В настоящата тема следва да бъдат представени режимите на работа и организация на паметта за IA-32 и Intel-64 архитектура. При тези две архитектури (IA-32 и Intel-64) съществуват два режима на работа на процесора – Real Mode (наричан още Real-Address Mode) и Protected Mode (наричан още Protected Virtual Address Mode), като процесорите Pentium и Itanium CPU работят стандартно при Protected Mode, а предходните модели процесори работят в Real-Address Mode (Pentium CPU позволява да се симулира Real-Address Mode). Режимът на работа се определя от флага PE в управляващия регистър CR_0 – установяването на 1 превключва процесора в Protected Mode (защитен режим), а на 0 – Real Mode. При Protected Mode нововъведенията са с цел поддържане на многозадачност, стабилност на системата, защита на паметта, както и хардуерна поддръжка на виртуалната памет. В съответствие с изброените два режима на работа на архитектурите IA-32 и Intel-64, съществуват три режима на работа и организация на паметта: плосък модел (Flat Model), сегментиран модел (Segmented Model) и Real-Address Mode Model. При Real-Address Mode Model цялата памет се разделя на равни по размер сегменти. В CPU за всеки един от тези от тези сегменти има по един регистър, който сочи началото на съответния сегмент. Програмистът има достъп и може да променя съдържанието на кой да е от тези регистри. Промяната на съдържанието автоматично води до преместване на съответния сегмент в друга начална точка. Съществуват и други регистри с общо предназначение, които също са програмно достъпни. Те служат за задаване на отместването на адресната клетка от началото. За останалите три сегмента може да се използва кой да е от останалите регистри. Следователно адресът на клетката се задава винаги чрез двойка регистри – сегментен регистър и регистър-отместване. При така наречения плосък модел (Flat Model) на оперативната памет адресите се получават по следния начин: съдържанието на сегментния регистър се измества с 4 позиции наляво и се събира с регистъра на отместване, като това, което се получава, е абсолютният адрес на клетката. Сегментните регистри са винаги 16 бита. В Real Mode те са младшата част (първите 16 десни бита). По този начин получаваме 20-битов адрес, тоест максималната адресируема единица в Real-Address Mode е 220B или 1MB. Този 1MB е за управляващи таблици, код на потребителската програма. Управление на паметта и транслация от логически (ефективен) към физически адрес би могла да бъде имплементирана върху плосък модел. Чрез този модел може да се постигне функционалността на операционна система. Но главното предимство на този модел е, че цялото адресно пространство е линейно от адрес 0 до адрес MaxBytes-1. Този режим е предвиден с цел програмите, направени за Intel 286, 386 (16 бита), да могат да работят директно без тяхното прекомпилиране. Предимствата му са **1)** Прост интерфейс за програмисти **2)**Изчистен от детайли дизайн **3)**Предоставя най-голяма гъвкавост 4)Максимална скорост на изпълнение Недостатъкът му е, че не е подходящ за многозадачни операционни системи, освен ако не е подобрен със допълнителен хардуер и софтуер за организация на паметта. Това е най-честия случай при модерните CISC процесори – сложна технология на организация на паметта и сигурност върху прост плосък модел на паметта. С това завършва прегледът на режима на работа и организация на паметта Real mode. В защитния режим на работа (Protected Mode) оперативната памет се организира по по-сложен начин – чрез така наречения Segmented Model. Потребителската програма се разполага на части в оперативната памет и за всяка такава част се създава един ред от една таблица – Локална Дескрипторна Таблица (ЛДТ). Може да има максимум 8152 части (реда). Всеки ред от тази таблица се нарича „дескриптор". Операционната памет има една, т. нар. „Глобална Дескрипторна Таблица", всеки ред на която сочи началото на всеки ЛДТ на тази програма. Тоест, ако има 500 програми, ще има 500 ЛДТ и 500 реда в ГДТ. Всеки елемент от ДТ се нарича „дескриптор" (8 байта) и има следната структура: При ГДТ ограничителят е 24b (размерът на частта, която описва). При Pentium IV и Itanium 8b от 16-те допълващи спадат към ограничителя. Базата съдържа адреса на съответното начало. При Pentium IV и Itanium останалите 8b от 16-те допълващи са към базата и така тя става 32b. Следващите 8b са за флагове за защита (видът достъп до съответната част от паметта). На всяка една от ЛДТ съответства по един регистър (Local Descriptor Table Register) и Global Descriptor Table Register (винаги сочи началото на GDT). LDTR – променя съдържанието си. Адресът се задава с двойка регистри – segment register и отместване или сегментен регистър и регистър-база. Винаги е 16-битов, като десният бит (стойност 1 или 0) показва дали да се адресира спрямо ЛДТ или ГДТ. В два бита се кодира какъв режим се използва. Остават 13 бита – едно число, което е отместване в дескрипторната таблица. Адресът на клетката, до която се иска достъп (32-битово число), спрямо абстрактната виртуална памет дава линеен адрес (това не е истинската физическа част на паметта).

