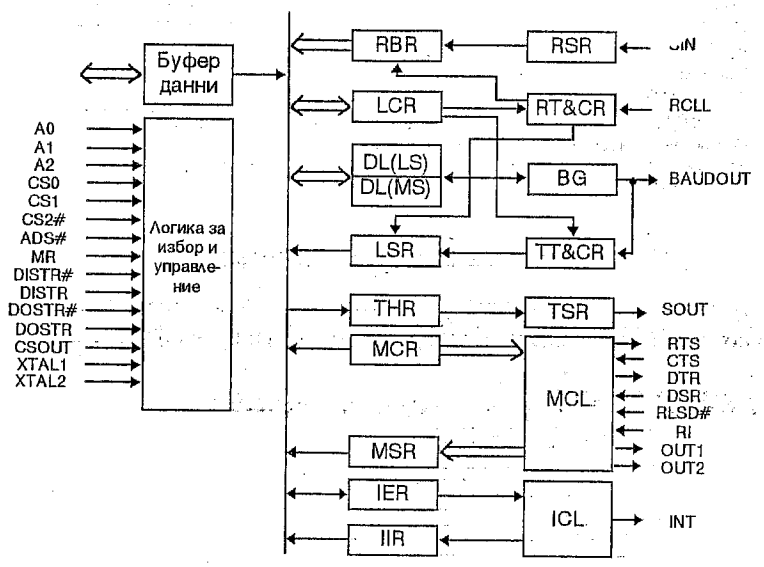



Фиг. 12.4. Регистър на състоянието

## 3. Последователен интерфейсен адаптер I8250

### 3.1. Вътрешна структура на I8250

Специализираната ИС I8250 притежава аналогични на I8251 функционални възможности: независимост на приемната и предавателната част (пълен дуплекс); програмно задаване на параметрите на обмен - формат на данните, брой стопови битове, наличие или не на контрол по четност/нечетност; програмно задаване на скоростта на обмен (от 50bits/sec до 9600 бит/с.); генериране на заявка за прекъсване при една от следните ситуации: наличие на приет байт данни, празен предавателен буфер, грешка по линията, промяна на състоянието на модема; самодиагностика чрез вътрешни обратни връзки.




Фиг. 12.5. Структура на I8250

Структурата на последователния интерфейс I8250 е показана на фиг.12.5. Адресацията на програмно достъпните регистри се извършва по входните сигнали CS0, CS1, CS2# (сигнали за избор на чип), адресните сигнали A0, A1, A2 (отместване от базовия адрес на чипа) и значението на старшия бит на регистъра за управление на параметрите на линията LCR[7]. Адресирането на съответните регистри по отношение на базовия адрес е приведено в таблица 12.1.

**Таблица 12.1**

| b7(LCR) | Отместван | Регистър | Означение |
|---|---|---|---|
| 0 | 0H | Приемен буфер за данни (четене) | RBR |
| 0 | 0H | Предавателен буфер за данни (запис) | THR |
| 0 | 1H | Разрешение на прекъсванията | IER |
| x | 2H | Идентифициране на прекъсванията | IIR |
| x | 3H | Управление на линията | LCR |
| x | 4H | Управление на модема | MCR |
| x | 5H | Състояние на линията | LSR |
| x | 6H | Състояние на модема | MSR |
| 1 | 0H | Фактор за деление на честотата (мл.байт) | DLL |
| 1 | 1H | Фактор за деление на честотата (ст.байт) | DLM |

### 3.2. Програмно достъпни регистри

#### 3.2.1.Регистър за управление на линията (LCR)

В този регистър се задават параметрите на асинхронната връзка. Предназначението на отделните битове е следното:

**Битове 0,1** - дължина на обменяни символи

| Бит 0 | Бит 1 | Дължина |
|---|---|---|
| 0 | 1 | 6 бита |
| 1 | 0 | 7 бита |
| 1 | 1 | 8 бита |

**Бит 2** - брой стопови битове (стойност **0** на този бит задава един стопов бит, а **1** - два стопови бита);

**Бит 3** - включва (при стойност 1) и изключва (при стойност **0**) контрола по четност/нечетност;

**Бит 4** - задава тип на контрола (**0** - по нечетност, **1** - по четност);

**Бит 5** - при включен контрол задава стойността на бита за контрол (**0** или **1**);

**Бит 6** - при стойност 1 установява линията в състояние **Break**;

**Бит 7** - допълнително адресиране на регистри (стойност **1** осигурява достъп до регистрите за фактора за деление на честотата, а **0** - достъп до приемо-предавателните регистри);

#### 3.2.2.Регистри за фактора за деление на честотата (DLM, DLL)

Регистрите съдържат 16-разредно число, показващо на колко се дели тактовата честота на чипа I8250 (1.8432 MHz), за да се получи честотата на стробиране. Тя от своя страна е 16 пъти по-висока от честотата, с която се обменят данните. Факторът на деление (ФД) за всяка от необходимите скорости (НС) може да се изчисли по формулата:

$$ФД = 1.84332.10\ 6/\ 16 * НС$$

#### 3.2.3. Регистър за състояние на линията (LSR)

Регистърът съдържа информация за състоянието на комуникационните линии в даден момент. Предназначението на отделните битове е следното:

**Бит 0** - индикатор за приети данни. Установява се в 1 при прием пореден байт данни в буферния регистър а в **0** - при четене от приемния буферен регистър;

**Бит 1** - индикатор за препълване (**1** - прием пореден байт данни преди прочитане на съдържанието на приемния буферен регистър. Нулира се при четене на съдържанието на LSR);

**Бит 2** - грешка при контрол по четност/нечетност. Стойност 1 индицира грешка. Нулира се при четене съдържанието на LSR;

**Бит 3** - грешка при приемане. Стойност 1 индицира липса на стоп-бит. Нулира се при четене съдържанието на LSR;

**Бит 4** - индикатор за състояние **Break** по линията. Установява се в **1** при състояние Break, а се нулира се при четене съдържанието на LSR;

**Бит 5** - готовност за предаване (**1** - празен предавателен буферен регистър, **0** - зареден предавателен регистър);

**Бит 6** - предадени данни (**1** - предадено съдържание на преместващия предавателен регистър. Нулира се при запис на поредния символ от предавателния буферен регистър в преместващия предавателен регистър;

**Бит 7** - винаги има стойност **0**.

#### 3.2.4. Регистър за идентификация на прекъсването (IIR)

Чипът I8250 може да издава само една заявка за прекъсване към контролера на прекъсванията I8259, но причините, които я предизвикват могат да бъдат различни. Поради това в него са предвидени схеми за определяне приоритета на прекъсванията, като информацията за източника на прекъсване се съдържа в регистъра IIR (табл.12.2). Прекъсванията са групирани в четири приоритетни нива, като най-висок е приоритетът на ниво 1. Заявките за прекъсване се обслужват съгласно техния приоритет.

**Таблица 12.2**

| Стойност на битовете в IIR | | | | Източник на прекъсването и условия за снемане на заявката | |
|---|---|---|---|---|---|
| б2 | б1 | б0 | Ниво на приоритет | Тип прекъсване | Източник на прекъсване |
| 0 | 0 | 1 | - | Няма прекъсване | - |
| 1 | 1 | 0 | 1 | Състояние на линията при приемане. Заявката се снема при четене на LSR | Прекъсване по препълване (бит 1 на LSR) |
| | | | | | Грешка при контрол по четност (бит 2 на LSR) |
| | | | | | Грешка при приемане (бит 3 на LSR) |
| | | | | | Състояние Break (бит 4 на LSR) |
| 1 | 0 | 0 | 2 | Наличие на приет байт данни. Заявката се снема при четене на RBR | Приет байт данни (бит 0 на LSR) |
| 0 | 1 | 0 | 3 | Празен буферен регистър за предаване. Завката се снема при запис в THR или четене на IIR | Готовност за предаване (бит 5 на LSR) |
| 0 | 0 | 0 | 4 | Състояние на модема. Заявката се снема при четене на MSR | Промяна в състоянието на модема |

### 3.2.5. Регистър за разрешение на прекъсванията (IER)

Този регистър позволява маскиране на видовете прекъсвания. Предназначението на отделните битове е следното:

**Бит 0** - при стойност 1 разрешава прекъсване при наличие на приети данни;
**Бит 1** - при стойност 1 разрешава прекъсване при празен буферен регистър за предаване;
**Бит 2** - при стойност 1 разрешава прекъсване при възникване на грешки по линията;
**Бит 3** - при стойност 1 разрешава прекъсване при промяна в състоянието на модема.
**Битове 4,5,6,7** - не се използват.

### 3.2.6. Регистър за управление на модема (MCR)

Чрез него се осъществява управлението на веригите към модема, към съответното периферно устройство или към друг компютър. Предназначението на отделните битове е следното:

**Бит 0** - при стойност 1 се активира изводът DTR на I8250;
**Бит 1** - при стойност 1 се активира изводът RTS на I8250;
**Бит 2** - при стойност 1 се активира изводът OUT1 на I8250;
**Бит 3** - при стойност 1 се активира изводът OUT2 на I8250;
**Бит 4** - при стойност 1 се осигурява вътрешна обратна връзка за тестване на последователния адаптер;
**Битове 5,6,7** - не се използват.

Нормално в персоналните компютри IBM AT има два основни комуникационни асинхронни адаптера (ПА), като BIOS позволява включване на още два допълнителни ПА. Всеки ПА заема 8 последователни адреса във входно/изходното пространство на процесора, като първият се приема за базов:

3F8H .. 3FFH - портове за Com1;
2F8H .. 2FFH - за Com2;
3E8H .. 3EFH - за Com3;
2E8H .. 2EFH - за Com4.

Генерираните от ПА заявки за прекъсване се изпращат чрез изхода OUT2 към системния контролер на прекъсванията I8259, като съответно Com1 е свързан към IRQ4, а Com2 - към IRQ3. Векторите на прекъсване са съответно 0Ch и 08h. Допълнителните ПА могат да се свържат към други входове на контролера на прекъсванията в зависимост от конфигурацията на системата.

4. Обмен на информация чрез схемите за последователен интерфейс

Обменът на данни може да се реализира като програмно управляем или с използване на механизма на прекъсване. При програмно управляемия обмен преди предаването или приемането на поредния байт програмно се проверява съотвтното условие - празен буфер THR (установен бит 5 в регистъра LSR) или приети данни (установен бит 0 на LSR).

За обмен на информация чрез механизма на прекъсване трябва да се инсталира процедура, обработваща прекъсванията от ПА, в зависимост от техния тип (приети данни, грешка по линията и др.). Заявката за прекъсване може да е генерирана и вследствие на едновременното настъпване на няколко събития (напр. приети данни и грешка от контрол по четност). Необходимо е инсталираната процедура да обработва всичките ситуации, в зависимост от съдържанието на регистъра за идентификация на прекъсванията IIR.

**Въпроси и задачи:**

1. Напишете процедура, инициализираща ПА на COM порта на вашия компютър със следните параметри: дължина на символа 8 бита; един стопов бит; контрол по нечетност; забранено генериране на прекъсвания. Проверете правилността на установените параметри, като прочетете и тествате значението на съответните регистри.

2. Напишете програми, реализиращи обмен на информация чрез връзка между ПА на два различни компютъра. Инициализирайте ПА на двата компютъра с еднакви параметри. Програмата, изпълнявана на единия компютър, приема от клавиатурата символ при натискане на клавиш и го изпраща през ПА. Програмата, изпълнявана на втория компютър, приема от съответния свързан ПА символ и да го извежда на екран. Обменът завършва при предаване/приемане на символа, определен като символ за край.

3. Свържете логически анализатор към нужните изводи на ПА и снемете времедиаграмите при предаване и приемане на символ от страна на МП и последователната линия. Изводите на ПА са описани във файла chips.doc.

4. Напишете програми, реализиращи обмен на информация чрез връзка между ПА на два различни компютъра посредством механизма на прекъсванията. Програмата, изпълнявана на единия компютър, да изпраща през ПА символ при натискане на клавиш. Програмата, изпълнявана на втория компютър, да инсталира процедура, обработваща прекъсване от съответния свързан ПА, да приема символи от него и да ги извежда на екран.

5. Решете горната задача в защитен режим. Допълнете таблицата за прекъсване с нужните дескриптори за обръщение към процедури, обработващи прекъсване при изпратени и приети данни.

# 13. Пряк достъп до паметта

Целта на упражнението е запознаване с реализацията на един от методите за повишаване на ефективността за изпълнение на входно-изходните операции - пряк достъп до паметта. Разглежда се контролерът за пряк достъп до паметта I8237A и включването му в микропроцесорни системи.

## 1. Пряк достъп до паметта

Каналът за пряк достъп до паметта (DMA) позволява да се обменят данни между външните входно-изходни устройства (ВУ) и паметта без намесата на процесора. Процесорът се грижи само да стартира операцията за обмен, като зададе необходимата за протичането ѝ информация и да регистрира нейното завършване. След стартиране на операцията процесорът и DMA работят паралелно, като всеки от тях изпълнява различна задача. За да изпълнява възложените му функции по обмена на данните, DMA трябва да има възможност за заемане и управление на шината за връзка с паметта, която нормално се владее от процесора.

## 2. Контролер за пряк достъп до паметта

Най-използваният DMA контролер в микропроцесорните системи на базата на процесори I8086 и I286 е програмируемият DMA I8237A. В компютрите на базата на 32 разредните процесори Intel се използват по-бързодействащи и производителни DMA контролери, включени в състава на интегрирани периферни чипове, стилът на работа на които е съвместим с I8237A.

Контролерът I8237A има 4 независими канала, посредством които могат да се изпълняват паралелно до 4 различни входно-изходни операции с различни устройства. С каскадно свързване на няколко контролера броят на каналите може да се увеличи. Всеки канал е индивидуално програмирамируем с възможност за адресация на памет до 64K.

### 2.1. Вътрешна структура

В структурната схема на DMA контролера (фиг.13.1) влизат три основни блока за управление:

• синхронизация и управление - генерира вътрешните и външни сигнали за управление. Този блок приема и изработва управляващите сигнали за интерфейс с микропроцесора, паметта и входно-изходните устройства. Сигналите MEMR# (четене) MEMW# (запис) задават посоката на обмен на данни с паметта, а сигналите IOR# (четене) и IOW# (запис) - посоката на обмен с входно-изходните устройства. Сигналът AEN стробира валиден адрес на изходните адресни линии A7-A0, а ADSTB може да бъде използван за синхронизация на адреса. Сигналът READY отразява готовността на паметта за завършване на обмена и служи за въвеждане на празни тактове при работа с бавни памети или входно-изходни устройства. DMA може да бъде избран от микропроцесора за програмиране или тестване на състоянието чрез сигнала CS#;

• управление на команди - декодира различните команди, изпратени от микропроцесора и определя режима на обслужване на заявките;

• блок за управление на приоритета - разрешава приоритетното обслужване на едновременно възникнали заявки към DMA. Заявките от външните устройства за DMA цикъл постъпват асинхронно по входовете DREQ0-DREQ3 към съответния номер канал. Приоритетът на каналите може да бъде фиксиран или ротационен. Със сигналите DACK0-DACK3 се уведомяват отделните устройства, че им е разрешен DMA цикъл. Заявка към микропроцесора за заемане на шината се издава с изходния сигнал HRQ, а със сигнала HLDA микропроцесорът предоставя управлението на шината на DMA контролера.

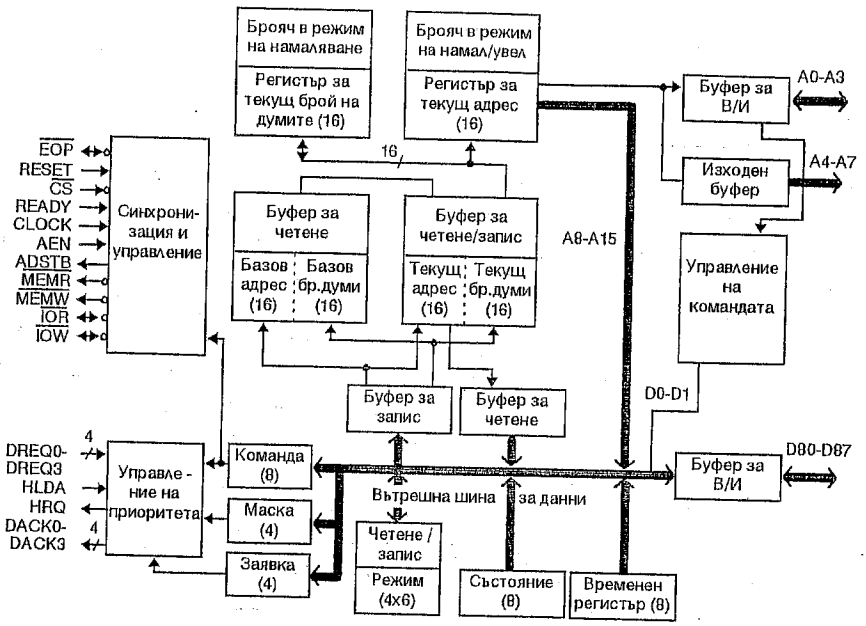
Всеки от четирите канала на DMA съдържа по четири 16 разредни регистъра:

• регистър за базов адрес. В него преди стартиране на операция процесорът зарежда началния адрес в паметта на обменяния чрез канала блок данни;

• базов брояч. Преди стартиране на входно-изходна операция в него се зарежда размерът на обменяния блок данни в брой думи. Регистърът за базов адрес и базовият брояч съдържат началните параметри на входно-изходната операция, управлявана от съответния канал, и не се променят при нейното изпълнение;

• регистър за текущ адрес. Съдържа адреса в паметта на следващия обменян елемент данни;

• текущ брояч. Съдържа броя думи, които остават да бъдат предадени до завършване на операцията.




Фиг. 13.1. Структурна схема на DMA контролер I8237A.

Регистърът за текущ адрес и текущият брояч съхраняват текущите параметри на входно-изходната операция. Текущите параметри на даден канал при неговото активиране се прехвърлят в регистъра за временен адрес и временния брояч, откъдето се използват за адресация на паметта през адресните изходи A7-A0 и откриване края на операцията съответно. След предаването на поредния елемент данни текущите параметри автоматично се актуализират, като адресът се увеличава (или намалява), а броячът се намалява с единица, и отново се връщат в регистрите на канала.

Вътрешните адресни регистри на каналите са 16-разредни и с тях могат да се адресират области с максимален размер 64K, наречени страници на паметта. Адресът на страницата , т.е старшата част на пълния адрес се задава чрез външни регистри на страниците. Всеки от каналите на контролера използва по един регистър на страници.

### 2.2. Функциониране на DMA контролера

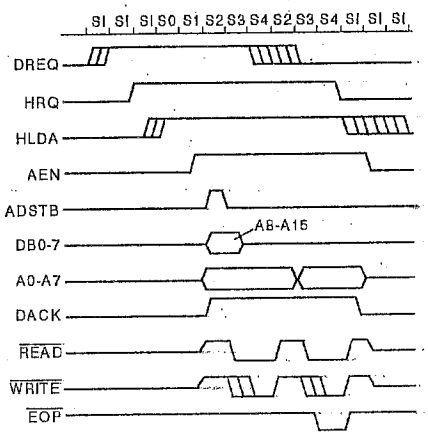
Контролерът изпълнява два основни цикъла - пасивен и активен. Всеки цикъл се състои от няколко различни състояния, заемащи по един пълен период на тактовия импулс CLK.

#### 2.2.1. Пасивен цикъл

Когато няма заявка за обмен каналът изпълнява пасивен цикъл, състоящ се от последователни състояния SI (фиг.13.2). В този цикъл се следят сигналите за заявка DREQ0-DREQ3 от ВУ и сигнала CS# за адресиране на вътрешните регистри от страна на МП. При активен CS# (ниско ниво) и неактивен HOLDA контролерът преминава в състояние на програмиране. МП адресира вътрешните регистри чрез адресните входове A3-A0, а изпълняваната с тях операция със сигналите IOR# (четене) и IOW# (запис). Данните се предават през данновите изводи - D7-D0. За избор на старшия или младши байт на 16-разредните регистри на каналите се използва вътрешен тригер, който генерира допълнителен адресен бит. В състояние на програмиране могат да бъдат изпълнявани няколко програмни команди, задавани като комбинация от значение на сигналите A3-A0, IOR# и IOW# - начално установяване, нулиране на регистъра на маските, нулиране на тригера за адресация на 16 разредните регистри.

### 2.2.2. Активен цикъл

Когато по някой от каналите се получи заявка за обслужване DREQi, контролерът издава към МП сигнала HRQ и влиза в активен цикъл - състояние S0 (фиг.13.2). Той остава в състояние S0 до получаване на разрешение от МП за ползване на шината чрез сигнала HLDA, след което преминава в първото работно състояние S1. В състояние S1 контролерът извежда върху адресната шина текущия адрес към паметта, заедно с управляващия сигнал AEN. Старшата част на 16-разредния адрес се извежда през изводите D7-D0, а младшата - A7-A0. Старшата част на адреса се запомня временно във външен регистър по сигнала ADSTB. Тъй като обменът е в последователни адреси, то при блоково предаване старшата част се извежда само веднъж на всеки 256 цикъла, т.е. следващите цикли ще се състоят само от състояния S2, S3, S4. В състояние S2 контролерът изпраща сигнала DACK към ВУ, чиято заявка за обмен е приета. Данните се предават директно между ВУ и паметта. За тази цел в състояния S3 и S4 контролерът изработва сигналите MEMW# и IOR# при запис от ВУ в паметта или MEMR# и IOW# при четене от паметта във ВУ, които трябва да са активни едновременно. В края на предаването контролерът снема HRQ, което предизвиква прекратяване на HLDA. Това осигурява между циклите на контролера поне един цикъл на МП за управление на системната шина.