В настоящата тема следва да бъдат представени основните програмни регистри, указатели, както и флаговия регистър за IA-32 и Intel-64 архитектура. Архитектурата на IA-32 и Intel-64 процесорите съдържа 16 регистъра, които следва да се познават от програмиста. Фигурата показва, че тези регистри може да се разделят на няколко основни групи: Регистри с общо предназначение. Тези осем 32/64-битови регистри с общо предназначение се използват главно за да съхраняват операнди за аритметични и логически операции. Сегментни регистри. Тези специални регистри позволяват на дизайнерите на системния софтуер да използват различни модели на организация на паметта. Тези шест регистъра определят в определено време кои сегменти на паметта са адресируеми в дадения момент. Регистри за статус и инструкции (включително флагов регистът). Тези регистри се използват да записват и променят определени аспекти от състоянието на процесора. Регистрите за общо предназначение на IA-32 и Intel-64 архитектурата са EAX, EBX, ECX, EDX, EBP, ESP, ESI и EDI. Тези регистри се използват за съхранението на операндите на логическите и аритметични операции. Те също така могат да бъдат използвани за операнди на изчисленията на адрес (освен че ESP не може да бъде използван като индексен операнд). На фигурата е показано, че 16-битовите части от тези регистри за общо предназначение, имат собствени имена, като това се оказва особено полезно при употребата на 16 битови данни. Регистрите с дължина дума се наричат AX, BX, CX, DX, BP, SP, SI и DI. Фигурата също така демонстрира и факта, че всеки байт от 16-битовите регистри AX, BX, CX и DX има отделно име и може да бъде третиран самостоятелно. Това е полезно при употребата на 8-битова информация. Тези байтови части са наречени AH, BH, CH и DH (високи байтове) и AL, BL, CL и DL (ниски байтове). Всички регистри за общо предназначение са на разположение за изчисления при адресиране и за резултатите на повечето аритметични и логически изчисления. Сегментните регистри на Intel процесорите дават на дизайнерите на системен софтуер гъвкавостта да избират сред множество модели на организация на паметта(разгледахме го в предната тема). Една пълна програма обикновено се състои от множество различни модули, всеки от които съдържа инструкции и информация. Във всеки момент от изпълнението на програмата само малка част от тези модули е всъщност в употреба. IA-32 и Intel-64 архитектурата се възползва от това като осигурява механизми които поддържат директния достъп в инструкциите и информацията не текущия модул. Във всеки момент шест сегмента от паметта могат да бъдат достъпни за изпълняваща се IA-32 и Intel-64 програма. Сегментите CS, DS, SS, ES, FS и GS се използват за идентификацията на тези шест текущи сегмента. Всеки един от тези регистри специфицира определен вид сегменти("code," "data" или "stack"), показани на фигурата. Всеки регистър уникално определя един определен сегмент, от сегментите които образуват програмата. Сегментът, съдържащ в момента изпълняваща се последователност от инструкции, е известен като текущия code сегмент и е специфициран като CS регистър. Intel процесорите извличат всички инструкции от този кодов сегмент, използвайки като отместване (offset) съдържанието на Instruction Pointer. Подпрограмните извиквания и параметри обикновено изискват регион от паметта да бъде асоцииран като стек. Всички стекови операции използват SS регистъра, за да се намери стека. DS, ES, FS и GS регистрите позволяват спецификацията на четири сегмента на данни всеки адресиран от текущата програма. Флаговият регистър е от голямо значение за управлението и протичането на програмата. Той се състои от еднобитови флагове, които се установяват на 0 или 1 от процесора, в зависимост от резултата от различни операции. Следва изброяване на основните флагове в този регистър. AF – междинен пренос (междинен пренос между младшата и старшата тетрада). Смисъла от този флаг е някоя от следващите инструкции да провери този флаг. Използва се при двоично кодиране на десетични числа. SF – най-левия бит на резултата (знака). OF – за препълване ако и двата са нула или и двата са единица нямаме препълване. Ако се различават, то имаме такова. DF – обикновено при сравнение на низове (обръщане на посоката на прочитане на данните). Този флаг може да се постави от приложната програма (разрешен е за достъп от всяка програма). TF – когато в този флаг се запише единица, се изпълнява една инструкция и се прави трап (в режим на трасировка). IF – за да се случи прекъсване този флаг трябва да е единица. IOPL – ниво на привилегия на входно изходните устройства. От това се определя дали да се състои дадено входно-изходно прекъсване или не. AC – когато този флаг се установи в единица, процесорът се задължава да прави проверка за подравняването на границите на операндите. Ако е в нула, за процесора няма значение къде са границите. ID – Ако в този флаг е записана стойност единица, се позволява да се използва идентификатора на процесора. Status Flags – статус флаговете на EFLAGS регистъра позволяват резултатите от една инструкция да влияят на следващи инструкции. Аритметичните инструкции използват OF, SF, ZF, AF, PF и CF. SCAS (Scan String), CMPS (Compare String) и LOOP инструкциите използват ZF за да сигнализират, че са свършили работата си. Control Flag – контролният флаг DF от регистъра EFLAGS контролира инструкциите с низове. Регистърът instruction pointer (EIP) съдържа адреса на отместването свързан с началото на текущия кодов сегмент на следващата инструкция, която чака изпълнение. Указателят на инструкция не е директно видим за програмиста, а се контролира неявно от инструкции за трансфер, прекъсвания и изключения.

В настоящата тема следва да бъде разгледано адресирането на операндите за IA-32 и Intel-64 архитектура. Първоначално е разгледано адресирането на операндите за архитектурата IA-32. Формирането на 32 битовия адрес става по следния начин: При адресацията на IA-32 процесорен набор инструкции една инструкция може да оперира върху нула или повече операнди. Пример за инструкция без операнди е NOP (no operation). Един операнд може да бъде във всяка от следните позиции:1)В инструкцията (immediate operand) 2)В регистър (EAX, EBX, ECX, EDX, ESI, EDI, ESP или EBP при 32-битови операнди; AX, BX, CX, DX, SI, DI, SP или BP при 16-битови операнди; AH, AL, BH, BL, CH, CL, DH или DL при 8-битови операнди; сегментните регистри; или EFLAGS регистъра за флагови операции) Достъпът до операндите в инструкцията и в регистрите може да бъде осъществяван много по-често от операндите в паметта, понеже операндите в паметта трябва първо да бъдат извлечени от паметта. Непосредствени операнди (Immediate Operands) Някои инструкции използват информация от самата инструкция като един (някой път и 2) от операндите. Такъв операнд се нарича immediate operand. Операндът може да е 32, 16 или 8-бита. Например: `SHR PATTERN, 2` Един байт от инструкцията съдържа стойността 2, номера на битовете с които трябва да се направи shift променливата PATTERN. Двойна дума от инструкцията съдържа маската която се използва за да се тества променливата PATTERN. Регистрови операнди (Register Operands) Операндите могат да бъдат поставени в един от 32-битовите регистри за общо ползване (EAX, EBX, ECX, EDX, ESI, EDI, ESP или EBP), в един от 16-битовите регистри за общо ползване (AX, BX, CX, DX, SI, DI, SP или BP) или в един от 8-битовите регистри за общо ползване (AH, BH, CH, DH, AL, BL, CL или DL). x86 процесорите имат инструкции за референция към сегментните регистри (CS, DS, ES, SS, FS, GS). Тези инструкции се използват от програмите само ако дизайнерите на системите са избрали сегментен модел на паметта.

x86 процесорите имат инструкции за референция към флаговия регистър. Операнди в паметта (Memory Operands) Инструкциите, чиито операнди се намират в паметта трябва да специфицират(директно или индиректно) сегмента който съдържа операнда и отместването на операнда във сегмента. За по-голямо бързодействие и компактност сегментните селектори се съхраняват във високо скоростни сегментни регистри. Така инструкцията трябва само да специфицира само желания сегментен регистър и отместването за да се адресира операнд в паметта. Инструкциите които изискват достъп до паметта или от регистрите използват един от следните методи за специфициране на отместването на операнда в сегмента:1)Повечето такива инструкции съдържат байт който явно специфицира адресиращия метод за операнда. Този байт, наречен modR/M байт, следва кода на операцията и специфицира дали операнда е в регистър или в паметта. Ако операнда е в паметта адресът се изчислява от сегментния регистър и някои от следните стойности: базов регистър, индексен регистър, scaling factor, displacement. Когато се използва индексен регистър modR/M байтът също е последван от друг байт който идентифицира индексния регистър и scaling factor.2)Някои инструкции неявно използват специални методи за адресация:1)За някои кратки форми на MOV които неявно използват EAX регистърът, отместването на операнда се кодира като двойна дума в инструкцията. Не се използвата базов регистър, индексен регистър или scaling factor.2)Операциите с низове неявно адресират паметта посредством DS:ESI, (MOVS, CMPS, OUTS, LODS, SCAS) или чрез ES:EDI (MOVS, CMPS, INS, STOS).3)Операциите със стек неявно адресират операндите чрез SS:ESP регистри; e.g., PUSH, POP, PUSHA, PUSHAD, POPA, POPAD, PUSHF, PUSHFD, POPF, POPFD, CALL, RET, IRET, IRETD, exceptions и interrupts. **Избиране на сегмент (Segment Selection)** Инструкциите за манипулация на информацията не е необходимо да посочват явно кой сегментен регистър се използва. За тези инструкции които не го правят процесорът автоматично избира сегментен регистър по съответни правила.**Адресация на x87 процесорен набор инструкции. x87 е наименование на набор от инструкции за работа с числа с плаваща запетая в процесорната архитектура x86. Подобно на архитектурата x86, тези инструкции носят името си от поредицата математически копроцесори на Intel, които въведоха x87 инструкциите. Операндите на x87 инструкциите може да се намират или в x87 регистрите, или в паметта (не се използват непосредствени операнди – такива, които са кодирани в самата инструкция). Повечето x87 инструкции могат да приемат операнд от паметта, а някои от тях могат и да записват резултата в паметта. При достъп до операнд в паметта, могат да се използват стандартните x86 методи за адресиране. Повечето x87 инструкции могат да приемат операнд от x87 регистър и да записват резултата в него. При достъп до x87 регистрите се използва означението ST(i), където i е число от 0 до 7. ST(0) обозначава регистъра, който е на върха на регистровия стек, ST(1) е регистърът под него и т.н. Повечето x87 инструкции са с два операнда, като първият е едновременно и място, където се записва резултатът от операцията (подобно на x86 инструкциите).** Следва разглеждане на адресирането на операндите за архитектурата Intel-64. Това е формирането на 64 битовия адрес. x86-64 процесор се държи като IA-32 процесор когато е включен в реален или защитен режим. Това са режими които се поддържат когато процесора не е в long режим. Докато големината на регистрите е увеличена до 64-бита в сравнение с предишната x86 архитектура, адресирането на паметта все още не е увеличено до пълните 64 бита. Поради това, load/store unit(s), cache tags, MMUs и TLBs могат да бъдат по-прости без никаква загуба на използваема памет. **Изчисляване на ефективния адрес (отместването)**