Фиг. 13.2. Времедиаграма на работа на DMA

В активен цикъл контролерът работи в един от следните режими:

• единично предаване. В този режим след получаване на HLDA се извършва само едно предаване, след което шината на МП се освобождава. След всяко предаване се актуализират текущият адрес и текущият брояч. Изчерпването на брояча задава края на входно-изходната операция;

• предаване на блок. След получаване на HLDA се извършва серия предавания до изчерпване на брояча на съответния канал или до получаване на външен сигнал EOP# (край на предаването);

• предаване по заявка. След получаване на HLDA се извършва серия предавания до изчерпване на брояча на канала, до получаване на външен сигнал EOP# или снемане на DREQ. В този режим предаването може да продължи, докато се изчерпят данните на обслужваното ВУ;

• каскаден режим. Този режим се използва при каскадно свързани DMA контролери. Изводите HRQ и HLDA на допълнителния контролер се свързват с изводите DREQ и DACK на един от каналите на главния контролер. Главният контролер се грижи само за заемане на шината от МП, като управлението на предаването по нея се извършва от издалия заявка допълнителен контролер.

Във всеки от първите три активни режима са възможни три различни типа предавания - четене, запис и проверка. При запис данните се прехвърлят от ВУ към паметта чрез активни MEMW# и IOR#, а при четене - от паметта към ВУ чрез активни MEMR# и IOW#. Проверката е псевдопредаване, при което DMA генерира адреси и реагира на сигнал EOP, но не активира управляващите сигнали към паметта и ВУ.

Обикновено DMA контролерът извършва прехвърляне на данни от паметта към ВУ и обратно. Освен това той може да прехвърля блок данни от памет в памет, като използва канал 0 като източник и канал 1 като приемник на данните. Така могат да се прехвърлят блокове от данни и програми от едно място на паметта в друго с минимум усилия за програмиране и с минимален

разход на процесорно време. Прехвърлянето памет-памет се извършва в блоков режим и се инициира посредством програмно активиране на сигнала DREQ0 (за канал 0). След получаване на шината с HLDA контролерът чете байт данни от паметта по текущия адрес на канал 0 и го съхранява във вътрешния си буферен регистър. След това се активира канал 1 и по неговия текущ адрес данните от буферния регистър се записват в паметта. След нулиране на текущия брояч на канал 1 се генерира изходен сигнал EOP, сигнализиращ край на прехвърлянето. Канал 0 може да бъде програмиран да съхрани един и същи адрес до края на предаването. Това позволява във всички клетки на един блок да се запише едно и също съдържание.

Програмно чрез регистъра на режима може да се зададе режим на автоинициализация, при който след завършване на входно-изходна операция чрез даден канал текущите му параметри автоматично се установяват, равни на базовите му параметри. Автоинициализираният канал е способен да извърши DMA прехвърляне, веднага щом се активира неговият DREQ.

### 2.3. Програмиране на контролера

Работата на контролера се управлява с 8-разредния управляващ регистър (CR) . Той се програмира от МП и задава:

- разрешава (CR[0]=1) или забранява (CR[0]=0) обмен памет-памет;
- канал 0 съхранява (CR[1]=1) или не (CR[1]=0) един и същ адрес;
- разрешава (CR[2]=0) или забранява (CR[2]=1) контролера;
- фиксиран (CR[4]=0) или ротационен (CR[4]=1) приоритет на каналите;
- сигналите DREQ са с активно ниско (CR[6]=1) или високо (CR[6]=0) ниво;
- сигналите DACK са с активно ниско (CR[7]=1) или високо (CR[7]=0) ниво;

С всеки канал има свързан 6 разреден регистър на режима (MR). При запис в него се задава:

- номера на канала MR[0-1];
- тип на предаването MR[3-2]: 00 - проверка, 01 - запис, 10 - четене, 11 - невалидно предаване;
- разрешена MR[4] = 1 или забранена MR[4] = 0 автоинициализация;
- инкрементиране MR[5] = 0 или декрементиране MR[5] = 1 на текущия адрес след поредното предаване;
- режим на предаване MR[7-6]: 00 - по заявка, 01 - единично, 10 - блоково, 11 - каскадно.

Програмно може да се имитира заявка за обслужване (като сигнал DREQ) по някой от каналите чрез запис в неговия регистър за заявка (RR). Тази заявка е немаскируема. Форматът на информацията, която МП записва в RR е следният:

- номер на канала RR[1-0];
- вдига RR[2]=1 или снема RR[2]=0 бита за заявка.

Битът за заявка се снема или програмно или при възникване на условие за край на операцията (нулиране на текущия брояч, сигнал EOP#).

С регистъра на маските (RM) може да се маскира заявка за предаване по съответния канал. По този начин може програмно да се променя приоритетът на каналите. Форматът на информацията, която МП записва в MR е следният:

- номер на канала MR[1-0];
- установява MR[2]=1 или снема MR[2]=0 бита за маска.

Регистърът на състояние (SR) е достъпен само за четене от МП. Той съдържа информация за текущото състояние на контролера, която показва кой канал е достигнал края на операцията си и кой канал все още обработва заявка.

Отделните регистри вътрешно се адресират чрез следните сигнали:

| регистър | операция | сигнали | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | CS# | IOR# | IOW# | A3 | A2 | A1 | A0 |
| CR | запис | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| MR | запис | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| RR | запис | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| RM | запис | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| SR | четене | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| BDR | четене | 0 | 0 | 1 | 1 | 1 | 0 | 1 |

С BDR е обозначен буферният регистър за данни

С указаните сигнали се стартират и командите:

| CS# | IOR# | IOW# | A3 | A2 | A1 | A0 | команди |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | Нулиране на тригера указател на байтове |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | Начално установяване |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | Нулиране на регистъра на маските |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | Запис във всички разреди на регистъра на маските |

В персоналните компютри IBM PC XT/AT DMA контролерът заема адреси 000h-00Fh. Регистрите за страници за каналите се разполагат на адреси 080h-083h.

**Пример:**

Пример за конфигурация на канал 2 на DMA за четене на блок данни (512B) от дискетното устройство. Програмата е предназначена за работа в реален режим. Канал 2 се конфигурира чрез запис на 46h (01000110B) в регистъра за режим (адрес 00Bh). С този байт на режима се определя: номер на канала - 2, тип на обмена - четене, автоинициализация - забранена, модификация на адреса +1, режим на единичен обмен. Чрез запис по адрес 00Ch се стартира командата "Начално установяване". Етикетът BUF е име на областта, където трябва да се запишат данните. Абсолютният адрес на BUF в системата се определя от програмата, като се сумира отместването на етикета BUF към началния адрес на съдържащия го сегмент. Програмата записва 16-те младши бита на адреса в адресния регистър на канал 2 (адрес 004h), а старшите 4 бита - в регистъра на страницата (адрес 081h). Броят на байтовете се записва в брояча на канал 2 (адрес 005h). 16-битовите стойности на адреса и брояча се записват с по две последователни инструкции ***out***.

```
CODE    SEGMENT
        ASSUME      CS:CODE
        mov         al, 46H     ; изпращане на командата за режима
        out         00BH, al    ; на канал 2
        out         00CH, al    ; нулиране на тригера указател на байт
        mov         ax, cs      ; прочитане на адреса на текущия сегмент
        mov         cl, 4;
        rol         ax, cl      ;
        mov         ch, al      ; запомняне на 4-те старши бита в ch
        and         al, 0F0H    ; нулиране на 4-т4 младши бита
        add         ax, offset BUF ;добавяне на отместването на буфера
        jnc         next        ; ако е имало пренос 4-те старши
        inc         ch          ; бита се увеличават с 1
next:
        out         004H, al    ; запис на младшия байт на адреса
        mov         al, ah      ;
        out         004H, al    ; запис на старшия байт на адреса
        mov         al, ch      ;
        out         081H, al    ; запис на старшите битове в регистъра на страниците
        mov         ax, 511     ; стойност на брояча за един сектор
        out         005H, al    ; запис на младши байт на брояча
        mov         al, ah      ;
        out         005H, al    ; запис на старши байт на брояча
        mov         al, 2       ; разрешаване на командата за
        out         00AH, al    ; канал 2 на контролера

BUF:    DB          512 DUP (?); буфер на информацията, прочетена от дискетата
        CODE        ENDS
```

### 3. Интегрирани DMA контролери

DMA контролери се включват в съвременните интегрирани схеми за връзка с компютърна периферия. Например в набора схеми за управление и връзка с шината PCI (PCI chipset) в схемата 82371AB (PIIX4), реализираща мост PCI-ISA/IDE, са включени два DMA контролера тип 8237. Двата контролера вътрешно са каскадно свързани, така че осигуряват седем независимо програмируеми канала. Всеки от тези канали може да бъде независимо конфигуриран да осъществява DMA предаване за ISA или PCI базирани ВУ. Конфигурацията се извършва посредством регистър, включен в конфигурационното пространство на PIIX4, в един от следните конфигурационни режима:

• стандартен DMA канал за ISA шина, използващ стандартните сигнали за заявка и разрешение към канала DREQ и DACK;

• DMA стил PC/PCI. Този стил позволява PCI базирани ВУ да стартират DMA предаване чрез кодиране на информацията за заявка и разрешение към канала по PCI линиите REQ# и GNT#. В този стил могат да бъдат конфигуриран до три канала;

• разпределена DMA схема.

И към двете шини DMA може да работи като главно или подчинено устройство. В режим на главно устройство контролерът управлява шината, за да извърши необходимото DMA предаване с паметта. Като подчинено устройство предоставя възможност за адресиране на вътрешните си регистри от страна на МП.

Разпределената DMA (DDMA) схема позволява регистрите, свързани с индивидуален DMA канал, физически да се намират извън PIIX4 в други PCI базирани устройства или отделен контролер от тип 8237. Логиката на DDMA в PIIX4 се използва само, когато МП осъществява достъп до регистрите на канала, а прехвърлянето на данните се управлява от ВУ. За да се определи разположението на разпределените DMA регистри във входно-изходното адресно пространство, PIIX4 използва два регистъра от конфигурационната си памет. Първият регистър задава базовия адрес на регистрите, свързани с каналите 0-3, а вторият - базовия адрес на регистрите за каналите 5-7. Схемата DDMA се конфигурира от BIOS или друга конфигурационна програма. PIIX4 играе роля на посредник при обръщение на МП към регистрите на разпределения DMA канал. Например при четене от МП на някой от регистрите на DDMA се изпълнява следния алгоритъм:

• в отговор на стартирания цикъл четене от процесора PIIX4 въвежда повторение (PCI retry), с което го прекратява;

• издава заявка за заемане като главно устройство на PCI шината и при получаване на управлението й изпълнява един или няколко цикъла четене към ВУ или контролера 8237, съдържащи нужния регистър. PIIX4 определя адреса на нужния регистър на базата на няколко параметъра: базовия адрес от конфигурационния регистър, номера на канала и номера на 8237 регистъра;

• при повторение на цикъла от МП PIIX4 изпраща прочетените данни, представляващи съдържанието на адресирания 8237 регистър.

Възможността за DDMA дава богати възможности за гъвкавост при проектиране на системи с различни видове периферни устройства в това число и за разширение на вече изградени периферни системи.

Интересно приложение намира и каскадният режим на работа на каналите. Той може да бъде използван за арбитраж на достъп до ISA шината. Устройството, подключено към канал, който е програмиран в каскаден режим, издава заявка за заемане на шината посредством DREQ. Вътрешният арбитър на DMA активира DACK, с което предава управлението на шината, устройството става главно и може да изпълнява цикли по нея, докато не снеме DREQ.

В PCI базираните персонални компютри DMA каналите заемат съответно адреси: DMA1 - 0000h-000Fh, DMA2 - 00C0h-00DEh, а регистрите за страници - 0080h-008Fh.

**Въпроси и задачи:**

1. Свържете логически анализатор към изводите на DMA контролера в IBM PC AT на базата на I286 и снемете времедиаграмата на работа на канал 2, например при изпълнение на ДОС командата dir.

2. Свържете логически анализатор към изводите на DMA контролера и снемете времедиаграмата на работа на канал 0 при опресняване на паметта. Отбележете разликите в двете времедиаграми.

3. Напишете примерна програма за прехвърляне на данни от една област на паметта в друга посредством DMA. Възможен ли е такъв режим в IBM PC, където канал 0 се използва за опресняване на паметта?

4. В какво се състои разликата в работата на DMA за ISA и DMA за PCI.

5. Обосновете необходимостта от разпределената DMA схема, реализирана в PIIX4.

# 14. Микропроцесорни системи на базата на MC68000

Целта на упражнението е запознаване с архитектурните особености на базовия за фамилията MC680x0 микропроцесор - MC68000. Разглеждат се основните характеристики на системата инструкции, системата за прекъсване, организацията на шината и взаимодействието с външните компоненти. Практическите упражнения се провеждат на микропроцесорна система за проектиране MEX68KDM, построена на базата на MC68000.

## 1. Архитектурни особености на MC68000

Основните технически характеристики на микропроцесора MC68000 са:
- 32 разредни вътрешни регистри;
- 16 разредни вътрешни и външни даннови шини;
- 24 разреден адрес и 16/64 MB адресно пространство;
- два привилегировани режима на работа - потребителски и системен (супервайзор);
- вход-изход, вмъкнат в адресното пространство на паметта;
- основна тактова честота 8 MHz;
- 64 извода на интегралната схема.

### 1.1. Програмен модел

Програмният модел на микропроцесора се състои от следните регистри:

• осем 32-разредни регистри за данни (D0-D7). Данновите регистри могат да съдържат операнди с дължина байт, дума или двойна дума;

• седем 32-разредни адресни регистри (A0-A6) за определяне адреса на операндите. Те могат да бъдат използвани като базови регистри и стекови указатели;

• сдвоен 32-разреден регистър A7 и A7' за стекови указатели в потребителски и системен режим. В асемблерските инструкции могат да се обозначат съответно с USP и SSP.

• 32-разреден програмен брояч PC, от който за адресация на външната памет се използват само младшите 24 бита;

• 16-разреден регистър за състояние SR. Младшият байт на SR съдържа кодовете на условия (CCR): C - пренос, V - препълване, Z - нулев резултат, N - знак и X - разширение за операции с повишена точност. Старшият байт съдържа разреди за управление на процесора: T - трасировка, I2, I1, I0 - маскиране на прекъсването, S - режим на работа.

Всеки един от данновите и адресните регистри може да бъде използван като индексен регистър.

МП работи в един от двата привилегировани режима - потребителски (S=0) и системен (S=1). Механизмът на привилегия е въведен за осигуряване на защита на областите на код и данни на отделните програми. В системен режим са достъпни всички ресурси на микропроцесорната система и могат да се изпълняват всички инструкции на МП. В него работят програмите на операционната система. Изключенията и прекъсванията се изпълняват в системен режим и всички стекови операции при обработката им използват стековия указател SSP. В по-ниско привилегирования потребителски режим не могат да се използват инструкциите, които имат важен ефект върху системата, например инструкциите ***stop, reset*** и за промяна на съдържанието на регистъра SR. Болшинството от програмите се изпълнява в потребителски режим.

### 1.2. Типове данни

Микропроцесорът поддържа пет типа данни: бит, байт (8 бита), дума (16 бита), дълга дума (32 бита) и двоично-десетични цифри (BCD).

### 1.3. Формати на инструкциите

Инструкциите на MC68000 са с променлива дължина (от 1 до 5 думи). Първата дума на всяка инструкция съдържа кода на операцията и указание за методите на адресация. Останалите думи носят допълнителна информация за изчисляване на адреса на операнда (отместване, непосредствен операнд, абсолютен адрес).

### 1.4. Методи за адресация

МП поддържа 14 метода за адресация:

• регистрова. Чрез нея може директно да се избира съдържанието на даннов или адресен регистър: **EA=Dn** или **EA = An**, където EA - ефективен адрес, Dn - номер на даннов регистър, An - номер на адресен регистър.

*Обозначение*: **An** или **Dn**. Вместо A7 може да се използва SP.

• косвена адресация. В групата на косвената адресация се отнасят:

• косвена регистрова. Като източник на адрес се използва адресен регистър A: **EA = (An)**.

*Обозначение*: **(An)**;

• автоинкрементна. Адресът на операнда се съдържа в адресен регистър, чието съдържание след извличането на операнда се увеличава с 1, 2 или 4 в зависимост от неговия размер - байт, дума или двойна дума: **EA = (An), An <—An + N**, където N = 1, 2 или 4.

*Обозначение*: **(An)+**;

• автодекрементна. Адресът на операнда се съдържа в адресен регистър, чието съдържание преди извличането на операнда се намалява с 1, 2 или 4 в зависимост от размера на операнда - байт, дума или двойна дума: **An <—An + N, EA = (An)**.

*Обозначение*: **-(An)**;

• базова адресация. Адресът на операнда се формира чрез прибавяне към съдържанието на адресен регистър на съдържащото се в инструкцията 16-разредно отместване d16: **EA = (An) + d16**.

*Обозначение*: **<expr16>(An)**, където <expr16> е израз, по който се изчислява 16-разредното отместване;

• индексна адресация. Адресът на операнда се формира от три компонента: съдържанието на адресен базов регистър An, съдържанието на индексен регистър Xn, който може да бъде адресен или даннов, и 8-разредно отместване d8: **EA = (An) + (Xn) + d8**.

*Обозначение*: **<expr8>(An,Xn)**, където <expr8> е израз, по който се изчислява 8-разредното отместване;

• абсолютна. Адресът на операнда се съдържа в инструкцията и може да бъде къс (16-разреден) и пълен. Късият адрес преди използване се разширява до пълен адрес чрез знаково разширение.

*Обозначение:* **<expr>**;

• непосредствена. Операндът се намира в инструкцията.

*Обозначение*: **#<expr>**;

• относителна. Адресът на операнда се изчислява като сума от съдържанието на програмния брояч PC и 16-разредно отместване в инструкцията: **EA = (PC) + d16**.

*Обозначение*: **<expr16>PC**;

• относителна индексна. Адресът се получава като сума от съдържанието на програмния брояч PC, съдържанието на индексен регистър и 8-разредно отместване в инструкцията: **EA□=(PC)+(Xn)+d8**.

*Обозначение*: **<expr8>(PC, Xn)**.

• неявна. Източникът на операнд се определя неявно от кода на операцията и може да бъде един от регистрите: SR, USP, SP, PC.

Автоинкрементната и автодекрементна адресации дават богати възможности за обработка на масиви и организация на потребителски стекове, като при това с елементите в стека могат да се извършват всички операции, допускащи адресация в паметта.

### 1.5. Основни типове инструкции

По-голямата част от инструкциите на процесора могат да използват всеки от 14-те метода за адресация и с малки изключения да оперират с всички размери операнди (байт, дума, дълга дума). По отношение на задаваните операции инструкциите на микропроцесора MC68000 се разделят на следните основни групи:

• прехвърляне на данни. Инструкцията *MOVE* е основната инструкция за прехвърляне на данни и адреси между регистрите на процесора, клетките на паметта и между регистрите и паметта. Освен *MOVE* могат да се използват и няколко специализирани инструкции: *MOVEM* (прехвърляне на множество регистри), *MOVEP* (обмен на данни с периферни схеми), *EXG* (размяна на съдържанието на два регистъра), *LEA* (зареждане на адрес в адресен регистър), *PEA* (запомняне на адрес в стека), *LINK* (свързване на два стека), *UNLK* (разделяне на два стека), *MOVEQ* (бързо прехвърляне на

непосредствени данни);

• целочислена аритметика. Аритметичните операции включват четирите базови целочислени операции: *ADD* (събиране), *SUB* (изваждане), *MUL* (умножение), *DIV* (деление), а така също и *CMP* (аритметично сравнение), *CLR* (нулиране), *NEG* (допълнение). Операциите събиране, изваждане и сравнение са дефинирани и върху операнди, представляващи валидни адреси, т.е. с размер 16 или 32 бита (*ADDA*, *SUBA*, *CMPA*). За повишена точност се използват операциите *ADDX* (разширено събиране), *SUBX* (разширено изваждане), *EXT* (разширение на знака), *NEGX* (допълнение с разширение), използващи флага X на регистъра SR. Операциите умножение и деление са възможни, както за знакови, така и за беззнакови операнди.

Към групата на аритметичните операции могат да се отнесат и инструкциите *TST* (сравнение с нула) и *TAS* (проверка и установяване на операнд). Инструкцията *TAS* е синхронизираща инструкция и се използва в мултипроцесорни системи за реализация на синхронизиращи примитиви (семафори и др.).

• логически. Инструкциите за логически операции (*AND*, *OR*, *EOR* и *NOT*) са възможни за всички размери операнди. За работа с непосредствени операнди мнемониката им е съответно *ANDI*, *ORI*, *EORI* и *NOTI*;

• преместване и ротация. Преместването може да бъде аритметично наляво (*ASL*) и надясно (*ASR*) и логическо наляво (*LSL*) и надясно (*LSR*). Инструкциите за ротация могат да бъдат с разширение (*ROXR*, *ROXL*) или без разширение (*ROR*, *ROL*). Тези операции се извършват над операнди във вътрешните регистри или в паметта. Инструкциите за преместване и ротация на съдържанието на регистри поддържат всички размери операнди със стъпка на преместване от 1 до 8 бита, зададена в инструкцията, и от 0 до 63 бита, зададена чрез регистър. Преместване и ротация в паметта се извършват само над операнди с размер дума и само със стъпка 1 бит;

• манипулация с битове. Операциите за манипулация с отделни битове се задават със следните инструкции: *BTST* (проверка на значението на даден бит), *BSET* (проверка и установяване значение на даден бит), *BCLR* (проверка и нулиране на даден бит), *BCHG* (проверка и промяна на значението на даден бит);

• двоично-десетични операции. Инструкциите, задаващи аритметични операции над двоично кодирани десетични числа, включват: *ABCD* (десетично събиране с разширение), *SBCD* (десетично изваждане с разширение), *NBCD* (десетично допълнение с разширение);

• управление на програмата. Инструкциите за управление на програмата се разделят на следните подгрупи:

• инструкции за условен преход: *Bcc* - група от 14 условни прехода в зависимост от условието *cc*, като е възможно 8- и 16-битово отместване, *DBcc* - инструкции за цикъл по условието *cc* с 16-разредно отместване, *Scc* - условно установяване на байт по условието *cc*. Условието *cc* може да бъде:

| | |
|---|---|
| *CC* - нулев пренос | *LS* - логическо по-малко или равно |
| *CS* - вдигнат пренос | *LT* - по-малко от |
| *EQ* - равенство | *MI* - минус |
| *F* - не истина (false) | *NE* - не равно |
| *GE* - по-голямо или равно | *PL* - плюс |
| *GT* - по-голямо от | *T* - истина (true) |
| *HI* - логическо по-голямо | *VC* - няма препълване |
| *LE* - по-малко или равно | *VS* - препълване |

• инструкции за безусловен преход: *BRA* - относителен безусловен преход на 8- или 16-разредно отместване по отношение на PC, *BSR* - относителен преход към подпрограма на 8- или 16-разредно отместване, *JMP* - безусловен преход, *JSR* - преход към подпрограма;

• инструкции за връщане от подпрограма: *RTR* - връщане с възстановяване на състоянието, *RTS* - връщане от подпрограма;

• операции за управление на системата. За управление на системата са въведени следните групи инструкции:

• привилегировани инструкции: *RESET* (начално установяване на външните компоненти), *RTE* (връщане от изключение), *STOP* (спиране на изпълнението на програмата), *ORI to SR* (логическо OR с регистъра на състояние SR), *MOVE USP* (зареждане на потребителския стеков указател), *ANDI to SR* (логическо AND с регистъра SR), *EORI to SR* (логическо EOR с регистъра SR), *MOVE EA to SR* (зареждане на ново съдържание на SR). Привилегированите инструкции се изпълняват само в режим

*супервайзор*. Опит за изпълнение на привилегирована инструкция в потребителски режим предизвиква изключение;

• инструкции за генериране на изключение: ***TRAP*** (капан), ***TRAPV*** (капан при препълване), ***CHK*** (проверка дали съдържанието на данков регистър е в зададени граници);

• инструкции за работа с регистъра CCR: ***ANDI to CCR*** (логическо AND с кодовете на условие), ***EORI to CCR*** (логическо EOR с кодовете на условие), ***MOVE EA to CCR*** (зареждане на нови кодове на условие), ***ORI to CCR*** (логическо OR с кодовете на условие), ***MOVE SR to EA*** (съхранение на регистъра на състояние).

Пълна таблица на инструкциите на МП е приведена в Приложение 3.

## 2. Организация на шината

Шината на МП е организирана на принципа "главен - подчинен" (master - slave) с асинхронно управление. Във всяко предаване по шината участвуват две устройства - главно (master), управляващо предаването, и подчинено (slave), извършващо указаните от главното устройство действия. Главни устройства могат да бъдат МП и устройствата с пряк достъп до паметта, а подчинени - паметта и периферните устройства.

### 2.1. Сигнали на MC68000

Сигналите на МП (фиг.14.1) са обединени в следните функционални групи:

• Адресна магистрала (A1-A23).

• Даннова магистрала (D0-D15).

• Асинхронно управление на шината. Асинхронното предаване на данни по шината се управлява със сигналите:

AS# (Address strobe) - задава валиден адрес на адресната магистрала.

R/W# (Read/Write) - посока на предаване на данните: четене или запис.

UDS#, LDS# (Upper and Lower Data Strobe) - строб на данните върху данновата магистрала. UDS# стробира старшия (разреди 8-15) байт, а LDS# - младшия (разреди 0-7) байт. Значението на тези сигнали се определя от дължината на предавания операнд и нулевия бит от неговия адрес. Например при байтова операция и четен адрес (бит A0 = 0) се активира UDS#, а при нечетен адрес (A0 = 1) - LDS#.

DTACK# (Data Transfer Acknowledge) - входен сигнал, индициращ завършило предаване на данните в случай на запис, или валидни данни в случай на четене.

• Арбитраж на шината. В тази група са включени три сигнала, с помощта на които вътрешна в МП арбитражна схема определя кое устройство ще бъде главно на шината:

BR# (Bus Request) - входен сигнал, с който външно устройство издава заявка към МП за заемане на на шината;

BG# (Bus Grant) - изходен сигнал, с който МР уведомява всички останали потенциални главни устройства, че в края на текущия цикъл ще освободи шината;

BGACK# (Bus Grant Acknowledge) - входен сигнал, индициращ, че друго устройство става главно на шината след издаването на BG#;

• Управление на прекъсването. Външните устройства са групирани в 7 приоритетни нива (с номера от 1 до 7) по отношение на заявките за прекъсване към микропроцесора. Номерът на приоритетното ниво на устройството, издало заявка за прекъсване, се подава на входовете IPL0#, IPL1#, IPL2#. Ниво 7 има най-висок приоритет, а ниво 0 индицира отсъствие на заявка за прекъсване;

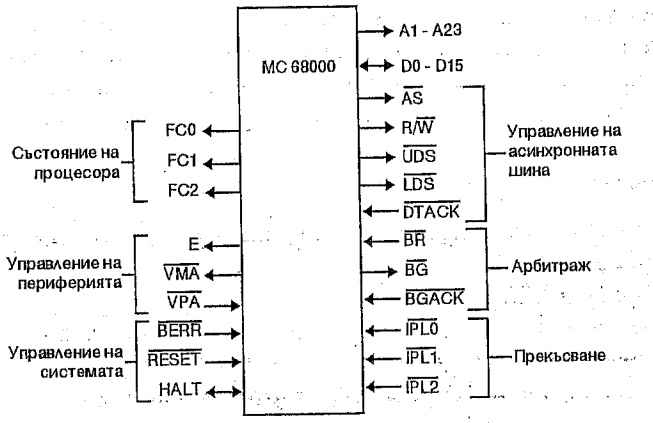
• Управление на системата. Входните сигнали за управление на системата се използват за начално установяване на МП (RESET#), приостановяване на МП след завършване на текущия цикъл (HALT#) или за индикация за грешка по шината (BERR#). Сигналите RESET# и HALT# могат да бъдат изходни. RESET# се издава в резултат от изпълнението на инструкцията ***RESET***, а HALT# при спиране на изпълнението на програмата, например при възникване на двойна грешка по шината;

• Управление на периферията. Управляващите сигнали E (Enable), VPA# (Valid Peripheral Address), VMA# (Valid Memory Address) се използват за съвместимост с периферни устройства, разработвани за синхронния МП M6800;

• Състояние на процесора. Изходните сигнали FC0, FC1, FC2 задават т.н. функционален код и индицират режима на МП (потребителски или системен) и типа на текущия цикъл на шината. Допустими са следните комбинации:

| FC2 | FC1 | FC0 | режим | тип на цикъла |
|---|---|---|---|---|
| 0 | 0 | 1 | потребител | предаване на данни |
| 0 | 1 | 0 | потребител | четене на инструкция |
| 1 | 0 | 1 | супервайзор | предаване на данни |
| 1 | 1 | 0 | супервайзор | четене на инструкция |
| 1 | 1 | 1 | подтвърждаване на прекъсване | |

С помощта на тези сигнали могат да бъдат отделени адресните пространства за данни и програми (харвардска архитектура), а също да се реализират отделни памети за системен и потребителски режим. По този начин се получават 4 независими адресни пространства, всяко с обем 16 MB. Сумарното адресно пространство на МП може да бъде увеличено до 64 MB;




Фиг. 14.1. Сигнали на MC68000

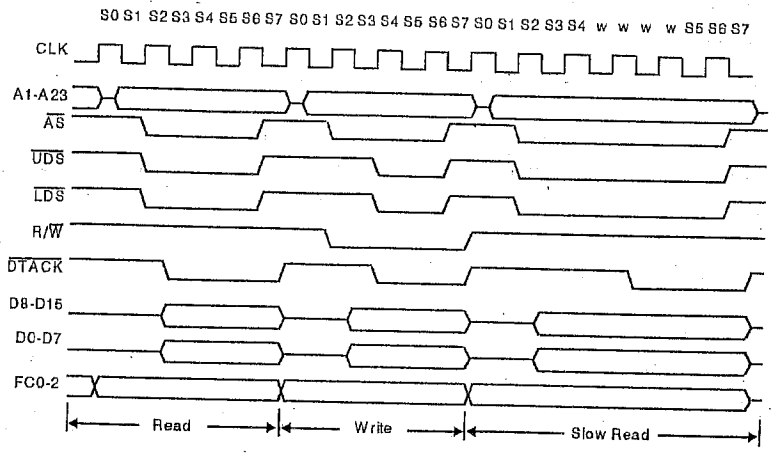
### 2.2. Цикли на шината

Всички управляващи и даннови сигнали се възприемат и въздействат по време на положителния полупериод на тактовия сигнал CLK. Във всичките цикли главното устройство трябва да управлява сигналите, които издава, в началото и в края на цикъла, а така също да приема сигналите за подтвърждение и данни от подчиненото устройство. За шината са характерни следните типове цикли: цикли за запис и четене на байт и дума; цикъл за подтвърждаване на прекъсване, чрез който се получава номерът на вектора на прекъсване; цикъл на арбитраж за предоставяне на шината на друго главно устройство; заключен цикъл *Четене-Модификация-Запис*; цикли при грешка на шината и операцията *HALT*.

При цикъл за четене главното и подчиненото устройсво работят по следния начин:

| **Главно устройство** | | **Подчинено устройство** |
|---|---|---|
| Адресира подчиненото устройство | | |
| 1. R/W# = 1 | | |
| 2. Разполага кода FC0-FC2 | | |
| 3. Разполага адреса на A1-A23 | | |
| 4. Активира AS# | | |
| 5. Установява UDS# и LDS# в зависимост от размера | ——> | Предоставя данните |
| | | 1. Декодира адреса |
| | | 2. Разполага данните върху D0-D15 |
| | | 3. Активира DTACK# |
| Приема данните | <—— | |
| 1. Приема данните | | |
| 2. Дезактивира UDS# и LDS# | | |
| 3. Дезактивира AS# | ——> | Завършва цикъла |
| | | 1. Освобождава данновата шина |
| | | 2. Дезактивира DTACK# |
| Стартира нов цикъл | <—— | |

Аналогични действия се извършват и при цикъл за запис. На фиг.14.2 е показана времедиаграма за циклите четене и запис. Сигналът DTACK# се проверява в състояние на шината S4, а в състояние S6 се приемат данните. В случай, че сигналът DTACK# не е активен при достигане на състоянието S4 се въвеждат състояния на изчакване w.




Фиг. 14.2. Времедиаграма на циклите четене и запис

Циклите *Четене-Модификация-Запис* са въведени за реализация на мултипроцесорни системи и са неделими (заключени). Заключването на шината се осъществява чрез задържане активен сигнала AS# през целия цикъл. Докато сигналът AS# е активен шината не се арбитрира за друго главно устройство. Тези цикли се използват при инструкцията *TAS*.

При циклите за подтвърждаване на прекъсване върху адресните линии A1, A2 и A3 се извежда номерът на нивото на приетото прекъсване, а останалите адресни линии се привеждат в единично състояние.

### 3. Прекъсвания и изключения

Обработката на прекъсвания и изключения се състои от 4 основни стъпки: съхраняване на регистъра на състояние SR във временен вътрешен регистър и установяване на МП в системен режим; определяне на вектора на прекъсване; съхраняване текущото състояние на МП в стека на системния режим; превключване на МП към процедурата за обработка. Векторите са с дължина две думи и определят адреса на входните точки на процедурите за обработка на съответните прекъсвания или изключения. Разполагат се в системната област за данни от адреси 0h-1023h, като се осигуряват 255 уникални вектора. Векторите са достъпни чрез своите еднобайтови номера, които се генерират вътрешно или се получават от външната среда. Външните причини за прекъсване на работата на МП са сигнали за прекъсване от външни устройства, грешка по шината и RESET. Вътрешните събития, предизвикващи изключения, са вследствие на инструкция, грешка в адреса и трасировка. Инструкциите *TRAP*, *TRAPV*, *CHK* и *DIV* могат да предизвикат изключение като част от своето изпълнение. Изключение се предизвиква и от невалидна инструкция, извличане на дума от нечетен адрес и нарушение на привилегията.

Прекъсванията могат да бъдат маскирани чрез флаговете I0, I1 и I2 на регистъра CR. Възприемат се само прекъсвания от нива, с по-висок номер от установения с тези флагове, който се интерпретира като приоритет на МП. Микропроцесорът извлича номера на вектора от прекъсващото устройство чрез цикъл за подтвърждаване на прекъсването. Особеност на процесорите MC680x0 е възможността ВУ да заяви автоматична генерация на номер на вектор (автовектор), който се определя в зависимост от нивото на прекъсване. Възможни са 7 различни автовектора на прекъсване.

## 4. Особености на асемблера

### 4.1. Език на асемблера

Синтаксис на програмен ред:

```
<label> <operation> <src_operand>,<dst_operand> <comment>
```

където:

**<label>** - етикет. Етикетът или се разполага от първа позиция на реда, или трябва да завършва със символа ":". Етикетът не е задължителен;

**<operation>** - код на операция. Съдържа мнемоничното обозначение на инструкция или псевдоинструкция. За инструкции, които могат да оперират с операнди с различна дължина, размерът на операнда се задава с помощта на суфикс: .B - байт, .W - дума, .L - дълга дума. По премълчаване се подразбира операнд с дължина една дума (16 бита). Ако няма етикет полето <operation> трябва да бъде предшествувано поне от един интервал;

**<src_operand>** - операнд-източник;

**<dst_operand>** - операнд-назначение;

Броят на операндите зависи от кода на операцията. Операндите се задават чрез изброените в 1.4 методи за адресация.

**<comment>** - коментар. Ако коментарът се разполага в отделен ред, той се отбелязва със символа "*" в първата позиция на реда.

**Пример:**

```
loop    MOVE.L   (A0)+,(A1)+   Пример на програмен ред
```

Примерът демонстрира инструкция за прехвърляне на двойна дума от адрес в адресния регистър A0, на адрес, зададен от съдържанието на регистър A1. И двата операнда се адресират чрез автоинкрементна адресация.

В полето на операндите могат да се използват изрази, които се изчисляват до 32 разредно двоично число. Типът на константите в тях се определя с префикс и може да бъде: без префикс - десетично число, $ - шестнадесетично число, % - двоично число, @ - осмично число. ASCII константи и символни низове се заключват в единични кавички. При символните низове най-левият байт се разполага в байта с най-малък адрес.

Оператори в изразите могат да бъдат: "-" унарен минус, "~" инверсен код, "<<" преместване наляво (X << Y означава преместване на операнда X на Y разреда наляво), ">>" преместване надясно, "$" побитово AND, "!" побитово OR, "*" умножение, "/" целочислено деление, "\" деление по модул, "+" събиране, "-" изваждане.

### 4.2. Псевдокоманди (директиви) на асемблера

• Установяване на програмния брояч

Със псевдокомандата ORG се задава началният адрес на програмата:

```
<label> ORG <expression>
```

• Дефиниране на символи.

• дефиниране на константи:

```
<symbol> EQU <expression>
```

**<symbol>** задава името на константата, а <expression> е израз, определящ нейното значение.

• задаване значение на символ, което впоследствие може да бъде променяно в програмата:

```
<symbol> SET <expression>
```

• дефиниране на регистров диапазон

```
<symbol> REG <expression>
```

С името <symbol> се означава произволно множество от данновите и адресните регистри на МП. Множеството може да се формира чрез обединяване на диапазони в масива регистри, разделени посредством "/".

**Пример:**

```
WORKSET   REG   D0-D3/D7/A1-A3
```

формира множество с име WORKSET, състоящо се от регистрите D0, D1, D2, D3, D7, A1, A2 и A3. Множеството WORKSET може да се използва като операнд в инструкцията ***MOVEM***. Например

```
MOVEM.L   WORKSET, -(SP)
```

съхранява в стека регистрите от множеството WORKSET.

• Определяне на памет.

• записване на низове и списъци от константи в последователни адреси в паметта:

```
<label> DC.<size> <item>, <item>, . . .
```

където

**<label>** задава началото на областта на константите в паметта;

**<size>** определя дължината на константите (B - байт, W - дума, L - дълга дума);

**<item>** може да бъде израз или низ.

**Пример:**

```
TEXT DC.B 'DC Example', $0D, $0A, 0
```

Резултатът от изпълнението на горната псевдокоманда ще се изрази в следната последователност от данни в областта TEXT:

```
44 43 20 45 78 61 6D 70 6C 65 0D 0A 00
```

• дефиниране на блок от байтове, думи или дълги думи, които се инициализират с едно и също значение:

```
<label> DCB.<size> <length>, <value>
```

където

**<length>** - брой единици от указания размер <size>;

**<value>** - инициализиращо значение.

• задаване на неинициализиран блок от байтове, думи или дълги думи:

```
<label> DS.<size> <length>
```

• край на програмата - END

### 4.3. Стартиране на асемблера

Асемблерът работи върху персонални компютри IBM PC. Стартира се от MSDOS чрез следния команден ред:

```
ASM   <options> <filename>,
```

където

**<options>** опции, задавани като последователност от символи, предшествувани от тире "-". Основните опции са:

**C** - показва всички константи, генерирани чрез директивата DC;

**L** - изработва листингов файл;

**N** - не генерира обектен код.

По премълчаване се генерира обектен код, но не се формира листинг;

**<filename>** - име на файла с изходния текст на програмата, което трябва да бъде с разширение .ASM.

В резултат на асемблирането се генерира файл с обектен код в ASCII формат със същото име, но с разширение .H68 и ако е зададен - листингов файл с разширение .LIS.

**Пример:**

Основните особености и предимства на микропроцесора MC68000 са:

• 32 разредна вътрешна архитектура;

• ортогонална система инструкции (независими полета за код на операция и адрес на операндите във формата на инструкцията);

• автоинкрементна и автодекрементна адресация, даващи възможност за ефективна обработка на списъчни структури, работа с множество потребителски стекове с възможност за комплексни операции при запис и четене в тях;

• инструкции за множествено прехвърляне "регистри - памет";

• гъвкава система за прекъсване, изразяваща се във възможността за прекъсване на няколко нива и за предоставяне на автовектори на прекъсване, което позволява използването на МП в системи за управление в реално време;

• умножение и деление със и без знак;

• директно адресиране на 16 MB и адресиране на 64 MB посредством функционално сегментиране.

Изброените възможности се запазват и обогатяват в следващите 32-разредни микопроцесори на фамилията MC680x0.

Приведената по-долу програма демонстрира удобството от автоинкрементната и автодекрементна адресация при обработката на масиви. Програмата умножава с 2 елементите на масив, записан в областта **inarr** и записва резултата в друг масив **outarr** в обратен ред. Елементите на масивите са с размер дума.

```
        ORG       $1000          Начален адрес на програмата
inarr   DC.W      4,7,8,2        Записване елементите на нач. масив
outarr  DS.W      4              Заделяне памет за резултантния масив
        MOVE.W    #inarr, A1     Адресът на нач. масив в регистър A1
        MOVE.W    #outadrr+8,A2  Адресът на последния елемент в A2
        MOVE.B    #4, D0         Брой елементи в D0
loop    MOVE.W    (A1)+, D1      Поредният елемент от нач. масив в D
        MULU.W    #2, D1         Беззнаково умножение с 2
        MOVE.W    D1, -(A2)      Запис в предходния елемент на рез. масив
        SUB.B     #1, D0         Намаляване с единица брояча на елементи
        BNE       loop           Преход към loop при ненулев резултат
        END                      Край на програмата
```

Най-старши представител на фамилията MC680x0 е 32 разредният микропроцесор MC68060, който е със суперскаларна архитектура с възможност за изпълнение на 3 инструкции на такт и производителност 110 MIPS при тактова честота 75 MHZ. В кристала на МП са включени устройство за работа с плаваща запетая (FPU), устройство за управление на паметта (MMU), разделена вътрешна кеш памет за данни и инструкции по 8KB и управление на консумацията. МП е разположен в корпус с 206 извода.