Фундаментални типове данни Байт, дума, двойна дума, четворна дума и двойна четворна дума са фундаменталните типове данни. Байтът представлява 8 последователни бита, започващи от някой логически адрес. Битовете са номерирани от 0 до 7; 0 бит е най-малко значимият. Думата представлява два последователни байта, започващи от някой адресиран байт. Така една дума съдържа 16 бита. Битовете на една дума са номерирани от 0 до 15, като 0-ят бит е най-малко значимият. Байтът, съдържащ 0-ят бит се нарича „нисък байт”; байтът съдържащ 15-ия бит се нарича „висок байт”. Всеки байт в думата има собствен адрес и по-малкият от адресите е адресът на думата. Байтът на този „нисък адрес” съдържа 8-те най-малко значими бита на думата, докато байта на „високия адрес” съдържа 8-те най-значими бита. Двойната дума представлява две последователни думи започващи от който и да е байт адрес. Двойната дума съдържа 32 бита. Битовете са номерирани от 0 до 31; нулевия бит е най-малко значимия бит. Думата съдържаща нулевия бит се нарича „ниска дума”; думата съдържаща 31-ия бит се нарича „висока дума”. Всеки байт в двойната дума има свой адрес и най-малкият адрес от тези адреси е адресът на двойната дума. Байтът на този най-малък адрес съдържа 8-те най-малко значими бита от двойната дума, докато байта на най-високия адрес съдържа осемте най-значими бита. Аналогично може да се представи образуването на четворни думи и двойни четворни думи. Следва да се отбележи, че думите не трябва да бъдат подравнени на четно-номерирани адреси и двойните думи не е необходимо да бъдат подравнени на адреси които се делят на четири. Това добавя максимална гъвкавост при структурите данни. Независимо, че байтовете, думите, двойните думи, четворните думи и двойните четворни думи са фундаменталните типове от операнди, процесорът също поддържа и някои интерпретации на тези операнди: Цели числа (Integer) Знакова двоична числова стойност съдържаща се в 64-битова четворна дума, 32-битова двойна дума, 16-битова дума или 8-битов байт. Знаковия бит се намира в 7-ия бит в един байт, 15-ия бит в дума, 31-ия бит в двойна дума и 63-ия бит четворна дума. Този знаков бит има стойност нула за положителни и 1 за отрицателни числа. Нулата има положителен знак. Цели числа без знак (Unsigned integer) Беззнакова двоична числова стойност съдържаща се в 64-битова четворна дума, 32-битова двойна дума, 16-битова дума или 8-битов байт. Цели числа без знак (Floating point) Форматът с плаваща запетая се основава на представянето на числата в нормализирана форма като произведение от мантиса с основата повдигнато на степен (експонентна част). Нормализираното представяне означава в ляво от десетичната точка да има само едно ненулево число. При двоичния формат с плаваща запетая принципът на представянето е същият, но основата е 2 вместо 10. Числото g се представя чрез следните зависимости: $g = (-1)^s \times M \times 2^{E-127}$. Знаковият бит е $(-1)^s$ като S=0 е за положителни и S=1 за отрицателни числа. Променливата E е 8 двоично число между 0 и 255. Променливата E се намалява със 127, което означава, че експонентата варира от 2^{-127} до 2^{128} . Мантисата M се формира от 23 бита като двоична дроб. Прецизността при формата с плаваща запетая не е фиксирана. Тя е около десет милиона пъти по-малка (2^{-24} до 2^{-23}) от стойността на числото. Това е предимство на формата с плаваща запетая. Големите числа имат големи разлика между съседните стойности, докато при малките числа тази разлика намалява Диапазони на плаващата запетая. -Near Pointer е 32-битов логически адрес – отместването в сегмент. Тези указатели се използват в плоския и сегментния модел на организация на паметта. -Far Pointer: 48-битов логически адрес от два компонента: 16-битов сегментен селектор и 32-битово отместване. Далечните указатели се използват при сегментен модел на организация. -Bit field е последователност от битове, която започва от произволна позиция и байт и съдържа до 32 бита. BCD-формат Десетичните числа са специален вид на представяне на числова информация, в основата на който е заложен принципа за кодиране на всяка десетична цифра на числото като набор от 4 бита. При това всеки байт на числото съдържа една или две десетични цифри в така наречения двоично-десетичен код (BCD – Binary-Coded Decimal). CPU съхранява BCD-числата в два различни формата:• опакован формат – в този формат всеки байт съдържа две десетични цифри. Десетичната цифра представлява двоичната стойност в диапазона от 0 до 9 с размер 4 бита. При което кодът на най-старшата цифра на числото заема старшите 4 бита. Следователно диапазонът на представяне на десетичното опаковано число в един байт е от 00 до 99; • неопакован формат – в този формат всеки байт съдържа една десетична цифра в четирите си младши бита. Старшите 4 бита (зона) имат нулева стойност. Тоест, диапазонът на представяне на десетичното неопаковано число в 1 байт е от 0 до 9. SIMD типовете данни са MMX и XMM. MMX типът е разработен, за да ускори мултимедийните програми и програмите за комуникация като включват нови инструкции и типове данни които позволяват на програмите да достигат ново ниво на производителност. MMX типът е разработен като множество от integer инструкции които могат да бъдат приложени