## 5. Едноплатков микрокомпютър на базата на MC68000

### 5.1. Архитектура на едноплатков микрокомпютър MEX68KDM

Лабораторните упражнения се провеждат с микропроцесорна система за проектиране MEX68KDM. Тя представлява пълна микрокомпютърна система на базата на MC68000 (фиг.14.3), включваща:

• 32 KB динамична RAM памет (ИС MEX6815-3);

• 8 KB ROM памет с управляващата програма MACSbug;

• два 16-разредни паралелни входно-изходни порта (ИС MC6821P);

• три 16-разредни таймера (ИС MC6840);

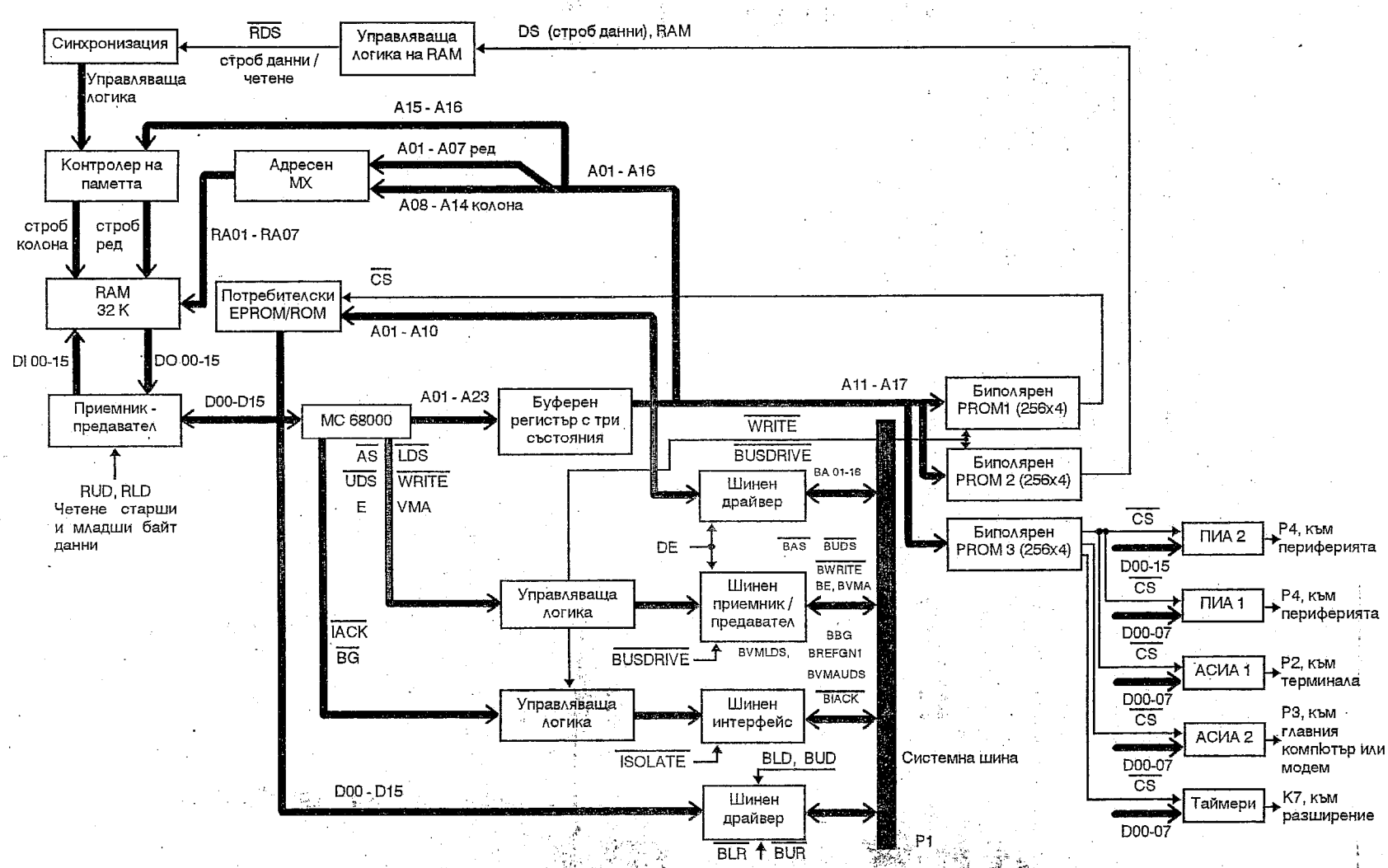
• цокли за допълнителни схеми ROM/EPROM;

• използвани 17 адресни разреда, определящи 256 KB адресно пространство;

• 16-разредна шина данни;

• два асинхронни входно-изходни порта (ACIA - ИС MC6850L) с максимална скорост за обмен 9600 бита/s, единият от които е конфигуриран за свързване към терминал, а другият - за свързване към компютърна система за развитие EXORciser.

Системата може да работи автономно, като се свързва с интелигентен терминал от тип VT100 посредством стандартен интерфейс RS-232C, реализиран чрез единия от ACIA портовете (ACIA1). В лабораторното упражнение в качеството на терминал се използва персонален компютър IBM PC, който програмно емулира VT100. Това дава възможност програмите за системата да бъдат създавани и асемблирани със средствата на персоналния компютър, след което да бъдат прехвърляни и зареждани за изпълнение в паметта на MEX68KDM.




Фиг. 14.3. Архитектура на системата за проектиране MEX68KDM

Адресните линии A01-A23 на МП са свързани към три 8-разредни буферни регистъра с трето състояние, от изходите на които се подават към RAM, EPROM и системната шина. Старшите битове на адреса A11-A17 се използват за отделяне на адресните пространства на RAM, EPROM и периферните чипове посредством три адресни декодери на базата на памети PROM (256 x 4). Адресните сигнали A18-A23 в системата не се използват. Декодерите изработват сигналите избор на чип (CS): PROM1 - за избор на чип от четирите двойки цокли потребителски EPROM/ROM; PROM2 - определя адресното пространство на RAM паметта, изградена от 16 блока по 2KB; PROM3 - изработва индивидуални сигнали CS за входно-изходните чипове и таймера.

Приемо-предавателите на данновата шина са двупосочни, като DI00-DI15 са входни, а DO00-DO15 - изходни даннови сигнали от RAM паметта. Високото ниво на сигналите RUD (четене на стариш байт) и RLD (четене на младиш байт) задава четене от паметта, а ниското ниво на същите сигнали - запис в паметта. Адресният мултиплексор преобразува адресните сигнали A01-A14, задаващи номер на ред (A01-A07) и номер на колона (A08-A14), в седем адресни линии RA01-RA07 за свързване към адресните изводи на интегралните схеми на паметта. Мултиплексорът съдържа адресен брояч за адресиране по време на циклите за опресняване на паметта. Адресните сигнали A15 и A16 постъпват в контролера на паметта и се използват за генериране на стробиращите импулси на адреса на реда и адреса на колоната за RAM, а така също и стробиращите сигнали за запис на стариш (WU#) и младиш (WL#) байт. Синхронизиращите и закъснителни сигнали се осигуряват от схемата за синхронизация. Сигналът RDS# (RAM Data Strobe) стартира синхронизиращата схема. Този сигнал се изработва от управляващата логика на RAM на базата на сигналите DS (Data Strobe) и избор на RAM, генерирани от PROM2. Синхронизиращата схема изработва и сигнала за край на предаването DTACK#.

Адресните, данновите и управляващите сигнали през съответни шинни драйвери се подават към външна (системна) шина. Тази шина през куплунг P1 служи за подключване на допълнителни външни компоненти.

### 5.2. Управляваща програма MACSbug 1.32

Управляващата програма MACSbug осигурява възможност за зареждане на програмите и управление на тяхното изпълнение, тестване на паметта и регистрите на МП. Командите на MACSbug имат следния формат:

```
*COmmand parameters; options,
```

където:

* - промпт на програмата;

**CO** - необходими за разпознаване на командата символи. Името на всяка команда се разпознава по първите една или две главни букви;

**mmand** - останалата част от името на командата, която в синтаксиса се използва само за по-добра четливост и може да бъде пропускана;

**parameters** - параметри, зависещи от конкретната команда. Обикновено се задават в шестнадесетичен код. ASCII символни низове се затварят в единични кавички;

**options** - модифицират действието на командата.

Управляващата програма се активира след включване на захранването и инициализация чрез бутона RESET. MACSbug изпълнява следните основни команди:

**reg#** - извежда съдържанието на зададения регистър;
**reg# hexdata** - зарежда шестнадесетично значение (hexdata) в регистър;
**reg# 'ASCII'** - зарежда в регистър ASCII кодове;
**reg#:** - извежда текущото съдържание на регистър и изисква нова стойност;
**class** - извежда съдържанието на зададения клас регистри (A - адресни, D - даннови);
**class:** - извежда съдържанието на зададения клас регистри и изисква нови значения;
**DM start end** - визуализира съдържанието на паметта от адрес **start** до адрес **end**;
**SM address data** - записва данни **data** по зададен адрес **address**;
**OPen address** - отваря памет за четене/модификация;
**Go** - стартира програма от начален адрес, зареден в програмния брояч;
**Go address** - стартира програма от зададен адрес **address**;
**Go TILL address** - задава временна точка на прекъсване по адрес **address** и стартира програмата;
**BReakpoint** - отпечатва всички точки на програмно прекъсване;

**BR address:count** - задава нова точка на прекъсване и брой прекъсвания в нея;
**BR- address** - анулира точката на програмно прекъсване по адрес **address**;
**TD** - визуализира състоянието на микропроцесора в стъпков режим;
**TD reg#, format** - включва във визуализацията на стъпковия режим регистъра **reg#** с указания формат;
**TD CLEAR** - изключва всички регистри от визуализация при стъпков режим;
**TD ALI** - включва всички регистри за визуализация при стъпков режим;
**T** - трасира една инструкция;
**T count** - трасировка на зададен брой инструкции;
**T TILL address** - постъпково изпълнение до достигане на зададения адрес;
**REad; =text** - изчаква получаване на запис 'S' от терминала;
Мнемонични обозначения на регистрите:
D0, D1, ..., D7 - даннови регистри;
A0, A1, ..., A7 - адресни регистри;
PC - програмен брояч;
SR - регистър на състоянието;
SS - стеков указател в системен режим;
US - стеков указател в потребителски режим.

### 5.3. Работа с микропроцесорна система за проектиране MEX68KDM

Платката с микропроцесорната система MEX68KDM се подключва през куплунг P2 с интерфейсен кабел RS-232C към порт COM1 или COM2 на IBM PC. Стартира се програмата за терминална емулация MTE, заредена в IBM PC. От главното меню на MTE се избира Change, а от подменюто Parameters, за задаване на параметрите на комуникационната линия: порт (COM1 или COM2); скорост, равна на зададената чрез джъмперите на платката (фабрично установена 9600 бода); размер на думата (8 бита); контролен бит (не); брой стопови битове (1). След задаване на параметрите се установява режим Terminal от главното меню на MTE. След включване на захранването и инициализиране с бутона RESET на системата на екрана на IBM PC се извежда:

```
MACSBUG 1.3
*
```

За да се зареди изпълнима програма в паметта на системата се процедира по следния начин:

- срещу промпта на системата се въвежда командата:

```
* READ;=
```

- избира се режим File от главното меню на MTE. От подменюто на File се стартира Send, избира се ASCII формат и в изведения прозорец се записва името на файла с компилираната програма. Програмата се зарежда от зададения в нея начален адрес, след което може да бъде стартирана за изпълнение с командата GO. Изпълнението може да бъде проследено с командите, изброени в 7.2.

**Въпроси и задачи:**

1. Въведете в основната памет на едноплатковия микрокомпютър MEX68KDM дадената в т.6 програма. Изпълнете програмата в стъпков режим, като снемете състоянието на МП на всяка стъпка. Потребителската област в RAM в системата MEX68KDM заема адреси 100h-3FFh.

2. Снемете с логически анализатор времедиаграмите на шината на МП при изпълнение на отделните инструкции. Използвайте сонда за МП MC68000.

3.Напишете програма за събиране на две 56 битови числа, като всяко число се разполага в 7 последователни байта в паметта. Изпълнете програмата в системата MEX68KDM в стъпков режим, като снемете състоянието на МП на всяка стъпка.

4. Напишете програма за сортировка на едномерен масив с помощта на два стека. Изпълнете програмата в системата MEX68KDM в стъпков режим. В единия стек се формира сортираният масив, а другия е помощен и се използва за разтоварване на първия стек при необходимост от преподреждане. Използвайте автоинкрементна и автодекрементна адресация за работа с двата стека.

5. Проектирайте логическа схема, която: да изпраща към МП заявки за прекъсване от външните устройства на дадено приоритетно ниво; да предоставя номера на прекъсване в отговор на цикъл по шината "потвърждаване на прекъсване".

# 15. Микропроцесорни системи на базата на PowerPC

Целта на упражнението е запознаване с архитектурните особености на семейството RISC микропроцесори PowerPC. Разглеждат се работата на суперскаларния конвейер на инструкциите, основните характеристики на системата инструкции, организацията на паметта.

## 1. Особености на архитектурата PowerPC

Архитектурата PowerPC е трето поколение RISC архитектура, базирана на използваната в работната станция System/6000 архитектура POWER на фирмата IBM. Характеризира се с реализиране на следните възможности в единичен микропроцесорен чип:

- суперскаларност. Концептуално инструкциите се разпределят за изпълнение едновременно върху три функционално независими изпълнителни устройства - устройство за обработка на преходите, устройство за операнди с фиксирана запетая и устройство за операнди с плаваща запетая. Паралелизмът може допълнително да се увеличи при реализацията на конкретен процесор чрез размножаване на тези устройства (например в процесора PPC 604 има вградени две устройства с фиксирана запетая);
- динамично преподреждане на инструкциите по време на изпълнение. Инструкциите се планират за изпълнение по реда си в програмата, а завършват изпълнението в друг ред;
- предсказване и "спекулативно" изпълнение на преходите в програмата;
- удобни и ефективни средства за изграждане и поддържане на мултипроцесорни системи;
- поддържане на многозадачни ОС;
- естествено разширение към реална 64 разредна архитектура (PPC 620);

### 1.1. Функционално описание на конвейера на инструкциите

Първият реализиран 32 разреден процесор, напълно поддържащ архитектурата PowerPC, е PPC 603 (фиг.15.1). Степените на основния конвейер са:

- **извличане** (Fetch). Инструкциите се извличат от кеш паметта за инструкции по две във всеки такт и се помещават в шест елементна опашка (IQ). При постъпването си в опашката инструкциите за преход веднага се придвижват в устройството за обработка на преходите;
- **декодиране и извличане на операнди**. На тази степен диспечерът извлича от IQ по две поредни инструкции на всеки такт, декодира ги, заделя необходимите временни регистри, чете операндите в тях и разпределя инструкциите в съответните изпълнителни устройства.
- **изпълнение**. Изпълнителните устройства изпълняват действията, зададени от инструкциите, записват резултатите в назначените временни регистри и уведомяват завършващото устройство. Ако някой резултат е необходим като операнд за изпълнението на друга инструкция, то той веднага се изпраща към съответното изпълнително устройство. Предизвикалата изключение по време на своето изпълнение инструкция се маркира.
- **завършване**. Завършващото устройство (EU) записва съдържанието на временните регистри в архитектурните регистри и ги освобождава за бъдещо използване. Ако дадена инструкция е отбелязана като предизвикала изключение, EU преустановява конвейера и инициира обработката на изключението. Тъй като инструкциите могат да завършват в произволен ред, в EU е реализиран механизъм за тяхното подреждане и изхвърляне от завършващия буфер в програмен ред. Това гарантира стриктната обработка на изключенията.

Характерно за PPC 603 е разпределението във всеки такт на три инструкции в пет различни изпълнителни устройства: за обработка на преходите (BPU), с фиксирана запетая (IU), с плаваща запетая (FPU), за запис и четене, и за работа със системните регистри.

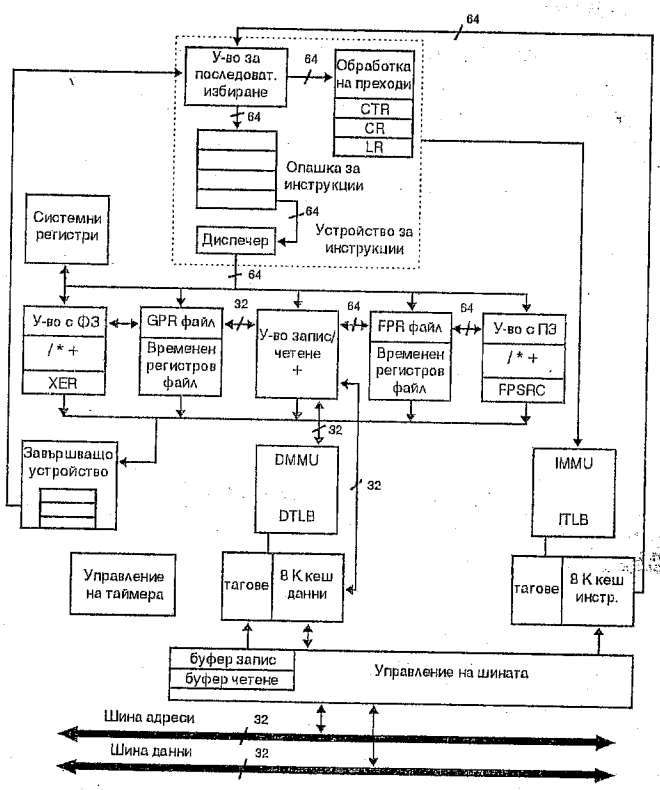
**Устройството с фиксирана запетая** изпълнява повечето инструкции за един такт с изключение на умножение (2-6 такта) и деление (37 такта).

**Устройството с плаваща запетая** изпълнява операции с единична и двойна точност с нормализирани и ненормализирани операнди. Реализирано е посредством тристепенен конвейер, като операциите с единична точност и повечето операции с двойна точност се изпълняват за три такта. Такава реализация позволява в FPU да се планира по една инструкция на всеки такт. Операцията деление не е конвейризирана и изпълнението й заема 18 такта за единична и 33 такта за двойна точност.

**Устройството за запис и четене** изпълнява зареждане и съхранение на регистрите с общо предназначение (GPR) и регистрите с плаваща запетая (FPR). Реализирано е като двустепенен конвейер с вградена схема за изчисляване на ефективните адреси.

**Устройството за обработка на преходите** декодира и изпълнява болшинството инструкции за преход за един такт. Условните преходи се разрешават на базата на 32 разреден регистър на условията (CR) и регистър брояч на цикли (CTR). В случай че условният преход е разрешим, т.е. съответният бит в CR не е блокиран от изпълнявана в момента инструкция, то адресът на прехода се изчислява в същия цикъл. В противен случай адресът се получава чрез статично предсказване на прехода, съгласно зададен бит в кода на инструкцията за преход. Извличаните след предсказан преход инструкции се отбелязват като "спекулативни" и се изпълняват в конвейера по нормален начин. Виновният бит в CR се проверява във всеки такт. При грешна прогноза отбелязаните като спекулативни инструкции се изхвърлят от конвейера. Не могат да се запускат повече от един спекулативен преход. Включеният в BPU регистър LR се използва за съхраняване на адреса на връщане след преход.

**Устройството за работа със системните регистри** реализира операциите над съдържанието на управляващите регистри в процесора.




Фиг. 15.1. Вътрешна структура на PPC603

### 1.2. Разрешаване на зависимостите в конвейера

Суперскаларната архитектура, съчетана с предсказване на преходите и непоследователно изпълнение на инструкциите, позволява да се достигне особено висока производителност на процесорите от серията PowerPC. От друга страна тя поставя проблеми при поддържането на последователен от гледна точка на програмиста модел, произтичащи от зависимостите в потока инструкции (например следващите след преход инструкции са зависими от неговото решение) и потока данни (ползване на общи данни от различни инструкции в задължителна последователност). В общия случай тези проблеми водят до преустановяване на конвейера за определено количество

тактове и снижават неговата производителност. За намаляване на влиянието им в PowerPC са предвидени следните апаратни решения:

• включване във всяко изпълнително устройство на независим набор архитектурни (програмно достъпни) регистри. Това минимизира необходимостта от комуникация и синхронизация между устройствата, които при необходимост трябва да се реализират и планират явно от компилатора;

• временни (програмно недостъпни) регистри. Регистровите файлове GPRR и FPRR осигуряват временно съхраняване на резултатите от изпълнението на инструкциите. По този начин се предотвратява престой на конвейера при опит за едновременен достъп на две инструкции до един и същ архитектурен регистър. Улеснява се също и анулирането на резултатите от спекулативно изпълнените инструкции без да е необходима промяна в архитектурните регистри GPR, FPR и CR;

• включване във всяко изпълнително устройство на едноелементна опашка, в която инструкция може да изчака докато необходимите й операнди станат достъпни. Това позволява на диспечера да разпределя следващите инструкции от входната опашка към другите устройства без задръжка;

• работа на отделните изпълнителни устройства с различни битове за условие в регистъра на условията CR. По този начин отпада необходимостта от синхронизация на достъпа до общи ресурси.

### 1.3. Подсистема на паметта

#### 1.3.1. Управление на паметта

32 разредната архитектура PowerPC поддържа 52 разреден виртуален адрес и 32 разреден реален, а 64 разредната - 80 разреден виртуален и 64 разреден реален адрес на паметта. Преобразуването на ефективния адрес (EA) се извършва от MMU на два етапа - сегментиране (преобразува ефективния адрес във виртуален) и страницираме (преобразува виртуалния адрес в реален). Ефективното адресно пространство се дели от MMU на сегменти с еднакъв размер (256MB), разположени на адреси, кратни на 256MB. Всеки сегмент съдържа 64K страници, всяка с размер 4KB. Сегментирането при 32 разредните PPC се извършва с помощта на 16 сегментни регистри (SR), посредством които сегмент i от ефективното адресно пространство, определен от разреди 0-3 на EA, се проектира в сегмент j от виртуалното адресно пространство, определен от разреди 8-31 на SRi. Страницирането се извършва с помощта на резидентна таблица на страниците. Базовият адрес и размерът на таблицата се съхраняват във вътрешенен регистър SDR1. Индексът в таблицата е функция на номера на виртуалния сегмент j, номера на страницата в сегмента, определен от разреди 4-19 на EA, и размера на таблицата. Съответният елемент от таблицата (PTE) съдържа базовия реален адрес на страницата, ако тя е заредена в паметта и правата на достъп до нея. За да се ускори преобразуването на адресите в MMU се включва специален кеш буфер (TLB), в който автоматично се съхраняват PTE на използваните страници. Ако PTE на необходимата страница е вече в TLB, то преобразуването се съкращава до един такт.