към нуждите на голямо разнообразие от мултимедия. Основните факти за този тип са: една инструкция, много информация (Single Instruction, Multiple Data (SIMD)); 57 нови инструкции; 8 64-битови MMX регистри, (от 0 до 7); следните 4 нови типа данни: три пакетирани (bytes, words и doublewords, респективно с по 8, 16 и 32 бита) и нов 64-битов. Всеки елемент в пакетираните типове данни е отделен integer. Четирите MMX типа данни са: пакетиран байт (Packed byte) – 8 байта пакетирани в 64 бита; пакетирана дума (Packed word) – 4 16-битови думи пакетирани в 64 бита; пакетирана двойна дума (Packed doubleword) – 2 32-битови двойни думи пакетирани в 64 бита; четворна дума (Quadword) – 64 бита. Като пример, информацията за пикселите обикновено се представя като 8-битови integer-и или байтове. С MMX технологията осем от тези пиксели са пакетирани заедно в 64 бита и преместени в MMX регистър; когато една MMX инструкция се изпълнява, тя взима всички тези 8 пиксела наведнъж от MMX регистъра, прилага операцията на всички тях паралелно и записва резултата отново в MMX регистър.

В настоящата тема следва да бъдат разгледани форматът на инструкциите в защитен режим на работа: префиксите, кодове на операциите, Mod/rm и SIB байтове, отместването и непосредствените операнди, както и кодирането на адресните режими. Дадена инструкция е разделена на няколко елемента: 1) Instruction prefixes – отразяват се на поведението на операциите които трябва инструкцията да извърши. 2) Opcode – може да бъде един или повече байта (до три цели байта). 3) ModR/M байт – не задължителен и понякога може да съдържа част от opcode (кодът на операцията). 4) SIB байт – не е задължителен и представлява комплексни индиректни паметни форми. 5) Displacement – отместването не е задължително и е стойност с вариаща големина от байтове (byte, word, long). 6) Immediate – не е задължителен компонент и се използва като числова стойност от вариращи размери от байтове (byte, word, long). Инструкционни префикси (Instruction Prefixes). Те не са задължителни и се използват когато на основното поведение на инструкцията не достига функционалност. Има четири типа префикси:1) Lock and Repeat. 2) Segment Override. 3) Operand Size. 4) Address Size. Всяка инструкция може да има до 4 префикса и всеки префикс не трябва да бъде използван два пъти, иначе може да се интерпретира от процесора като недефинирано поведение. Lock префиксът се използва, за да се заключи магистралата за писане. Използва се с множество техники за синхронизация на кода. Repeat префиксите са замислени да бъдат от същия тип като Lock префиксът, така че инструкцията да може да има само Lock или Repeat, не и двата префикса по едно и също време. Repeat префиксите се използват само със string инструкции които поддържат този префикс. Има два repeat префикса, REPZ (повтаряне докато Zero флагът не е вдигнат) и REPNZ (обратното). Segment Override се използва, за да се смени сегмента по подразбиране на инструкцията. Всеки регистър за общо ползване има свой сегмент по подразбиране. Например SS за ESP, DS за BX, и т.н. Operand Size Override е отговорен за смяната на размера на операцията. Това означава, че ако даден код се изпълнява в 32 битова среда и този префикс се използва, тази инструкция ще се изпълни като 16 битова инструкция. Address Size Override работи като префиксът Operand Size, но върху операнда при косвен модел на паметта. Така, ако се чете (в 16 бита) от [BP+DI], след префиксиране, резултатът е четене от [EBX] Например (при модел на декодиране по подразбиране – 16 битов) MOV AX, BX // размерът на операцията е 16 бита, понеже AX и BX са 16 битови регистри MOV EAX, EBX // 32 бита MOV EAX, [EBX] // EAX е 32 бита, така че се четат 32 бита от паметта. Но тук:
MOV [EBX], 5 // Вижда се проблемът – не се знае размерът на операцията. Затова трябва явно да се укаже в асемблерния код размерът на операцията. Opcode Кодът на операцията е статичен елемент от инструкцията който води до (дефинира) самата инструкция. Големината на кода на операцията варира от 1 до 4 байта. Той също може да включва и още 3 бита от REG полето от ModR/M байта. ModR/M Някои инструкции изискват ModR/M байтът, за да специфицира формата на операндите. И при някои други инструкции типовете на операндите стават ясни предварително от техния код на операцията и не се нуждаят от ModR/M байт, за да се специфицират. ModR/M байтът не е задължителен. ModR/M може да доведе до следващ SIB байт. Кодът на операцията на инструкцията има информация за типа на операндите, но тези типове биха могли да бъдат разширени по някакъв начин, например да има immediate операнд или по-сложен начин за образуване на ефективния адрес. Ролята на ModR/M е да дефинира дали са необходими SIB байт, immediate операнд или отместване. По този начин той допълнително дефинира регистрите, които се използват, в зависимост от типа на операндите. ModR/M се образува от три полета: MOD полето са 2-та най-значими битове. То дефинира дали се използва отместване (displacement) и ако се използва такова, колко е голямо. MOD 00 означава че се използва отместване. MOD 01 изисква отместване 8 бита. MOD 10 изисква отместване 16 и 32 бита, като размерът се определя от кодиращия модел. MOD 11 означава, че се използват само регистри за общо ползване. Така нареченото REG поле (3 бита) се използва, за да специфицира регистъра на единия от операндите, ако се използва такъв, разбира се. R/M полето означава регистър (Register) или памет (Memory). То има различно значение, което е избрано от MOD полето.