В PPC 603 са реализирани отделни устройства за управление на паметта за данни (DMMU) и инструкции (IMMU), всяко от които работи със свой 64 елементен TLB буфер (DTLB и ITLB съответно). Търсенето в таблицата на страниците и зареждането на необходимия PTE в TLB се осъществява програмно чрез механизма на изключенията.

Архитектурата позволява дефинирането на големи области памет като блокове с различен размер (от 128KB до 256MB). Назначението им е по-бърз достъп до информация, над която са установени определени операционни правила (например големи видеобуфери, големи информационни масиви и др.). Обявените блокове са изцяло резидентни и над тях не се изпълнява механизмът на страницирането. Транслирането на ефективния адрес се извършва с помощта на специални регистри - BAT регистри. Даден блок се описва с един BAT регистър, който съдържа следната информация: начален ефективен адрес; дължина; начален физически адрес, от който е разположен блокът в паметта; права и атрибути на достъп. В архитектурата са определени четири BAT регистъра за данови блокове (DBAT) и четири BAT регистъра за код (IBAT).

#### 1.3.1. Вътрешна кеш памет

Вътрешната кеш памет в различните процесори от серията PowerPC може да бъде обща или разделена за инструкции и данни, с обратен или транспарентен запис.

## 2. Програмен модел. Система инструкции

Основните характеристики на системата инструкции на PowerPC са типични за RISC архитектурите:

- фиксирана дължина на инструкциите;
- малко и прости методи за адресация;
- голям брой програмно достъпни регистри с общо предназначение (регистров файл);
- три операнден формат на инструкциите;
- достъп до операндите за болшинството инструкции от тип "регистър-регистър".

### 2.1. Програмно достъпни регистри

Номерацията на битовете в регистрите на процесора се подчинява на схемата big-endian, при която най-старши значещ е най-левият бит, имащ номер 0, а най-младши - най-десният с номер 31 или 63 в зависимост от разредността. Достъпни за потребителските приложни програми (потребителски режим на МП) са следните регистри:

- 32 регистъра за общо предназначение GPR0-GPR31 (32- или 64-разредни). Тази регистри се използват при целочислените операции (инструкции) и за изчисляване на адреси при косвена адресация;
- 32-разреден регистър на условието CR. Битовете на CR отразяват резултатите от някои операции и осигуряват механизъм за тестване и преход при наличие на определено условие. Регистърът съдържа осем 4-разредни полета CR0 - CR7. Полето CR0 (битове CR[0-3]) съдържа флаговете на условие от изпълнението на целочислените инструкции: отрицателен резултат (LT), положителен резултат (GT), нулев резултат (EQ), препълване (SO). Полето CR1 (CR[4-7]) отразява кодовете на условие за операциите с плаваща запетая: възникнало прекъсване (FX) или изключение (FEX), невалидна операция (VX), препълване (OX). Останалите полета CR2 - CR7 съдържат битове на условия, които могат да се дефинират от потребителя;
- регистър XER за индициране на пренос и препълване при целочислените операции и задаване на броя байтове при инструкциите за запис и четене на символни низове;
- регистър LR. Регистърът LR се използва за: задаване на адреса на прехода при някои инструкции за условен преход; съхраняване на адреса на връщане при условни и безусловни преходи и инструкции за свързване, типично използвани за обръщение към подпрограми;
- регистър брояч CTR. Използва се за: задаване на броя итерации на цикли; задаване на адреса на прехода за някои инструкции за условен преход. Подобно на регистъра LR може да бъде използван за извикване на подпрограми на същото привилегировано ниво;
- 32 регистъра 64-разредни за операнди с плаваща запетая FPR0 - FPR31. Използват се от инструкциите с плаваща запетая;
- 32-разреден регистър на състояние и управление при операциите с плаваща запетая FPSCR. Регистърът съдържа всички необходими битове за: сигнализиране на възникнали изключения при операциите с плаваща запетая; разрешение и маскиране на отделни типове изключения; резултата от предходната изпълнена операция с плаваща запетая (битове за състояние на резултата и регистрирано изключение); осигуряване на механизъм за превключване или отключване на устройството за плаваща запетая в/от режим на работа съгласно стандарта IEEE 754; управление на закръгляването;
- два 32-разредни регистъра за отчитане на времето TBU (старши) и TBL (младши). Регистрите са достъпни за приложните програми само за четене.

### 2.2. Формат на инструкциите

Всички инструкции в PowerPC са с еднаква дължина - 4 байта. Операндите за целочислените инструкции могат да бъдат с дължина байт, полудума (16 разреда) и дума (32 разреда), а инструкциите за плаваща запетая могат да оперират с операнди с единична и двойна точност.

### 2.3. Методи за адресация

Ефективният адрес EA за обръщение към паметта се изчислява от устройството за запис и четене и от устройството за предварително извличане. При инструкциите за запис и четене от паметта са възможни три метода за адресация:

- косвена регистрова адресация с непосредствен индекс. Инструкция за зареждане с този тип

адресация има следния вид:

```
lwz     rD, d(rA)
```

където rD е регистър, в който се зарежда операндът, d - 16-разредно отместване (индекс), rA - GPR регистър, съдържащ базов адрес. EA на операнда се изчислява като сума от базовия адрес и знаковоразширеното отместване.

**Примери:**

```
lis     r2, x'3000'     # зарежда адрес 30000100h в регистър GPR2
ori     r2, r2, x'0100'
lbz     r1, 4(r2)       # зарежда байт от адрес 30000100h+4 в GPR1
lhz     r1, 4(r2)       # зарежда 2 байта от адрес 30000100h+4 в GPR1
```

• косвена индексна адресация. Инструкциите за зареждане имат следния вид:

```
lwz     rD, rA, rB
```

където EA на операнда се изчислява като сума от съдържанието на GPR регистрите rA и rB. Индексът се задава чрез GPR регистър, което не налага ограничение на неговия размер.

**Примери:**

```
lis     r2, x'0200'     # зарежда адрес 02000100h в регистър GPR2
ori     r2, r2, x'0100'
li      r3, 4           # зарежда индекс в GPR 3
lbz     r1, r3, r2      # зарежда байт от адрес 02000100h+4 в GPR1
lwz     r1, r3, r2      # зарежда дума от адрес 02000100h+4 в GPR1
```

• косвена адресация. Инструкциите за зареждане имат следния формат:

```
lwz     rD, 0, rB
```

Ефективният адрес на операнда се разполага в GPR регистъра rB.

**Примери:**

```
lis     r2, x'300'      # зарежда адрес 30000100h в регистър GPR2
ori     r2, r2, x'0100'
li      r1, x'3210'     # зарежда 3210 в GPR1
stb     r1, 0, r2       # съхранява младшия байт от GPR1 по адрес 30000100h
stw     r1, 0, r2       # съхранява дума от GPR1 по адрес 30000100h
```

В инструкциите за преход се използват следните методи за адресация:

• относителна. Адресът на прехода се изчислява по отношение на адреса на инструкцията за преход. 24-разредното за безусловните и 14-разредното за условните преходи знаково отместване спрямо този адрес се задава непосредствено в инструкцията. Условните и безусловните преходи, използващи относителна адресация, могат да бъдат с или без връщане, като във втория случай адресът на връщане (адресът на следващата инструкция) се съхранява в регистъра LR;

• абсолютна адресация. Адресът на прехода се задава непосредствено в инструкцията чрез 24-разредно поле за безусловните и 14-разредно поле за условните преходи. Условните и безусловните преходи с този тип адресация също могат да бъдат с или без връщане;

• косвена адресация посредством регистъра LR. Адресът на прехода се извлича от регистъра LR. Използва се при условни преходи, които могат да бъдат с или без връщане;

• косвена адресация посредством регистъра брояч CTR. Адресът на прехода се извлича от регистъра LR.

Инструкциите за преход с връщане се използват за извикване на подпрограма.

**Примери:**

```
b     offset     # относителен (+/- offset) безусловен преход без връщане
bl    offset     # относителен (+/- offset) безусловен преход с връщане чрез регистър LR
ba    target     # абсолютен преход към target без връщане
bla   target     # абсолютен преход към target с връщане чрез регистър LR
bnectr           # условен преход към адрес в CTR без връщане
bnectrl          # условен преход към адрес в CTR с връщане
bnelr            # условен преход към адрес в LR без връщане
bnelrl           # условен преход към адрес в LR с връщане
```

### 2.4. Типове инструкции

#### 2.4.1. Инструкции за работа с операнди с фиксирана запетая

Инструкциите с фиксирана запетая не използват операнди от паметта, а само от GPR регистрите и поместват резултата в GPR регистър. Размерът на операнда се задава явно за всяка инструкция. Възможно е и използване на непосредствен операнд, което се указва чрез суфикс "**i**" в името на инструкцията. В групата на целочислените инструкции се отнасят: аритметични инструкции (събиране - **add**, изваждане - **subf**, умножение - **mull**, деление - **div**, отрицание - **neg**), логически инструкции (**and**, **or**, **xor**, **nand**, **nor**), инструкции за сравнение (аритметично - **cmp**, логическо - **cmpl**), инструкции за преместване и ротация; инструкции за модифициране или четене на съдържанието на регистъра XER и обновяване на полетата на регистъра на условията CR. По премълчаване инструкциите оперират със знакови операнди, а беззнаковите операции се задават явно чрез суфикс "**u**" в името им. Например **divw** - деление на 32-разредни операнди със знак, а **divwu** - деление на 32-разредни операнди без знак. Явно се задава и установяването на кодовете за резултат в полето CR0 на регистъра CR като името на инструкцията завършва с точка ("."). Например **add** не установява, а **add.** установява битовете на CR0 съгласно характера на резултата.

Инструкциите за сравнение задават полето в CR, където да се отрази резултатът от сравнението:

```
cmp crfD, rA, rB
```

Стойността в GPR регистър rA се сравнява със значението на GPR регистър rB, а резултатът се записва в полето на CR, зададено с crfD.

Характерно за операциите ротация и преместване е възможността за задаване на маска на резултата. Изходният операнд от GPR регистър се премества на зададен брой позиции и резултатът се помества в регистъра на резултата под управлението на генерирана маска. Два са възможните способи за използване на маската: в регистъра на резултата се записват само тези разреди, на които в маската съответства "1", а останалите остават непроменени или се изпълнява операцията AND над преместения операнд и маската. Например:

```
rlwnm   rA, rS, rB, MB, ME
rlwnm.  rA, rS, rB, MB, ME
```

Съдържанието на rS се премества наляво на зададения в младшите 5 бита на rB брой разреди. Маската се формира като всички разреди, започвайки от номер MB до номер ME се установяват в 1, а останалите в 0. Над преместения операнд и маската се изпълнява операцията AND и резултатът се помества в регистър rA. Втората инструкция предизвиква актуализация на CR с типа на резултата.

Операцията ротация е възможна само наляво, а преместване - наляво и надясно.

#### 2.4.2. Инструкции за запис и четене.

Целочислените инструкции за запис и четене от паметта могат да бъдат:

• четене на един операнд от паметта в GPR регистър. Това са група инструкции, които зареждат байт (**b**), полудума (**h**), дума (**w**) или дълга дума (**d**) от паметта в зададен GPR регистър. Ефективният адрес се изчислява чрез един от изброените в т. 2.3. методи за адресация на паметта на базата на зададен в инаструкцията GPR регистър (rA във формата от т. 2.3.). Много инструкции за четене задават обновяване на rA с изчисления EA. В мнемониката това се отразява чрез добавяне на символа "**u**". При операнд с по-малка дължина останалите разреди на регистъра приемник се оставят: непроменени; нулират се (символ "**z**" в мнемониката) или се запълват с най-старшия разред на зареждания операнд (алгебрични инструкции, обозначавани със символ "**a**" в мнемониката). Например:

`lhzu rD, d(rA)` - зарежда в младшите 16 разреда на регистър rD операнд от паметта с дължина полудума и нулира останалите му разреди. Регистър rA се обновява с ефективния адрес на операнда.

`lha rD, d(rA)` - зарежда в младшите 16 разреда на регистър rD операнд от паметта с дължина полудума. В останалите разреди на rD се копира най-старшият разред на заредения операнд. В края на операцията регистър rA остава непроменен.

Алгебричните инструкции и инструкциите с обновяване на адресния регистър изискват по-

голямо време за изпълнение.

• запис на операнд от GPR регистър в паметта. Инструкциите за запис съхраняват байт, полудума, дума или дълга дума от съдържанието на GPR регистър в паметта. Много от тези инструкции след изпълнение на записа обновяват адресния регистър, като го зареждат с ефективния адрес.

**Пример:**

`stbu rS, d(rA)` - съдържанието на най-младшия байт на регистъра rS се съхранява в паметта по адрес EA = [rA]+d. В регистър rA се зарежда EA.

• запис/четене с реверсиране на байтовете, с което се променя способът за подреждане на байтовете - от little-endian в big-endian и обратно;

• запис/четене на множество думи между GPR регистри и паметта (**lmw** и **stmw**). Началният адрес в паметта на последователността от думи трябва да бъде изравнен на границата на дума.

**Пример:**

`lmw rD, d(rA)` - зарежда в регистрите GPR, започвайки от rD, последователност от n=32 - rD думи.

`stmw rS, d(rA)` - съхранява в паметта GPR регистрите, започвайки от регистър с номер rS, последователност от n=32 - rS думи.

• четене/запис на символни низове между GPR регистри и паметта. Тези инструкции позволяват преместване на данни от паметта в регистри или обратно, от регистри в паметта, без изискване за изравняване;

• инструкции за подреждане (синхронизиране) на достъпа до паметта. Използват се за управление на реда на изпълнение на обръщенията към паметта по шината на МП. С тяхна помощ се реализират неделими (атомарни) обръщения към паметта от типа на четене-модификация-запис.

Процесорите PowerPC не изпълняват обръщенията към паметта задължително по реда, зададен в програмата. Например инструкция за четене на данни, вече намиращи се в кеш паметта, съвсем естествено се изпълнява по-бързо от също такава инструкция за четене от паметта, независимо че последната може да бъде по-напред в програмата. При необходимост от стриктно подреждане на обръщенията към паметта се използват инструкциите **sync** и **eieio**. Изпълнението на **sync** изисква изчакване на завършването на всички инструкции преди нея, а също и всички започнати предавания по шината. При това инструкциите след **sync** не се стартират до нейното завършване. Действието на **eieio** предизвиква маркиране на всички предшестващи я операции за запис за изпълнение преди всички операции за запис, намиращи се след нея. Ефектът се състои в подреждане на параметрите и командите, предавани към входно-изходните устройства, за да се гарантира последователността, която изисква външната логика.

Инструкциите за зареждане на числа с плаваща запетая работят с операнди с единична и двойна точност. Тъй като FPR регистрите поддържат само формат с двойна точност, то преди зареждането си в регистър операндите с единична точност се преобразуват във формат с двойна точност.

**Пример:**

`lfs frD, d(rA)` - зарежда в FPR регистъра rD операнд с единична точност, който преди зареждането си се преобразува във формат с двойна точност.

`lfd frD, d(rA)` - зарежда в регистър frD операнд с двойна точност.

Инструкциите за запис на числа с плаваща запетая работят с три базови формата - единична точност, двойна точност и цели числа. Форматът с цели числа се поддържа само от инструкцията **stfiwx**.

**Пример:**

`stfs frS, d(rA)` - съдържанието на регистъра frS се преобразува във формат с единична точност и се съхранява в паметта.

`stfd frS, d(rA)` - съхранява в две последователни думи в паметта съдържанието на регистъра frS.

### 2.4.3. Инструкции за работа с числа с плаваща запетая.

Инструкциите с плаваща запетая задават следните типове операции: аритметични, умножение-събиране, закръгляване и преобразуване, сравнение, действия с регистъра за състояние и управление FPSCR, прехвърляне. Аритметичните инструкции за плаваща запетая задават събиране (**fadd**),

изваждане (**fsub**), умножение (**fmul**), деление (**fdiv**), квадратен корен (**fsqrt**), комбинирани операции умножение/събиране (**fmadd**) и др. с двойна или единична точност (суфикс "s"). Например:

```
fadd    frD, frA, frB
fadd.   frD, frA, frB
```

Операндът в регистър frA се събира с операнда в регистър frB. Ако най-старшият бит на резултата не е 1, то той е нормализиран. Резултатът се закръгля до зададената точност съгласно управляващото поле RN на регистъра FPSCR и се записва в регистър frD.

Комбинираните операции умножение/събиране **fmadd frD, frA, frC, frB** изпълняват умножение на два операнда (frA и frC) и към междинното произведение без закръгляване събират (изваждат) трети операнд (frB). Кодовете на условие се установяват за крайния резултат - в края на втората операция.

Инструкциите за закръгляване и преобразуване се използват за отсичане на 64 разредно число с двойна точност до 32 разредно число с единична точност (**frsp**) или преобразуване на число с двойна точност в 32-разредно цяло число със знак (**fctiw**).

Инструкциите за достъп до регистъра за състояние и управление на устройството с плаваща запетая FPSCR позволяват: копиране на съдържанието на FPSCR в зададен FPR регистър; копиране на поле от FPSCR в зададено поле в CR; записване на непосредствено значение в поле на FPSCR; нулиране или установяване на определен бит в FPSCR; копиране на поле от FPR в зададено поле на FPSCR.

Инструкциите за прехвърляне позволяват копиране на данни от един FPR регистър в друг, като при това е възможна модификация на знаковия разред на данните.

```
fmr     frD, frB  -  прехвърля съдържанието на регистър frB в регистър frD
fmr.    frD, frB  -  прехвърля съдържанието на регистър frB в регистър frD с
                     разрешение за актуализация на CR.
fneg    frD, frB  -  прехвърля съдържанието на регистър frB в регистър frD
                     с инвертиране на знаковия разред (бит 0)
```

Инструкциите за сравнение на операнди с плаваща запетая сравняват съдържанието на два FPR регистъра като се установява съответния разред в CR и полето FPCC на FPSCR (битове 16-19):

```
fcmpu   crfD, frA, frB  -  операндът в frA се сравнява с операнда в frB и резултатът
                           от сравнението се помества в crfD и FPCC.
```

### 2.4.4. Инструкции за преход и за работа с ресистъра на състояние

Инструкциите за безусловен преход могат да задават следните видове преходи: относителен (**b**), относителен с връщане (**bl**), абсолютен (**ba**), абсолютен с връщане (**bla**).

Инструкциите за условен преход променят посоката на изпълнение на програмата в зависимост от значението на битове от CR и имат следния общ формат:

```
bc BO, BI, addr
```

където полето BO задава условието на прехода (true или false), въздействието върху съдържанието на CTR (без промяна или декрементиране), въздействието на значението на CTR върху посоката на прехода и информация за предсказване на прехода. Полето BI задава номера на проверявания бит от регистъра CR, а полето addr се използва за определяне адреса на прехода. Ако проверяваното условие е блокирано от предходна, незавършила изпълнението си инструкция, то преходът се предсказва с използване на най-младшия бит в BO или на базата на динамично предсказване. Динамичното предсказване се извършва на базата на буфер за историята на преходите (BTB), но не е реализирано във всички МП на фамилията PowerPC. Условните преходи могат да бъдат относителни (**bc**), относителни с връщане (**bcl**), абсолютни (**bca**), абсолютни с връщане (**bcla**), към адрес в LR (**bclr**), към адрес в LR с връщане (**bclrl**), към адрес в CTR (**bcctr**), към адрес в CTR с връщане (**bcctrl**).

Инструкциите за работа с регистъра на състояние задават преместване на полетата (**mcrf**) и логически операции над неговото съдържание. Например:

```
crand   crbD, crbA, crbB  -  бит crbD приема значение логическата сума на бит
                             crbA и бит crbB.
```

Аналогично се реализират операциите OR (**cror**), XOR (**crxor**) и др.

В архитектурата PowerPC е предвидена специална инструкция за обръщение към операционната система за определени услуги - System Call (**cs**). Тази инструкция не е привилегирована, но предизвиква

прекъсване, което се обработва от ОС.

Инструкциите за генериране на прекъсване (**trap**) се използват за тестване на зададено множество условия. Ако някое от тези условия се изпълни, се генерира прекъсване. В противен случай изпълнението на инструкциите следва нормалния си ход.

### 2.4.5. Инструкции за управление на кеша.

В архитектурата PowerPC са включени инструкции за поддържане консистентността на кеш паметите за данни: нулиране на единичен блок в кеш паметта - **dcbz**; съхранение на блок от кеш паметта в основната памет **dcbst** и **dcbf**; четене на блок в кеш паметта за последващо използване от програмата **dcbt** с цел оптимизация на изпълнението. Програмно може да се поддържа кохерентност на кеш паметите за инструкции при самомодифициращ се код: **dcbst** - обновяване на паметта; **sync** - изчакване за обновяване; **icbi** - обявяване за невалиден блок в кеша за инструкции; **isync** - синхронизация.