SIB означава Scale-Index-Base, това е един байт който съдържа тези полета. Той не е задължителен и ако се използва, трябва да бъде след ModR/M байтът. SIB прави ефективния адрес по-изразителен и по такъв начин се използват по-малко инструкции, за да се изчислят по-сложни адреси. Формата е:
и се представя като [INDEX*SCALE + BASE].Отместване (Displacement). Отместването не е задължителен компонент, а може да бъде използвано, само ако R/M полето позволи. Размерът на отместването варира между 1 байт, 2 байта или 4 байта, MOD и R/M полетата дефинират това. Това поле трябва да следва ModR/M байта или SIB байта, ако такъв съществува **Immediate. Инструкциите могат да зареждат стойности в регистри или да използват просто чисел, а за да извършват други операции. За да не зареждат тази информация тя може да бъде част от инструкцията.**

В настоящата тема следва да бъдат разгледани форматът на инструкциите в IA-32e режим на работа и използването на REX префикс. Дадена инструкция е разделена на няколко елемента: 1) Instruction prefixes – отразяват се на поведението на операциите които трябва инструкцията да извърши. 2) Opcode – може да бъде един или повече байта (до три цели байта). 3) ModR/M байт – не задължителен и понякога може да съдържа част от opcode (кодът на операцията). 4) SIB байт – не е задължителен и представлява комплексни индиректни паметни форми. 5) Displacement – отместването не е задължително и е стойност с вариаща големина от байтове (byte, word, long). 6) Immediate – не е задължителен компонент и се използва като числова стойност от вариращи размери от байтове (byte, word, long). Инструкционни префикси (Instruction Prefixes). Те не са задължителни и се използват когато на основното поведение на инструкцията не достига функционалност. Има четири типа префикси:1) Lock and Repeat. 2) Segment Override. 3) Operand Size. 4) Address Size. Всяка инструкция може да има до 4 префикса и всеки префикс не трябва да бъде използван два пъти, иначе може да се интерпретира от процесора като недефинирано поведение. Lock префиксът се използва, за да се заключи магистралата за писане. Използва се с множество техники за синхронизация на кода. Repeat префиксите са замислени да бъдат от същия тип като Lock префиксът, така че инструкцията да може да има само Lock или Repeat, не и двата префикса по едно и също време. Repeat префиксите се използват само със string инструкции които поддържат този префикс. Има два repeat префикса, REPZ (повтаряне докато Zero флагът не е вдигнат) и REPNZ (обратното). Segment Override се използва, за да се смени сегмента по подразбиране на инструкцията. Всеки регистър за общо ползване има свой сегмент по подразбиране. Например SS за ESP, DS за BX, и т.н. Operand Size Override е отговорен за смяната на размера на операцията. Това означава, че ако даден код се изпълнява в 32 битова среда и този префикс се използва, тази инструкция ще се изпълни като 16 битова инструкция. Address Size Override работи като префиксът Operand Size, но върху операнда при косвен модел на паметта. Така, ако се чете (в 16 бита) от [BP+DI], след префиксиране, резултатът е четене от [EBX] Например (при модел на декодиране по подразбиране – 16 битов):MOV AX, BX // размерът на операцията е 16 бита, понеже AX и BX са 16 битови регистри MOV EAX, EBX // 32 бита. MOV EAX, [EBX] // EAX е 32 бита, така че се четат 32 бита от паметта. Но тук MOV [EBX], 5 // Вижда се проблемът – не се знае размерът на операцията. Затова трябва явно да се укаже в асемблерния код размерът на операцията. Opcode Кодът на операцията е статичен елемент от инструкцията който води до (дефинира) самата инструкция. Големината на кода на операцията варира от 1 до 4 байта. Той също може да включва и още 3 бита от REG полето от ModR/M байта. ModR/M Някои инструкции изискват ModR/M байтът, за да специфицира формата на операндите. И при някои други инструкции типовете на операндите стават ясни предварително от техния код на операцията и не се нуждаят от ModR/M байт, за да се специфицират. ModR/M байтът не е задължителен, като ModR/M може да доведе до следващ SIB байт. Кодът на операцията на инструкцията има информация за типа на операндите, но тези типове биха могли да бъдат разширени по някакъв начин, например да има immediate операнд или по-сложен начин за образуване на ефективния адрес. Ролята на ModR/M е да дефинира дали са необходими SIB байт, immediate операнд или отместване. По този начин той допълнително дефинира регистрите, които се използват, в зависимост от типа на операндите. ModR/M се образува от три полета: SIB означава Scale-Index-Base, това е един байт който съдържа тези полета. Той не е задължителен и ако се използва, трябва да бъде след ModR/M байтът. SIB прави ефективния адрес по-изразителен и по такъв начин се използват по-малко инструкции, за да се изчислят по-сложни адреси. Форматът се представя като [INDEX*SCALE + BASE]. Отместване (Displacement). Отместването не е задължителен компонент, а може да бъде използвано, само ако R/M полето позволи. Размерът на отместването варира между 1 байт, 2 байта или 4 байта, MOD и R/M полетата дефинират това. Това поле трябва да следва ModR/M байта или SIB байта, ако такъв съществува. Immediate. Инструкциите могат да зареждат стойности в регистри или да използват просто числа, а за да извършват други операции. За да не зареждат тази информация тя може да бъде част от инструкцията. При IA-32e архитектурата съществува така нареченият REX префикс Той позволява следното: 1) Използване на допълнителни регистри с общо предназначение и XMM-регистри; 2) Използване на 64 битов размер на операнда.

В настоящата тема следва да бъдат разгледани системните регистри в защитен режим и техният състав, предназначение и формати. Флаговият регистър е от голямо значение за управлението и протичането на програмата. Той се състои от еднобитови флагове, които се установяват на 0 или 1 от процесора, в зависимост от резултата от различни операции. Следва изброяване на основните системни флагове в този регистър. ID – ако в този флаг е записана стойност единица, се позволява да се използва идентификатора на процесора. VIP и VIF – тези флагове позволяват на всяко софтуерно приложение в многозадачна среда за работа да има виртуална версия на системния флаг IF. AC – когато този флаг се установи в единица, процесорът се задължава да прави проверка за подравняването на границите на операндите. Ако е в нула за процесора няма значение къде са границите. VM – този флаг служи за указване на виртуален режим 8086. RF – този флаг временно прекратява възможността за генериране на debug-изключения. NT – процесорът използва този флаг, за да управлява свързването на прекъснатите и извиканите задачи. IOPL – ниво на привилегия на входно изходните устройства. От това се определя дали да се състои дадено входно-изходно прекъсване или не. IF – за да се случи прекъсване, този флаг трябва да е единица. TF – когато в този флаг се запише единица, се изпълнява една инструкция и се прави трап (в режим на трасировка). Регистрите за управление на паметта (Memory Management Registers) са друг вид системни регистри, които, както подсказва името им, се използват за управление на паметта. Регистрите GDTR, LDTR, IDTR и TR (Task Register) са 48-битови и съдържат адресите на дескрипторните таблици. Следва по-подробно описание на всеки от изброените регистри. GDTR (Global Descriptor Table Register) – този регистър сочи към началния адрес на глобална дескрипторна таблица (Global Description Table), която се използва при сегментирания модел на адресация. LDTR (Local Descriptor Table Register) – този регистър сочи към началния адрес на локалната дескрипторна таблица (Local Description Table), която се използва при сегментирания модел на адресация. IDTR (Interrupt Descriptor Table Register) – този регистър сочи към началния адрес на таблицата, съдържаща входящите точки на програмните модули за обработка на прекъсвания (Interrupt Descriptor Table). TR (Task Register) – този регистър сочи към информацията, необходима на процесора за дефиниране на текущата задача. Управляващите регистри CR0, CR1, CR2, CR3, CR4 са 32-битови регистри и се използват за задаване на различни режими на работа на микропроцесора, базови адреси на таблиците с каталога на страниците при странична организация на паметта и някои разширени функционални възможности на микропроцесорите с архитектура IA-32 и Intel-64. Следва описание на всеки от петте управляващи регистри. Регистърът CR0 съдържа системни флагове, които контролират или указват условията, които се отнасят към системата като цяло, а не към отделна задача. Някои от тези флагове са следните: EM (Emulation, Bit 2) – този флаг указва дали се емулират или не копроцесорни функции. ET (Extension Type, Bit 4) – този флаг указва типа на копроцесора в съответната система. MP (Math Present, Bit 1) – този флаг управлява функционирането на инструкцията WAIT, която се използва за координиране на работата на копроцесора. PE (Protection Enable, Bit 0) – чрез този флаг се включва и, съответно изключва защитния режим. PG (Paging, Bit 31) – този флаг указва дали процесорът използва или не страница на таблиците за преобразуване на линее адрес във физически адрес. TS (Task Switched, Bit 3) – процесорът задава този флаг при всяко превключване на задача по време на изпълнение на компроцесорни инструкции. Регистърът CR2 се използва за обработка на странични прекъсвания, при условие че страницирането е задействано. В такъв случай CR2 съдържа адресът, от който се е получило страничното прекъсване. Регистърът CR3 също като CR2 се използва само, ако страницирането е задействано. В такъв случай, CR3 позволява на процесора да намери PTD (Page Table Directory) за текущата задача Регистърът CR4 се използва в защитен режим на работа за контролиране на редица операции, като поддръжка на режим на виртуален 8086, задействане на входно/изходни прекъсвания и други.