## 3. Системна архитектура

В архитектурата PowerPC са дефинирани два режима на работа - потребителски и системен. Автоматично превключване в системен режим се извършва при възникване на прекъсване. В него са достъпни всички ресурси на МП - регистри и инструкции. Множеството системни регистри включва:

• регистър за състояние на процесора - MSR (Machine State Register). Разредите в MSR отразяват текущите операционни характеристики на процесора. Основните флагове в тази регистър задават: режим на работа (потребителски или системен); разрешение или забрана на прекъсвания от различни източници - външни, от схемите за контрол, при трасировка, вложени прекъсвания; режим на прекъсване при плаваща запетая; разрешение или забрана на FPU, на виртуалната адресация съответно за инструкции и за данни; разположение на таблицата с векторите на прекъсване - в началото или края на линейното адресно пространство; начина на разполагане на данните в паметта - little или big Endian и др. Съдържанието на MSR може да бъде тествано с инструкцията **mfmsr** (четене на съдържанието на MSR в регистър GPR) и променяно с **mtmsr** (запис в MSR). Неявно MSR се променя от инструкциите **rfi** (връщане от прекъсване) и **sc** (системно извикване);

• регистри за управление на виртуалната памет. Регистрите за управление на паметта включват: осемте BAT регистъра за дефиниране на области памет; регистъра за задаване на физическия адрес и размера на таблицата на страниците SDR1; шестнадесетте сегментни регистъра SR0 - SR15, съдържащи сегментните дескриптори;

• регистри, използвани при обработка на прекъсванията. Тази група регистри включва: регистър DAR за съхранение на адреса за обръщение към паметта, при който се е генерирало изключение; регистър DSISR за съхранение на причината, предизвикала изключение при обръщение към паметта; регистри SRR0 и SRR1, в които автоматично се съхраняват и възстановяват адресът на връщане и MSR при възникнало прекъсване или изключение; регистри със специално предназначение SPRG0 - SPRG3, предоставени за използване от ОС;

• други системни регистри. В тази група се включват: брояч за отмерване на квант от време DEC; регистрите за отчитане на времето TBU и TBL; регистър PIR, използван от ОС за зареждане на идентификатор на процесора; регистър VER, съхраняващ модела и версията на МП и др.

Промяната на съдържанието на системните регистри може да се извърши само в системен режим.

**Въпроси и задачи:**

1. Какви са различията между скаларната архитектура на PowerPC и на Intel?

2. Какви са основните собености на системата инструкции на PowerPC? Кои инструкции изпълняват обръщение към паметта и какви са възможните методи за адресация? Какви са особеностите в изпълнението на условни и безусловни преходи?

3. Каква е ролята на регистрите LR и CTR за осигуряване на ритмичността на конвейера на инструкциите? Какви са особеностите за работа с регистъра за кодовете на условие CR?

4. Какви са механизмите за виртуализация на паметта? Осъществява ли се и как защита по привилегии?

5. Напишете програма за PowerPC за умножение на елементите на масив с константа. По колко инструкции от вашата програма ще се стартират за изпълнение на всеки такт? Колко такта ще заеме изпълнението на програмата при 10 елементен масив?

# ПРИЛОЖЕНИЕ 1

## Листинг на програмата за превключване от реален в защитен режим

```
Turbo Assembler    Version 2.01        10/19/98 08:38:08        Page 1
pmtut.ASM

                                ;*************************************************************************
                                ;* Пример за програма реализираща превключване от                        *
                                ;* Реален в Защитен режим.                                               *
                                ;*                                                                       *
                                ;*************************************************************************

                                ideal
                                P386

                                ;-------------------------------------------------------------------------

          = 0100        STACK16_SIZE  =     100h          ; размер на стека за Реален Режим
          = 0100        STACK32_SIZE  =     100h          ; размер на стека за Защитен Режим

00000000                        struc segment_descriptor
00000000 01*(????)               seg_length0_15  dw   ?   ; младша дума на дължината на сегмента
00000002 01*(????)               base_addr0_15   dw   ?   ; младша дума на базов  адрес
00000004 01*(??)                 base_addr16_23  db   ?   ; младши байт на старшата дума на базовия адрес.
00000005 01*(??)                 flags           db   ?   ; тип на сегмента и флагове за присъствие.
00000006 01*(??)                 access          db   ?   ; старша част на дължината на сегмента.
                                                         ; флагове за достъп
00000007 01*(??)                 base_addr24_31  db   ?   ; старши байт на базов  адрес
00000008                        ends segment_descriptor

00000000                        struc interrupt_descriptor
00000000 01*(????)               offset0_15      dw   ?   ; младша дума на указателя на отместването
00000002 01*(????)               selector0_15    dw   ?   ; сегмент на селектора
00000004 01*(00)                 zero_byte       db   0   ; неизползван в този дескриптор формат
00000005 01*(??)                 flags           db   ?   ; байт  флагове
00000006 01*(????)               offset16_31     dw   ?   ; старша дума на отместването
00000008                        ends interrupt_descriptor

                                ;*************************************************************************

0000                    segment code16   para public use16   ; този сегмент съдържа целия 16-битов
                        assume  cs:code16, ds:code16         ; код и необходими данни

                                ;-------------------------------------------------------------------------

0000 0100*(??)          stack16  db     STACK16_SIZE dup (?)  ; 16-битов стек за Реален Режим.
0100                    label   stack16_end     word

0100 03FF 0000 0000     idt_real  dw    3ffh,0,0             ; IDT за Реален Режим


                                ;-------------------------------------------------------------------------
                                ; Главна процедура, това е входната точка

0106                    start16:
0106 8C C8                      mov     ax,cs                ; Зарежда кодов сегмент в DS и ES
0108 8E D8                      mov     ds,ax
010A 8E C0                      mov     es,ax
010C FA                         cli                          ; забрана на прекъсванията докато
010D 8E D0                      mov     ss,ax                ; се установяват SS и SP
010F BC 0100r           mov     sp,offset stack16_end
Turbo Assembler    Version 2.01        10/19/98 08:38:08        Page 2
pmtut.ASM

0112 FB                         sti                          ; разрешение на прекъсванията

0113 66| B8 00000000    mov     eax,0                        ; изчистване на EAX
0119 B8 0000s           mov     ax,code16                    ; зарежда кодовия сегмент в AX
011C 66| C1 E0 04               shl     eax,4                ; прави го физически адрес
0120 67| A3 00000128r   mov     [ds:code16_descriptor.base_addr0_15],ax   ; съхранява го в дескриптор
0126 67| A3 00000130r   mov     [ds:data16_descriptor.base_addr0_15],ax
012C 66| C1 E8 08               shr     eax,8
0130 67| 88 25 0000012Ar        mov     [ds:code16_descriptor.base_addr16_23],ah
0137 67| 88 25 00000132r        mov     [ds:data16_descriptor.base_addr16_23],ah
```

```
013E 66| B8 00000000          mov   eax,0                          ; изчистване на EAX
0144 B8 0000s                  mov   ax,code32                      ; зарежда 32-битовия кодов сегмент в AX
0147 66| C1 E0 04                    shl   eax,4                    ; прави го физически
014B 67| A3 00000110r          mov   [ds:code32_descriptor.base_addr0_15],ax      ; съхраняване в дескриптора
0151 67| A3 00000118r          mov   [ds:data32_descriptor.base_addr0_15],ax
0157 66| C1 E8 08                    shr   eax,8
015B 67| 88 25 00000112r             mov   [ds:code32_descriptor.base_addr16_23],ah
0162 67| 88 25 0000011Ar             mov   [ds:data32_descriptor.base_addr16_23],ah

0169 66| B8 00000000          mov   eax,0                          ; изчистване на EAX
016F B8 0000s                  mov   ax,code32                      ; зарежда 32-битовия кодов сегмент в AX
0172 66| C1 E0 04                    shl   eax,4                    ; прави го физически
0176 66| 05 00000106r          add   eax,offset dummy_descriptor          ; изчислява физическия адрес на GDT
017C 66| 67| A3 00000102r             mov   [dword ds:gdt_start+2],eax

0183 66| B8 00000000          mov   eax,0                          ; изчистване на EAX
0189 B8 0000s                  mov   ax,code32                      ; зарежда 32-битов кодов сегмент в AX
018C 66| C1 E0 04                    shl   eax,4                    ; създава физически адрес
0190 66| 05 0000013Cr          add   eax,offset interrupt_0         ; изчислява физически адрес на IDT
0196 66| 67| A3 00000138r             mov   [dword ds:idt_start+2],eax


019D FA                              cli                            ; забранява прекъсванията
019E 67| 0F 01 15          +         lgdt  [fword ds:global_descriptor_table]  ; зарежда регистър GDT
        00000100r
01A6 67| 0F 01 1D          +         lidt  [fword ds:interrupt_descriptor_table] ; зарежда регистър IDT
        00000136r
01AE 0F 20   C0                      mov   eax,cr0                  ; взема CR0 в EAX
01B1 0C 01                           or    al,1                     ; установява бита за Защитен Режим
01B3 0F 22   C0                      mov   cr0,eax                  ; след това вече сме в  Защитен Режим!
01B6 EA                              db    0eah                     ; КОП за far jump (за да установи правилно CS)
01B7 0144r   0008                    dw    offset   start32,code32_idx

01BB                 exit16:         ; Връщане от Защитен режим
01BB 0F 20   C0                      mov   eax,cr0                  ; Взема CR0 в EAX
01BE 24 FE                           and   al,not  1                ; изчиства бита за Защитен Режим
01C0 0F 22   C0                      mov   cr0,eax                  ; след което отново се връщаме в Реален Режим!
01C3 EA                              db    0eah
01C4 01C8r   0000s                   dw    offset   flush_ipq,code16
01C8                 flush_ipq:
01C8 8C C8                           mov   ax,cs                    ; възстановяване на важните регистри
01CA 8E D0                           mov   ss,ax
01CC BC 0100r                        mov   sp,offset stack16_end
01CF 8E D8                           mov   ds,ax
01D1 8E C0                           mov   es,ax
01D3 0F 01   1E 0100r          lidt  [fword idt_real]
```



```
01D8 FB                              sti                            ; разрешаване на прекъсванията

01D9 B8 4C00                   mov   ax,4c00h                       ; всичко е наред, и излизаме в MSDOS с код 0
01DC CD 21                           int   21h                      ;

                               ;-----------------------------------------------------------------------

01DE                 ends      code16

00000000                       segment code32  para public use32          ; този сегмент съдържа целия 32-битов
                               assume  cs:code32, ds:code32               ; код и необходимите данни

00000000 0100*(??)             stack32  db     STACK32_SIZE dup (?)       ; 32-битов стек
00000100                       label   stack32_end     dword

                               ;-----------------------------------------------------------------------

00000100                       label global_descriptor_table fword  ; тук започва GDT

00000100 0030 0000 0000   gdt_start          dw                gdt_size,0,0        ; стойност за рег.GDT
00000106 0000 0000 00 00 00 +  dummy_descriptor  segment_descriptor <0,0,0,0,0,0>
         00
0000010E FFFF 0000 00 9A CF +  code32_descriptor segment_descriptor <0ffffh,0,0,9ah,0cfh,0> ; 4GB 32-битов код
         00
00000116 FFFF 0000 00 92 CF +  data32_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0> ; 4GB 32-битови данни
         00
0000011E FFFF 0000 00 92 CF +  core32_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0> ; 4GB 32-битови данни
         00
00000126 FFFF 0000 00 9A 00 +  code16_descriptor segment_descriptor <0ffffh,0,0,9ah,0,0>  ;  64k 16-битов код
         00
0000012E FFFF 0000 00 92 00 +  data16_descriptor segment_descriptor <0ffffh,0,0,92h,0,0>  ;  64k 16-битови данни
         00
```

```
147
148                 = CODE32:0030                   gdt_size=$-(offset dummy_descriptor)
149
150                 = 0008          code32_idx      =       08h             ; отместване на 32-битов кодов  сегмент в GDT
151                 = 0010          data32_idx      =       10h             ; отместване на 32-битов данков сегмент в GDT
152                 = 0018          core32_idx      =       18h             ; отместване на 32-битов данков сегмент в GDT
153                 = 0020          code16_idx      =       20h             ; отместване на 16-битов кодов  сегмент в GDT
154                 = 0028          data16_idx      =       28h             ; отместване на 16-битов данков сегмент в GDT
155
156     00000136                        label interrupt_descriptor_table fword          ; тук започва IDT
157
158     00000136  0008 0000 0000   idt_start dw                         idt_size,0,0
159     0000013C  0163r  0008 00  8E  +  interrupt_0     interrupt_descriptor    <demo_int,code32_idx,0,8eh,0>
160               0000
161
162                 = CODE32:0008                   idt_size=$-(offset interrupt_0)
163
164                             ;------------------------------------------------------------------------
165
166     00000144                        start32:        ; тук започва Защитения Режим
167     00000144  66| B8 0010                  mov     ax,data32_idx           ; зарежда нужните регистри с подходящите
168     00000148  8E D0                         mov     ss,ax                   ; селектори
169     0000014A  BC 00000100r                  mov     esp,offset stack32_end  ; размер на стека
170     0000014F  8E D8                         mov     ds,ax
171     00000151  8E C0                         mov     es,ax
Turbo Assembler     Version 2.01        10/19/98 08:38:08       Page 4
pmtut.ASM

172     00000153  8E E0                         mov     fs,ax
173     00000155  8E E8                         mov     gs,ax
174
175     00000157  E8 0000000D                   call    main                    ; сега  всичко е наред за стартиране
176                                                                             ; на main
177     0000015C  EA                            db      0eah    ; far jump opcode ; когато main приключи, се връщаме обратно
178     0000015D  01BBr 0000 0020               dw      offset  exit16,0,code16_idx     ; към Реален Режим
179
180                             ;------------------------------------------------------------------------
181
182     00000163                        proc    demo_int
183     00000163  B8 0000000A                   mov     eax,10
184     00000168  CF                            iretd
185     00000169                        endp    demo_int
186
187                             ;------------------------------------------------------------------------
188                             ; главна процедура за Защитен Режим
189
190     00000169                        main:
191     00000169  B8 0000000A                   mov     eax,10                  ; ...някакви действия...
192     0000016E  C3                            ret                             ; ...и връщане
193
194                             ;------------------------------------------------------------------------
195
196     0000016F                        ends    code32
197
198                                     ;*******************************************************************
199
200                             end start16
```

# ПРИЛОЖЕНИЕ 2

## Листинг на програмата за превключване на задачи

```
Turbo Assembler    Version 2.01        10/30/98 13:27:07          Page 1
pmswitch.ASM

                                ;*************************************************************************
                                ;* Пример за програма реализираща превключване от    *
                                ;* Реален в Защитен режим. След което се реализира просто превключване на  *
                                ;* задачи.                                                                 *
                                ;*************************************************************************


                                ideal
                                P386

                                ;-------------------------------------------------------------------------

          = 0100                STACK16_SIZE  =     100h          ; размер на стека за Реален Режим
          = 0400                STACK32_SIZE  =     400h          ; размер на стека за Защитен Режим

00000000                        struc segment_descriptor
00000000 01*(????)              seg_length0_15      dw    ?       ; младша дума на дължината на сегмента
00000002 01*(????)              base_addr0_15       dw    ?       ; младша дума на базов адрес
00000004 01*(??)                base_addr16_23      db    ?       ; младши байт на старшата дума на базовия адрес.
00000005 01*(??)                flags               db    ?       ; тип на сегмента и флагове за присъствие
00000006 01*(??)                access              db    ?       ; старша част на дължината на сегмента
                                                                  ; флагове за достъп
00000007 01*(??)                base_addr24_31      db    ?       ; старши байт на базов  адрес
00000008                        ends segment_descriptor

00000000                        struc interrupt_descriptor
00000000 01*(????)              offset0_15          dw    ?       ; младша дума на указателя на отместването
00000002 01*(????)              selector0_15        dw    ?       ; сегмент на селектора
00000004 01*(00)                zero_byte           db    0       ; неизползван в този дескриптор формат
00000005 01*(??)                flags               db    ?       ; байт  флагове
00000006 01*(????)              offset16_31         dw    ?       ; старша дума на отместването
00000008                        ends interrupt_descriptor


                                ;*************************************************************************


0000                            segment code16   para public use16      ; този сегмент съдържа целия 16-битов
                                assume   cs:code16, ds:code16           ; код и необходими данни

                                ;-------------------------------------------------------------------------

0000 0100*(??)                  stack16  db      STACK16_SIZE dup (?)   ; 16-битов стек за Реален Режим
0100                            label    stack16_end      word

0100 03FF 0000 0000             idt_real                  dw     3ffh,0,0  ; IDT за Реален Режим


                                ;-------------------------------------------------------------------------

0106                            proc     err16exit
0106 B4 09                               mov      ah,9                   ; функция на DOS за отпечатване на низ
0108 CD 21                               int      21h                    ; отпечатване на съобщението
010A B8 4CFF                             mov      ax,4cffh               ; изход с код 0ffh
010D CD 21                               int      21h                    ;
010F                            endp     err16exit

Turbo Assembler    Version 2.01        10/30/98 13:27:07          Page 2
pmswitch.ASM

                                ;-------------------------------------------------------------------------
                                ;B:      DS:SI -   указател към низа

010F                            proc     write_msg
010F 50 56      57 06                    push     ax si di es
0113 B8 B800                             mov      ax,0b800h              ; сегмент на текстовия екран
0116 8E C0                               mov      es,ax                  ; взема го в ES
0118 8B 44      02                       mov      ax,[si+2]              ; взема позиция Y
011B BF 00A0                             mov      di,160
011E F7 E7                               mul      di
```

```
0120  03 04                    add     ax,[si]
0122  8B F8                    mov     di,ax
0124  8A 64     04             mov     ah,[si+4]          ; получава байта с атрибута
0127  83 C6     05             add     si,5
012A                   write_loop_pm:
012A  8A 04                    mov     al,[si]
012C  0A C0                    or      al,al              ; край  на стрингa?
012E  74 08                    jz      loop_end_pm
0130  46                       inc     si
0131  26: 89 05                mov     [es:di],ax
0134  47                       inc     di
0135  47                       inc     di
0136  EB F2                    jmp     write_loop_pm
0138                   loop_end_pm:
0138  07 5F     5E 58          pop     es di si ax
013C  C3                       ret
013D                   endp    write_msg

                               ;------------------------------------------------------------------------
                               ; Главна процедура, това е входната точка

013D  00 00     00 00 1F 4E 6F+ rm_msg          db      0,0,0,0,1fh,'Now in Real Mode - press a key to switch '
              77 20   69 6E 20 52 65+
              61 6C   20 4D 6F 64 65+
              20 2D   20 70 72 65 73+
              73 20   61 20 6B 65 79+
              20 74   6F 20 73 77 69+
              74 63   68 20
016B  74 6F     20 50 72 6F 74+                 db      'to Protected Mode!',0
              65 63   74 65 64 20 4D+
              6F 64   65 21 00
017E  00 00     04 00 1F 42 61+ rm2_msg         db      0,0,4,0,1fh,'Back in Real Mode - press a key to return
              63 6B   20 69 6E 20 52+
              65 61   6C 20 4D 6F 64+
              65 20   2D 20 70 72 65+
              73 73   20 61 20 6B 65+
              79 20   74 6F 20 72 65+
              74 75   72 6E 20
01AD  74 6F     20 44 4F 53 21+                 db      'to DOS!',0
              00

01B5                   start16:
01B5  8C C8                    mov     ax,cs              ; Зарежда кодов сегмент в DS и  ES
01B7  8E D8                    mov     ds,ax
01B9  8E C0                    mov     es,ax
01BB  FA                       cli                        ; забрана на прекъсванията докато
```



```
01BC  8E D0                    mov     ss,ax              ; се установяват SS и SP
01BE  BC 0100r                 mov     sp,offset stack16_end
01C1  FB                       sti                        ; разрешение на прекъсванията

                               ;16-битов кодов и данков сегмент
01C2  66| B8 00000000          mov     eax,0              ; изчистване на EAX
01C8  B8 0000s                 mov     ax,code16          ; зарежда кодовия сегмент в AX
01CB  66| C1 E0 04             shl     eax,4              ; прави го физически адрес
01CF  67| A3 00000428r         mov     [ds:code16_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
01D5  67| A3 00000430r         mov     [ds:data16_descriptor.base_addr0_15],ax
01DB  66| C1 E8 08             shr     eax,8
01DF  67| 88 25 0000042Ar      mov     [ds:code16_descriptor.base_addr16_23],ah
01E6  67| 88 25 00000432r      mov     [ds:data16_descriptor.base_addr16_23],ah

                               ;32-битов кодов и данков сегмент на старата задача
01ED  66| B8 00000000          mov     eax,0              ; изчистване на EAX
01F3  B8 0000s                 mov     ax,code32          ; зарежда 32-битовия кодов сегмент в AX
01F6  66| C1 E0 04             shl     eax,4              ; прави го физически адрес
01FA  67| A3 00000410r         mov     [ds:code32_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
0200  67| A3 00000418r         mov     [ds:data32_descriptor.base_addr0_15],ax
0206  66| C1 E8 08             shr     eax,8
020A  67| 88 25 00000412r      mov     [ds:code32_descriptor.base_addr16_23],ah
0211  67| 88 25 0000041Ar      mov     [ds:data32_descriptor.base_addr16_23],ah

                               ;стеков сегмент - стара задача
0218  66| B8 00000000          mov     eax,0              ; изчистване на EAX
021E  B8 0000s                 mov     ax,code32          ; зарежда 32-битовия стеков сегмент в AX
0221  66| C1 E0 04             shl     eax,4              ; прави го физически адрес
0225  66| 05 00000000r         add     eax,offset stack32
022B  67| A3 00000458r         mov     [ds:stack32_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
0231  66| C1 E8 08             shr     eax,8
```

```
0235 67| 88 25 0000045Ar        mov     [ds:stack32_descriptor.base_addr16_23],ah

                                ;32-битов кодов и данков сегмент на новата задача
023C 66| B8 00000000            mov     eax,0               ; изчистване на EAX
0242 B8 0000s                   mov     ax,extracode32      ; зарежда 32-битов кодов сегмент в AX
0245 66| C1 E0 04               shl     eax,4               ; прави го физически
0249 67| A3 00000438r           mov     [ds:ecode32_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
024F 67| A3 00000440r           mov     [ds:edata32_descriptor.base_addr0_15],ax
0255 66| C1 E8 08               shr     eax,8
0259 67| 88 25 0000043Ar        mov     [ds:ecode32_descriptor.base_addr16_23],ah
0260 67| 88 25 00000442r        mov     [ds:edata32_descriptor.base_addr16_23],ah

                                ;стеков сегмент - нова  задача
0267 66| B8 00000000            mov     eax,0               ; изчистване на EAX
026D B8 0000s                   mov     ax,extracode32      ; зарежда 32-битовия стеков сегмент в AX
0270 66| C1 E0 04               shl     eax,4               ; прави го физически адрес
0274 66| 05 00000000r           add     eax,offset estack32
027A 67| A3 00000460r           mov     [ds:estack32_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
0280 66| C1 E8 08               shr     eax,8
0284 67| 88 25 00000462r        mov     [ds:estack32_descriptor.base_addr16_23],ah

                                ;TSS на старата задача
028B 66| B8 00000000            mov     eax,0               ; изчистване на EAX
0291 B8 0000s                   mov     ax,TSS_code32       ; зарежда TSS сегмента  в AX
0294 66| C1 E0 04               shl     eax,4               ; прави го физически адрес
0298 67| A3 00000448r           mov     [ds:oldtask_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
```



```
029E 66| C1 E8 08               shr     eax,8
02A2 67| 88 25 0000044Ar        mov     [ds:oldtask_descriptor.base_addr16_23],ah

                                ;TSS на новата  задача
02A9 66| B8 00000000            mov     eax,0               ; изчистване на EAX
02AF B8 0000s                   mov     ax,TSS_extracode32 ; зарежда TSS сегмента на новата задача в AX
02B2 66| C1 E0 04               shl     eax,4               ; прави го физически
02B6 67| A3 00000450r           mov     [ds:newtask_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
02BC 66| C1 E8 08               shr     eax,8
02C0 67| 88 25 00000452r        mov     [ds:newtask_descriptor.base_addr16_23],ah

                                ;стеков сегмент - dpl_0
02C7 66| B8 00000000            mov     eax,0               ; изчистване на EAX
02CD B8 0000s                   mov     ax,dpl_0            ; зарежда стековия сегмент за DPL0 в AX
02D0 66| C1 E0 04               shl     eax,4               ; прави го физически адрес
02D4 67| A3 00000468r           mov     [ds:stack00_descriptor.base_addr0_15],ax ; съхранява го в дескриптор
02DA 66| C1 E8 08               shr     eax,8
02DE 67| 88 25 0000046Ar        mov     [ds:stack00_descriptor.base_addr16_23],ah

02E5 66| B8 00000000            mov     eax,0               ; изчистване на EAX
02EB B8 0000s                   mov     ax,code32           ; зарежда 32-битовия кодов сегмент в AX
02EE 66| C1 E0 04               shl     eax,4               ; прави го физически
02F2 66| 05 00000406r           add     eax,offset dummy_descriptor ; изчислява физическия адрес на GDT
02F8 66| 67| A3 00000402r       mov     [dword  ds:gdt_start+2],eax

02FF 66| B8 00000000            mov     eax,0               ; изчистване на EAX
0305 B8 0000s                   mov     ax,code32           ; зарежда 32-битов кодов сегмент в AX
0308 66| C1 E0 04               shl     eax,4               ; създава физически адрес
030C 66| 05 00000474r           add     eax,offset interrupt_0      ; изчислява физически адрес на    IDT
0312 66| 67| A3 00000470r       mov     [dword  ds:idt_start+2],eax

0319 B8 0003                    mov     ax,3                ; установява текстов режим 3, изчиства екрана
031C CD 10                      int     10h

031E BE 013Dr                   mov     si,offset rm_msg    ; изписва съобщението в Реален Режим
0321 E8 FDEB                    call    write_msg

0324 32 E4                      xor     ah,ah
0326 CD 16                      int     16h

0328 FA                         cli                         ; забранява прекъсванията
0329 67| 0F 01 15          +    lgdt    [fword  ds:global_descriptor_table]      ; зарежда регистър GDT
        00000400r
0331 67| 0F 01 1D          +    lidt    [fword  ds:interrupt_descriptor_table]   ; зарежда регистър IDT
        0000046Er
0339 0F 20      C0              mov     eax,cr0             ; взема CR0 в EAX
033C 0C 01                      or      al,1                ; установява бита за Защитен Режим
033E 0F 22      C0              mov     cr0,eax             ; след това вече сме в Защитен Режим!
0341 EA                         db      0eah                ; КОП за far jump (за да установи правилно CS)
0342 047Cr      0008            dw      offset  start32,code32_idx
```

```
0346                    exit16:                                     ; Връщане от Защитен режим
0346 0F 20      C0              mov     eax,cr0                     ; Взема CR0 в EAX
0349 24 FE                      and     al,not  1                   ; изчиства бита за Защитен Режим
034B 0F 22      C0              mov     cr0,eax                     ; след което отново се Връщаме в Реален Режим!
034E EA                         db      0eah
034F 0353r      0000s           dw      offset  flush_ipq,code16
```



```
0353                    flush_ipq:
0353 8C C8                      mov     ax,cs                       ; Възстановяване на важните регистри
0355 8E D0                      mov     ss,ax
0357 BC 0100r                   mov     sp,offset stack16_end
035A 8E D8                      mov     ds,ax
035C 8E C0                      mov     es,ax
035E 0F 01      1E 0100r        lidt    [fword  idt_real]
0363 FB                         sti                                 ; разрешаване на прекъсванията

0364 BE 017Er                   mov     si,offset rm2_msg ; извежда второто съобщение
0367 E8 FDA5                    call    write_msg

036A 32 E4                      xor     ah,ah                       ; чака  за клавиш
036C CD 16                      int     16h

036E B8 0003                    mov     ax,3                        ; изчиства екрана отново
0371 CD 10                      int     10h

0373 B8 4C00                    mov     ax,4c00h                    ; излизане в MSDOS с код 0
0376 CD 21                      int     21h                         ;

                                ;-----------------------------------------------------------------------

0378                    ends    code16

                                ;*****************************************************************************
00000000                        segment code32  para public use32           ; този сегмент  съдържа целия 32-битов
                                assume  cs:code32, ds:code32                ; код и необходимите данни

00000000 0400*(??)              stack32 db      STACK32_SIZE dup (?)    ; 32-битов стек
00000400                        label   stack32_end     dword

                                ;-----------------------------------------------------------------------
                                                        align 16
00000400                        label global_descriptor_table fword         ; тук започва GDT

00000400 0068 0000 0000 gdt_start       dw              gdt_size,0,0        ;стойност за рег.GDT
00000406 0000 0000 00 00 00 +   dummy_descriptor  segment_descriptor <0,0,0,0,0,0>
         00
0000040E FFFF 0000 00 9A CF +   code32_descriptor segment_descriptor <0ffffh,0,0,9ah,0cfh,0> ; 4GB 32-битов код
         00
00000416 FFFF 0000 00 92 CF +   data32_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0> ;4GB 32-битови данни
         00
0000041E FFFF 0000 00 92 CF +   core32_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0> ;4GB 32-битови данни
         00
00000426 FFFF 0000 00 9A 00 +   code16_descriptor segment_descriptor <0ffffh,0,0,9ah,0,0>   ;64k 16-битов код
         00
0000042E FFFF 0000 00 92 00 +   data16_descriptor segment_descriptor <0ffffh,0,0,92h,0,0>   ;64k 16-битови данни
         00
00000436 FFFF 0000 00 9A CF +   ecode32_descriptor segment_descriptor <0ffffh,0,0,9ah,0cfh,0> ;4GB 32-битов код
         00
0000043E FFFF 0000 00 92 CF +   edata32_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0> ;4GB 32-битови данни
         00
00000446 00E6 0000 00 89 C0 +   oldtask_descriptor segment_descriptor <000e6h,0,0,10001001b,0c0h,0> ; дескриптор към задача
         00
0000044E 00E6 0000 00 89 C0 +   newtask_descriptor segment_descriptor <000e6h,0,0,10001001b,0c0h,0> ; дескриптор към задача
         00
```



```
00000456 FFFF 0000 00 92 CF +   stack32_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0>  ; дескриптор на  стек
         00
0000045E FFFF 0000 00 92 CF +   estack32_descriptor segment_descriptor      <0ffffh,0,0,92h,0cfh,0>     ; дескриптор на  стек
         00
00000466 FFFF 0000 00 92 CF +   stack00_descriptor segment_descriptor <0ffffh,0,0,92h,0cfh,0>  ; дескриптор на  стек DPL 0
         00

         = CODE32:0068                  gdt_size=$-(offset dummy_descriptor)

         = 0008         code32_idx      =       08h             ; отместване на 32-битов кодов  сегмент в GDT
         = 0010         data32_idx      =       10h             ; отместване на 32-битов данков сегмент в GDT
```

```
297             = 0018          core32_idx      =       18h             ; отместване на 32-битов данков сеграмент в GDT
298             = 0020          code16_idx      =       20h             ; отместване на 16-битов кодов  сеграмент в GDT
299             = 0028          data16_idx      =       28h             ; отместване на 16-битов данков сеграмент в GDT
300             = 0030          ecode32_idx     =       30h             ; отместване на 32-битов кодов  сеграмент в GDT
301             = 0038          edata32_idx     =       38h             ; отместване на 32-битов данков сеграмент в GDT
302             = 0040          oldtask_idx     =       40h             ; отместване на TSS текуща задача
303             = 0048          newtask_idx     =       48h             ; отместване на TSS нова задача
304             = 0050          stack32_idx     =       50h             ; отместване на стека на старата задача
305             = 0058          estack32_idx    =       58h             ; отместване на стека на новата задача
306             = 0060          stack00_idx     =       60h             ; отместване на стека за DPL 0
307
308     0000046E                        label interrupt_descriptor_table fword          ; тук започва IDT
309
310     0000046E 0008 0000 0000  idt_start dw                   idt_size,0,0
311     00000474 051Er  0008 00  8E  +  interrupt_0     interrupt_descriptor    <demo_int,code32_idx,0,8eh,0>
312             0000
313
314             = CODE32:0008                   idt_size=$-(offset interrupt_0)
315
316                                     ;-----------------------------------------------------------------------
317
318     0000047C                        start32:        ; тук започва Защитения Режим
319     0000047C 66| B8 0010                    mov     ax,data32_idx           ; зарежда нужните регистри с подходящите
320     00000480 8E D0                          mov     ss,ax                   ; селектори
321     00000482 BC 00000400r                   mov     esp,offset stack32_end  ; размер на стека
322     00000487 8E D8                          mov     ds,ax
323     00000489 8E C0                          mov     es,ax
324     0000048B 8E E0                          mov     fs,ax
325     0000048D 8E E8                          mov     gs,ax
326
327     0000048F 66| B8 0050                    mov     ax,stack32_idx  ;
328     00000493 8E D0                          mov     ss,ax           ;
329
330     00000495 66| B8 0040                    mov     ax,oldtask_idx
331     00000499 0F 00  D8                      ltr     ax
332
333                                                                     ; зареждане на  регистри
334     0000049C E8 000000C0                    call    main            ; сега  всичко е наред за стартиране
335                                                                     ; на main
336
337     000004A1 EB 00                          jmp     next            ; изчистване на опашката
338     000004A3                        next:
339     000004A3 9A                             db      09ah            ; превключване  на новата задач
340     000004A4 0000r  0000 0048               dw      offset  newtask_TSS,0,newtask_idx
341
342                                     ;Връщане от превключената задача и връщане в Реален Режим
Turbo Assembler     Version 2.01        10/30/98 13:27:07       Page 7
pmswitch.ASM

343     000004AA EA                             db      0eah    ; far jump opcode ; когато main приключи, се Връщаме обратно
344     000004AB 0346r  0000 0020               dw      offset  exit16,0,code16_idx     ; към Реален Режим
345
346
347                                     ;-----------------------------------------------------------------------
348                                     ; protected mode translation of write_msg
349                                     ; In:   DS:ESI  - pointer to format string
350
351     000004B1                        proc    write_msg_pm
352     000004B1 66| 50 56 57 06        push    ax esi  edi es
353     000004B6 66| B8 0018                    mov     ax,core32_idx   ; за адресиране на екрана
354
355     000004BA 8E C0                          mov     es,ax
356     000004BC 66| 0F B6 7E 02                movzx   di,[esi+2]      ; Взема позицията на Y
357     000004C1 69 FF  000000A0                imul    edi,160
358     000004C7 66| 03 3E                      add     di,[esi]        ; Взема позицията на X
359     000004CA 66| 03 3E                      add     di,[esi]
360     000004CD 81 C7  000B8000                add     edi,0b8000h     ; физически адрес на текстовия  екран
361     000004D3 8A 66  04                      mov     ah,[esi+4]      ; Взема атрибутния байт
362     000004D6 83 C6  05                      add     esi,5
363     000004D9                        write_loop:
364     000004D9 8A 06                          mov     al,[esi]
365     000004DB 0A C0                          or      al,al           ; край на  низа?
366     000004DD 74 09                          jz      loop_end
367     000004DF 46                             inc     esi
368     000004E0 66| 26: 89 07                  mov     [es:edi],ax
369     000004E4 47                             inc     edi
370     000004E5 47                             inc     edi
371     000004E6 EB F1                          jmp     write_loop
372     000004E8                        loop_end:
373     000004E8 07 5F  5E 66| 58               pop     es edi  esi ax
```

```
000004ED  C3                                        ret
000004EE                                  endp      write_msg_pm
                                          ;-----------------------------------------------------------------
                                          ; проста обработка на прекъсване

000004EE  00 00   02 00 1F 49 27+         int_msg        db      0,0,2,0,1fh,'I'm the Interrupt Handler - returning '
          6D 20   74 68 65 20 49+
          6E 74   65 72 72 75 70+
          74 20   48 61 6E 64 6C+
          65 72   20 2D 20 72 65+
          74 75   72 6E 69 6E 67+
          20
00000519  6E 6F   77 21 00                        db      'now!',0

0000051E                                  proc      demo_int
0000051E  BE 000004EEr                              mov       esi,offset int_msg
00000523  E8 FFFFFF89                               call      write_msg_pm
00000528  CF                                        iretd
00000529                                  endp      demo_int

                                          ;-----------------------------------------------------------------
                                          ; главна процедура за Защитен Режим

00000529  00 00   01 00 1F 4E 6F+         pm_msg         db      0,0,1,0,1fh,'Now in Protected Mode - calling Interrupt '
          77 20   69 6E 20 50 72+
```



```
          6F 74   65 63 74 65 64+
          20 4D   6F 64 65 20 2D+
          20 63   61 6C 6C 69 6E+
          67 20   49 6E 74 65 72+
          72 75   70 74 20
00000558  48 61   6E 64 6C 65 72+                 db      'Handler!',0
          21 00

00000561                                  main:
00000561  BE 00000529r                              mov       esi,offset pm_msg        ; поставя съобщението...
00000566  E8 FFFFFF46                               call      write_msg_pm
0000056B  CD 00                                     int       0                        ; ...извиква просто прекъсване...
0000056D  C3                                        ret                                ; ...и се връща

                                          ;-----------------------------------------------------------------

0000056E                                  ends      code32

                                          ;*****************************************************************************
00000000                                  segment extracode32 para public use32        ;този сегмент съдържа целия 32-битов
                                          assume  cs:extracode32,   ds:extracode32     ; код и необходимите данни

00000000  0200*(??)                       estack32 db       200h dup (?)               ; 32-битов стек
00000200                                  label     estack32_end      dword

00000200  00 00   03 00 1F 4E 6F+         pm_msg1        db      0,0,3,0,1fh,'Now in New Code and Data Segment.
          77 20   69 6E 20 4E 65+
          77 20   43 6F 64 65 20+
          61 6E   64 20 44 61 74+
          61 20   53 65 67 6D 65+
          6E 74   2E 20
00000227  54 68   69 73 20 69 73+                 db      'This is the new task!',0
          20 74   68 65 20 6E 65+
          77 20   74 61 73 6B 21+
          00

0000023D                                  estart32:
0000023D                                  proc      newmain

0000023D  66| B8 0038                            mov       ax,edata32_idx           ; зарежда нужните регистри с подходящите
                                                                                     ; селектори
00000241  BC 00000200r                            mov       esp,offset estack32_end  ; размер на стека
00000246  8E D8                                   mov       ds,ax
00000248  8E C0                                   mov       es,ax
0000024A  8E E0                                   mov       fs,ax
0000024C  8E E8                                   mov       gs,ax
0000024E  66| B8 0058                            mov       ax,estack32_idx
00000252  8E D0                                   mov       ss,ax

00000254  BE 00000200r                            mov       esi,offset pm_msg1
```

```
00000259 66| 50 56 57 06          push    ax esi   edi es
0000025E 66| B8 0018                 mov     ax,core32_idx      ; адресиране на екрана
                                                                ;
00000262 8E C0                       mov     es,ax
00000264 66| 0F B6 7E 02             movzx   di,[esi+2]         ; Вземa позицията на Y
```



```
00000269 69 FF  000000A0             imul    edi,160
0000026F 66| 03 3E                   add     di,[esi]           ; Вземa позицията на X
00000272 66| 03 3E                   add     di,[esi]
00000275 81 C7  000B8000             add     edi,0b8000h        ; физически адрес на текстовия  екран
0000027B 8A 66  04                   mov     ah,[esi+4]         ; Вземa атрибутния байт
0000027E 83 C6  05                   add     esi,5
00000281                        write_loop1:
00000281 8A 06                       mov     al,[esi]
00000283 0A C0                       or      al,al              ; край  на стринга?
00000285 74 09                       jz      loop_end1
00000287 46                          inc     esi
00000288 66| 26: 89 07               mov     [es:edi],ax
0000028C 47                          inc     edi
0000028D 47                          inc     edi
0000028E EB F1                       jmp     write_loop1
00000290                        loop_end1:
00000290 07 5F  5E 66| 58            pop     es edi   esi ax

00000295 CF                          iretd              ; когато newmain приключи, се Връщаме обратно
                                                        ; към превключващата задача
00000296                        endp    newmain

00000296                        ends extracode32

                                ;*****************************************************************************
00000000                        segment dpl_0 para public use32          ; стеков сегмент за DPL_0
                                assume  cs:dpl_0, ds:dpl_0

00000000 0100*(????)            tos     dw      256 dup(?)
00000200                        label tos_end   dword

00000200                        ends dpl_0

                                ;*****************************************************************************
00000000                        segment TSS_code32 para public   use32   ; TSS сегмент на
                                assume  cs:TSS_code32, ds:TSS_code32     ; главната задача

00000000                        struc   oldtask_d
00000000 01*(0000)              back    dw      0                        ; указател към  TSS на предна задача
00000002 01*(0000)                      dw      0
00000004 01*(00000200r)         esp0    dd      offset tos_end           ; размер на стека за DPL 0
00000008 01*(0060)              ss0     dw      stack00_idx ;50h         ; индекс на дескриптора на стека в GDT
0000000A 01*(0000)                      dw      0
0000000C 01*(00000000)          esp1    dd      0                        ; размер на стека за DPL 1
00000010 01*(0000)              ss1     dw      0                        ; индекс на дескриптора на стека в GDT
00000012 01*(0000)                      dw      0
00000014 01*(00000000)          esp2    dd      0                        ; размер на стека за DPL 2
00000018 01*(0000)              ss2     dw      0                        ; индекс на дескриптора на стека в GDT
0000001A 01*(0000)                      dw      0
0000001C 01*(00000000)          tcr3    dd      0                        ; резервирано
00000020 01*(0000047Cr)         teip    dd      offset start32           ; указател на инструкциите
00000024 01*(0002)              teflags dw      00000000000010b          ; флагов регистър
00000026 01*(0000)                      dw      0
00000028 01*(0000)              teax    dw      0                        ; регистри на задачата
0000002A 01*(0000)                      dw      0
0000002C 01*(0000)              tecx    dw      0
0000002E 01*(0000)                      dw      0
```



```
00000030 01*(0000)              tedx    dw      0
00000032 01*(0000)                      dw      0
00000034 01*(00000000)          tebx    dd      0
00000038 01*(00000400r)         tesp    dd      offset stack32_end       ; размер на стека
0000003C 01*(00000000)          tebp    dd      0
00000040 01*(00000000)          tesi    dd      0
00000044 01*(00000000)          tedi    dd      0
00000048 01*(0010)              tes     dw      data32_idx               ; селектор на ES
0000004A 01*(0000)                      dw      0
0000004C 01*(0008)              tcs     dw      code32_idx               ; селектор на CS
0000004E 01*(0000)                      dw      0
```

```
525  00000050 01*(0050)                tss      dw      stack32_idx          ; селектор на SS
526  00000052 01*(0000)                         dw      0
527  00000054 01*(0010)                tds      dw      data32_idx           ; селектор на DS
528  00000056 01*(0000)                         dw      0
529  00000058 01*(0010)                tfs      dw      data32_idx           ; селектор на FS
530  0000005A 01*(0000)                         dw      0
531  0000005C 01*(0010)                tgs      dw      data32_idx           ; селектор на GS
532  0000005E 01*(0000)                         dw      0
533  00000060 01*(0000)                tldt     dw      0                    ; указател на LDT към задачата
534  00000062 01*(0000)                         dw      0
535  00000064 01*(0000)                         dw      0
536  00000066 01*(0068)                iomap    dw      104                  ; карта на I/O
537  00000068                          ends oldtask_d
538
539                                    align 16
540  00000000 0000 0000 00000200so+    oldtask_TSS      oldtask_d      <>   ; TSS на текущата задача
541           0060 0000 00000000  +
542           0000 0000 00000000  +
543           0000 0000 00000000  +
544           0000047Cso 0002 0000+
545           0000 0000 0000 0000 +
546           0000 0000 00000000  +
547           00000400so 00000000 +
548           00000000 00000000  +
549           0010 0000 0008 0000 +
550           0050 0000 0010 0000 +
551           0010 0000 0010 0000 +
552           0000 0000 0000 0068
553
554  00000068                          ends TSS_code32
555
556                                    ;*****************************************************************************
557  00000000                          segment TSS_extracode32 para public use32    ; TSS сегмент
558                                    assume  cs:TSS_extracode32, ds:TSS_extracode32        ; на новата задача
559
560  00000000                          struc newtask_d
561  00000000 01*(0040)                back     dw      oldtask_idx          ; указател към   TSS на предна задача
562  00000002 01*(0000)                         dw      0
563  00000004 01*(00000200r)           esp0     dd      offset tos_end       ; размер на стека за DPL 0
564  00000008 01*(0060)                ss0      dw      stack00_idx          ; индекс на дескриптора на стека в GDT
565  0000000A 01*(0000)                         dw      0
566  0000000C 01*(00000000)            esp1     dd      0                    ; размер на стека за DPL 1
567  00000010 01*(0000)                ss1      dw      0                    ; индекс на дескриптора на стека в GDT
568  00000012 01*(0000)                         dw      0
569  00000014 01*(00000000)            esp2     dd      0                    ; размер на стека за DPL 2
570  00000018 01*(0000)                ss2      dw      0                    ; индекс на дескриптора на стека в GDT
```



```
571  0000001A 01*(0000)                         dw      0
572  0000001C 01*(00000000)            tcr3     dd      0                    ; резервирано
573  00000020 01*(0000023Dr)           teip     dd      offset estart32      ; указател на инструкциите
574  00000024 01*(0002)                teflags dw       000000000000010b     ; флагов регистър
575  00000026 01*(0000)                         dw      0
576  00000028 01*(0000)                teax     dw      0                    ; регистри на задачата
577  0000002A 01*(0000)                         dw      0
578  0000002C 01*(0000)                tecx     dw      0
579  0000002E 01*(0000)                         dw      0
580  00000030 01*(0000)                tedx     dw      0
581  00000032 01*(0000)                         dw      0
582  00000034 01*(00000000)            tebx     dd      0
583  00000038 01*(00000200r)           tesp     dd      offset estack32_end  ; размер на стека
584  0000003C 01*(00000000)            tebp     dd      0
585  00000040 01*(00000000)            tesi     dd      0
586  00000044 01*(00000000)            tedi     dd      0
587  00000048 01*(0038)                tes      dw      edata32_idx          ; селектор на ES
588  0000004A 01*(0000)                         dw      0
589  0000004C 01*(0030)                tcs      dw      ecode32_idx          ; селектор на CS
590  0000004E 01*(0000)                         dw      0
591  00000050 01*(0058)                tss      dw      estack32_idx         ; селектор на SS
592  00000052 01*(0000)                         dw      0
593  00000054 01*(0038)                tds      dw      edata32_idx          ; селектор на DS
594  00000056 01*(0000)                         dw      0
595  00000058 01*(0038)                tfs      dw      edata32_idx          ; селектор на FS
596  0000005A 01*(0000)                         dw      0
597  0000005C 01*(0038)                tgs      dw      edata32_idx          ; селектор на GS
598  0000005E 01*(0000)                         dw      0
599  00000060 01*(0000)                tldt     dw      0                    ; указател на LDT към задачата
600  00000062 01*(0000)                         dw      0
601  00000064 01*(0000)                         dw      0
```

```
602   00000066 01*(0068)                iomap   dw        104                    ; карта на I/O
603   00000068                          ends newtask_d
604
605                                     align 16
606   00000000 0040 0000 00000200so+    newtask_TSS       newtask_d        <>    ; TSS на новата задача
607            0060 0000 00000000  +
608            00[illegible] 0000 00000000  +
609            0000 0000 00000000  +
610            0000023Dso 0002 0000+
611            0000 0000 0000 0000 +
612            0000 0000 00000000  +
613            00000200so 00000000 +
614            00000000 00000000  +
615            0038 0000 0030 0000 +
616            0058 0000 0038 0000 +
617            0038 0000 0038 0000 +
618            0000 0000 0000 0068
619
620   00000068                          ends TSS_extracode32
621
622
623                                     end start16
```

# ПРИЛОЖЕНИЕ 3

## Списък на инструкциите на MC68000

| Мнемоника | Описание | Операция | Флагове | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| ABCD | Add Decimal with Extend | (Destination)10+(Source)10 -> Destination | * | U | * | u | * |
| ADD | Add Binary | (Destination) + (Source) -> Destination | * | * | * | * | * |
| ADDA | Add Address | (Destination) + (Source) -> Destination | - | - | - | - | - |
| ADDI | Add Immediate | (Destination) + Immediate Data -> Destination | * | * | * | * | * |
| ADDQ | Add Quick | (Destination) + Immediate Data -> Destination | * | * | * | * | * |
| ADDX | Add Extended | (Destination) + (Source) + X -> Destination | * | * | * | * | * |
| AND | AND Logical | (Destination) AND (Source) -> Destination | - | * | * | 0 | 0 |
| ANDI | AND Immediate | (Destination) AND Immediate Data -> Destination | - | * | * | 0 | 0 |
| 0ASL,ASR | Arithmetic Shift | (Destination) Shifted by <count> -> Destination | * | * | * | * | * |
| Bcc | Branch Conditionally | if cc then PC + d -> PC | - | - | - | - | - |
| BCHG | Test a Bit and Change | ~(<bit number>) OF Destination -> Z<br>~(<bit number>) OF Destination -> <bit number> OF Destination | - | - | * | - | - |
| BCLR | Test a Bit and Clear | ~(<bit number>) OF Destination -> Z<br>0 -> <bit number> -> OF Destination | - | - | * | - | - |
| BRA | Branch Always | PC + d -> PC | - | - | - | - | - |
| BSET | Test a Bit and Set | ~(<bit number>) OF Destination -> Z<br>1 -> <bit number> -> OF Destination | - | - | * | - | - |
| BSR | Branch to Subraoutine | PC -> SP@-; PC + d -> PC | - | - | - | - | - |
| BTST | Test a Bit | ~(<bit number>) OF Destination -> Z | - | - | * | - | - |
| CHK | Check Register against Bounds | if Dn <0 or Dn> (<ea>) then TRAP | - | * | U | U | U |
| CLR | Clear an Operand | 0 -> Destination | - | 0 | 1 | 0 | 0 |
| CMP | Compare | (Destination) - (Source) | - | * | * | * | * |
| CMPA | Compare Address | (Destination) - (Source) | - | * | * | * | * |
| CMPI | Compare Immediate | (Destination) - Immediate Data | - | * | * | * | * |
| CMPM | Compare Memory | (Destination) - (Source) | - | * | * | * | * |
| DBcc | Test Condition, Decrement and Branch | if ~cc then Dn-1 -> Dn; if Dn<>-1 then PC+d ->PC | - | - | - | - | - |
| DIVS | Signed Divide | (Destination) / (Source) -> Destination | - | * | * | * | 0 |
| DIVU | Unsigned Divide | (Destination) / (Source) -> Destination | - | * | * | * | 0 |
| EOR | Exclusive OR Logical | (Destination) XOR (Source) -> Destination | - | * | * | 0 | 0 |
| EORI | Exclusive OR Immediate | (Destination) XOR Immediate Data -> Destination | - | * | * | 0 | 0 |
| EXG | Exchange Register | Rx <-> Ry | - | - | - | - | - |
| EXT | Sign Extend | (Destination).Sign-extended -> Destination | - | * | * | 0 | 0 |
| JMP | Jump | Destination -> PC | - | - | - | - | - |
| JSR | Jump to Subroutine | PC -> SP@-; Destination -> PC | - | - | - | - | - |
| LEA | Load Effective Address | Destination -> PC | - | - | - | - | - |
| LINK | Link and Allocate | An -> SP@-; Sp -> An; SP + d -> SP | - | - | - | - | - |
| LSL, LSR | Logical Shift | (Destination) Shifted by <count> -> Destination | * | * | * | 0 | * |
| MOVE | Move Data from Source to Destination | (Source) -> (Destination) | - | * | * | 0 | 0 |

| Мнемоника | Описание | Операция | Флагове | | | | |
|---|---|---|---|---|---|---|---|
| | | | X | N | Z | V | C |
| MOVE to CCR | Move to Condition Code | (Source) -> CCR | * | * | * | * | * |
| MOVE to SR | Move to the status Register | (Source) -> SR | * | * | * | * | * |
| MOVE from SR | Move from the Status Register | SR -> Destination | - | - | - | - | - |
| MOVEA | Move Address | (Source) -> Destination | - | - | - | - | - |
| MOVEM | Move Multiple Registers | Registers -> Destination<br>(Source[]) -> Registers | - | - | - | - | - |
| MOVEP | Move Peripheral Data | (Source) -> Destination | - | - | - | - | - |
| MOVEQ | Move Quick | Immediate Data -> Destination | - | * | * | 0 | 0 |
| MULS | Signed Multiply | (Destination) * (Source) -> Destination | - | * | * | 0 | 0 |
| MULU | Unsigned Multiply | (Destination) * (Source) -> Destination | - | * | * | 0 | 0 |
| NBCD | Negate Decimal with Extend | 0 - (Destination)10 - X -> Destination | * | U | * | U | * |
| NEG | Negate | 0 - (Destination) -> Destination | * | * | * | * | * |
| NEGX | Negate with Extend | 0 - (Destination) - X -> Destination | * | * | * | * | * |
| NOP | No Operation | - | - | - | - | - | - |
| NOT | Logical Complement | ~ (Destination) -> Destination | - | * | * | 0 | 0 |
| OR | Inclusive OR Logical | (Destination) OR (Source) -> Destination | - | * | * | 0 | 0 |
| ORI | Inclusive OR Immediate | (Destination) OR Immediate Data -> Destination | - | * | * | 0 | 0 |
| PEA | Push Effective Address | Destination -> SP@- | - | - | - | - | - |
| RESET | Reset External Devises | - | - | - | - | - | - |
| ROL, ROR | Rotate (Without Extend) | (Destination) Rotated by <count> -> Destination | - | * | * | 0 | * |
| ROXL, ROXR | Rotate with Extend | (Destination) Rotated by <count> -> Destination | - | * | * | 0 | * |
| RTE | Return from Exception | SP@+ -> SP; SP@+ -> PC | * | * | * | * | * |
| RTR | Return and Restore Cond. Codes | SP@+ -> CC; SP@+ -> PC | * | * | * | * | * |
| RTS | Return from Subroutine | SP@+ -> PC | - | - | - | - | - |
| SBCD | Subtract Decimal with Extend | (Destination)10 - (Source)10 - X -> Destination | * | U | * | U | * |
| Scc | Set According to Condition | if cc then 1's -> Destination else 0's -> Destination | - | - | - | - | - |
| STOP | Load Status Register and Stop | Immediate Data -> SR; STOP | * | * | * | * | * |
| SUB | Subtract Binary | (Destination) - (Source) -> Destination | * | * | * | * | * |
| SUBA | Subtract Address | (Destination) - (Source) -> Destination | - | - | - | - | - |
| SUBI | Subtract Immediate | (Destination) - Immediate Data -> Destination | * | * | * | * | * |
| SUBQ | Subtract Quick | (Destination) - Immediate Data -> Destination | * | * | * | * | * |
| SUBX | Subtract with Extend | (Destination) - (Source) - X -> Destination | * | * | * | * | * |
| SWAP | Swap Register Halves | Register [31:16] -> Register [15:0] | - | * | * | 0 | 0 |
| TAS | Test and Set an Operand | (Destination) Tested -> CC; 1 -> [7] OF Destination | - | * | * | 0 | 0 |
| TRAP | Trap | PC -> SSP@-; SR -> SSP@-; (Vector) -> PC | - | - | - | - | - |
| TRAPV | Trap on Overflow | if V then TRAP | - | - | - | - | - |
| TST | Test an Operand | (Destination) Tested -> CC | - | * | * | 0 | 0 |
| UNLK | Inlink | An -> SP; SP@+ -> An | - | - | - | - | - |

[ ] = номер на бит, * променян, - непроменян, U недефиниран, 0 нулиран, 1 установен в 1

# Литература


1. В. Григорьев. Микропроцессор i486. Архитектура и программирование. М., Гранал, 1993.
2. В. Михальчук, А. Ровдо, С. Рыжков. Микропроцессоры 80x86 Pentium. Архитектура, функционирование, программирование, оптимизация кода. Минск, Битрикс, 1994.
3. Дж. Фрир. Построение вычислительных систем на базе перспективных микропроцессоров. М., Мир, 1990.
4. Дж. Уокерли. Архитектура и программирование микро ЕВМ. М., Мир, 1984.
5. К. Боянов, В. Кисимов, Л. Бончев и др., Сборник приложни схеми с микропроцесори, С., Техника, 1981.
6. К. Боянов, Н. Вецев, С. Мачев и др. Справочник по персонални компютри. С., Техика, 1988.
7. Р. Иванов, О. Асенов. Архитектура и системно програмиране за IBM XT/AT. Габрово, 1993.
8. С. Стойков, И. Аманасов. Intel's 386. Ръководство на програмиста. С., MiSoft, 1993.
9. Intel Architecture Software Developer's Manual, vol.1, 2, 3, http://developer.intel.com.
10. MC68000 User's Manual. http://www.motorola.com/microprocessors.
11. MC68000 Family Programmer's Reference Manual. http://www.motorola.com/microprocesors.
12. Pentium II Processor Hardware Developer's Manual, http://developer.intel.com.
13. Pentium Processor Family Developer's Manual, vol. 1, 2, 3, http://developer.intel.com
14. Peripheral Components, Intel Corp., 1993.
15. PowerPC Microprocessor Family: The Programming Enviromnents. http://www.motorola.com/PowerPC.
16. T. Shanley. PowerPC System Arhitecture. MindShare Inc., 1995.

# Използвани съкращения и обозначения

| | |
|---|---|
| АЛУ | аритметико-логическо устройство |
| ВУ | външно устройство |
| ЕА | ефективен адрес на операнд |
| ИС | интегрална схема |
| МП | микропроцесор |
| ПУ | периферно устройство |
| УПИ | устройство за предварително извличане на инструкции |
| УУ | управляващо устройство |
| AGI | (Address Generation Interlock) - Блокиране генерацията на адрес |
| APIC | (Addvanced Programmable Interrupt Controler) - Интелигентен контролер за прекъсвания |
| BTB | (Branch Target Buffer) - Устройство за предсказване на преходи |
| CPU | (Central Processed Unit) - Централен процесор |
| CISC | (Complex Instruction System Computer) - Компютър със сложна система инструкции |
| CPL | (Current Privilege Level) - Текущо ниво на привилегия |
| DMA | (Direct Memory Access) - Директен достъп до паметта |
| DPL | (Descriptor Privilege Level) - Дескрипторно ниво на привилегия |
| DRAM | (Dynamic RAM) - Динамична RAM памет |
| FPU | (Floating Point Unit) - Устройство за работа с плаваща запетая |
| IFU | (Instruction Fetch Unit) - Устройство за извличане на инструкции |
| GDT | (Global Descriptor Table) - Глобална дескрипторна таблица |
| IDT | (Interrupt Descriptor Table) - Дескрипторна таблица на прекъсванията |
| IOPL | (Input Output Privilege Level) - привилегировано ниво за вход-изход |
| IU | (Integer Unit) - Устройство за целочислени операции |
| LDT | (Local Descriptor Table) - Локална дескрипторна таблица |
| LRU | (Least Recently Used) - Последно използван - стратегия за замяна в кеш паметта |
| MESI | (Modified, Exclusive, Shared, Invalid) - Протокол между кеш памети |
| MMU | (Memory Management Unit) - Устройство за управление на паметта |
| MMX | (Matrix Manipulation Extension) - Разширение за матрични изчисления |
| MIRR | (Memory Type Range Registers) - Регистри за дефиниране на области паметта |
| PC | (Personal Computer) - Персонален компютър |
| PCI | (PeripheralComponent Interconnect) - интерфейс за връзка между периферни компоненти |
| PDE | (Page Directory Entry) - Елемент в каталога на страници |
| PTE | (Page Table Entry) - Елемент в таблицата на страници |
| RISC | (Reduced Instruction System Computer) - Компютър с редуцирана система инструкции |
| RPL | (Requestor's Privilege Level) - Привилегировано ниво на заявителя |
| SIMD | (Single Instruction Multiple Data) - единичен поток инструкции с множество потоци данни |
| SMM | (System Management Mode) - Режим на управление на системата |
| TLB | (Translation Lookaside Buffer) - Кеш буфер за транслиране на виртуални във физически адреси |
| TR | (Task Register) - Регистър на задачата |
| TSS | (Task State Segment) - Сегмент за състояние на задачата |
| VME | (Virtual Mode Extension) - Разширение на виртуалния режим |

# Съдържание


Предговор ..... 2
1. Архитектурни особености на фамилията микропроцесори Ix86 ..... 3
1. Развитие на архитектурата на Ix86 ..... 3
2. Микроархитектура на PentiumII ..... 4
2. Изследване на архитектурните особености на I486 ..... 8
1. Симулация на вътрешните компоненти на микропроцесора ..... 8
2. Интерфейс и основни възможности на симулатора ..... 10
3. Симулация на конвейера на процесора при изпълнение на програма ..... 13
3. Развитие на системата инструкции на архитектурата Intel ..... 15
1. Развитие на базовата система инструкции ..... 15
2. Разширение на програмния модел ..... 15
3. Типове данни ..... 15
4. Разширение на функционалните възможности на МП ..... 16
5. Инструкции MMX ..... 17
6. Примери за използване на инструкциите MMX ..... 20
4. Оптимизация на код ..... 23
1. Влияние на методите за адресация върху оптимизацията на програмата ..... 23
2. Регистрова зависимост ..... 23
3. Изравняване ..... 24
4. Влияние на префиксите ..... 24
5. Оптимизация на преходите ..... 24
6. Планиране на инструкциите ..... 26
5. Системна архитектура. Управление на паметта в защитен режим ..... 30
1. Режими на работа на МП ..... 30
2. Системни регистри ..... 30
3. Управление на паметта в защитен режим ..... 34
6. Превключване от режим с реална адресация в защитен режим ..... 41
1. Превключване от реален в защитен режим ..... 41
2. Превключване от защитен в реален режим ..... 41
3. Примерна програма ..... 42
4. Използвани особености на асемблера TASM ..... 45
7. Многозадачност. Защита по привилегии ..... 47
1. Защита по привилегии ..... 47
2. Структура на задача ..... 48
3. Превключване на задачи ..... 52
4. Взаимодействие между задачи ..... 52
5. Примерна програма за превключване на задачи ..... 53
8. Организация на шината на микропроцесора ..... 55
1. Шинен интерфейс ..... 55
2. Функционални групи сигнали при i486 и Pentium ..... 55
3. Цикли за запис и четене ..... 58
4. Организация на шината при I80386 ..... 61
5. Организация на шината при I80286 ..... 61
6. Особености на шината при фамилия P6 ..... 61
9. Обработка на прекъсвания и изключения. Контролер за прекъсване ..... 64
1. Видове прекъсвания и изключения ..... 64
2. Алгоритъм на процесора при обработка на прекъсвания и изключения ..... 65
3. Потвърждаване на външно прекъсване ..... 66
4. Контролер на прекъсванията ..... 67
5. Особености на обработката на прекъсванията при фамилията P6 ..... 68

10. Изграждане на мултипроцесорни системи ..... 70
1. Заключване на шината ..... 70
2. Сериализиращи средства ..... 71
3. Програмируем интелигентен контролер на прекъсвания (APIC) ..... 72
4. Консистентност на кеш паметите ..... 77
11. Програмируем паралелен интерфейс. Таймери ..... 79
1. Програмируем паралелен интерфейс I8255A ..... 79
2. Използване на I8255A ..... 81
3. Таймери ..... 82
4. Интегрирани таймери ..... 85
12. Последователен програмируем интерфейс ..... 87
1. Предназначение на последователния интерфейс ..... 87
2. Програмируем последователен интерфейс I8251A ..... 87
3. Последователен интерфейсен адаптер I8250 ..... 90
13. Пряк достъп до паметта ..... 94
1. Пряк достъп до паметта ..... 94
2. Контролер за пряк достъп до паметта ..... 94
3. Интегрирани DMA контролери ..... 98
14. Микропроцесорни системи на базата на MC68000 ..... 100
1. Архитектурни особености на MC68000 ..... 100
2. Организация на шината ..... 103
3. Прекъсвания и изключения ..... 105
4. Особености на асемблера ..... 106
5. Едноплатков микрокомпютър на базата на MC68000 ..... 108
15. Микропроцесорни системи на базата на PowerPC ..... 112
1. Особености на архитектурата PowerPC ..... 112
2. Програмен модел. Система инструкции ..... 115
3. Системна архитектура ..... 120
ПРИЛОЖЕНИЕ 1 ..... 121
ПРИЛОЖЕНИЕ 2 ..... 124
ПРИЛОЖЕНИЕ 3 ..... 133
Списък на инструкциите на MC68000 ..... 133
Литература ..... 135
Използвани съкращения и обозначения ..... 136