В архитектурите IA-32 и Intel-64 са вградени два механизма за организация на виртуалната памет – сегментация (segmentation) и странициране (paging). В така наречения защитен режим могат да се използват и двата механизма за адресация. При сегментацията адресът се получава от 16-битов сегментен регистър и 32-битово отместване. Сегментните регистри в архитектурите IA-32 и Intel-64 са следните: 1) CS – кодов сегментен регистър – показва началото на сегмента на кода на програмата; 2) SS – стеков сегментен регистър – показва началото на сегмента на стека; 3) DS, ES, FS, GS – сегментни регистри за данни – показват началото на четири сегментна за данни. Регистърът EIP е 32-битов и в него се записва отместването относно началото на кодовия сегмент, а регистърът ESP е 32-битов и в него се записва адресът на върха на стека, зададен като отместване относно началото на стековия сегмент. Всеки от останалите 32-битови общи регистри може да се използва за задаване на отместване при адресация на данни в някой от сегментите за данни. В защитен режим сегментните регистри се наричат селектори, поради специалната роля която изпълняват. Шестнайсете бита на всеки селектор се интерпретират по следния начин: 1) Индексът показва отместване в глобална (GDT) или локална (LDT) дескрипторна таблица; 2) Ако бита TI е 0, селекторът е за поле в глобалната дескрипторна таблица, ако е 1, селекторът е за поле в локалната дескрипторна таблица; 3) Битовете RPL (Requested Privilege Level) са битове за нивото на привилегии на текущата програма. Нивата на привилегии са 4 като най високото е 0. Това е нивото на привилегия на операционната система. Дескрипторните таблици съдържат сегментни дескриптори, като всеки сегментен дескриптор описва един сегмент. Сегментите дескриптори са по 8 байта, разпределени по следния начин (на долния ред са младшите 4 байта, на горния ред са старшите 4 байта): 1) Полето Base задава адреса на началото на сегмента. То е разделено на 3 части; 2) Полето Limit задава размерът на сегмента; то е 20 бита, така че максималният размер на сегмент е 1 000 000 единици. 3) Битът G (гранулярност) определя големината на единиците: ако битът G е 0, единиците в сегмента са байтове, така че един сегмент е до 1 MB; ако битът G е 1, единиците в сегмента са страници по 4KB – един сегмент може да достигне 4GB. 4) Битът DB задава размерът на операндите и адресите за този сегмент: ако DB = 0 — 16 битови операнди и адреси, ако DB = 1 — 32 битови операнди и адреси. 5) Битът P е бит за наличност (present): ако P = 1 — ако сегментът е наличен в оперативната памет; ако P = 0 — ако сегментът не е наличен в оперативната памет, т. е. ще трябва да му се търси място, ако се наложи да се пише по него; 6) Битовете DPL са битове за нивото на привилегии на дескриптора - от 0 до 3; 7) Битът DT определя типа на дескриптора: ако DT = 0 за системен сегмент; ако DT = 1 за сегмент за приложна програма. При адресиране се използват два типа дескрипторни таблици: глобална дескрипторна таблица GDT, която е единствена и се използва най-вече за дескриптори на системни сегменти. Адресът на началото на GDT се записва в регистър GDTR, който се инициализира от ядрото на операционната система при нейното стартиране. локална дескрипторна таблица LDT, която не е единствена и се използва най-вече за дескриптори на приложни сегменти. В даден момент може да се използва точно една локална дескрипторна таблица – нейният начален адрес е поместен в дескриптор на GDT, индексът на който се намира в регистър LDTR. Коя дескрипторна таблица ще се използва – глобалната или текущата локална, зависи от бита TI на съответния сегментен селектор – ако TI = 0 се използва GDT, ако TI=1 се използва LDT. Тъй като размерът на индекса в селекторите е 13 бита, то една дескрипторна таблица може да съдържа най-много 8192 дескриптора; Получаването на адреса при сегментацията става по следния начин (зададени са селектор и отместване): 1) По бита TI се определя в коя дескрипторна таблица да се търси сегментния дескриптор; 2) Ако TI = 0, таблицата е GDT – по регистъра GDTR и индекса в селектора се определя сегментният дескриптор в GDT; 3) Ако TI = 1, таблицата е LDT – по регистъра LDTR и регистъра GDTR се определя дескриптор в GDT, от който се извлича адресът на началото на LDT и след това по индекса в селектора се определя сегментният дескриптор в LDT; 3) След като е определен сегментният дескриптор, се прави проверка дали сегментът е достъпен за програмата, дали се намира в оперативната памет, дали отместването е в рамките на сегмента; ако това е изпълнено, взима се адресът Base на началото на сегмента и към него се прибавя отместването. За по-голяма ефективност на тази адресация, към всеки селектор се „залепва” 64-битов невидим регистър, който съдържа дескрипторът на сегмента, който последно е бил адресиран. По този начин, ако непосредствено след това отново се адресира този сегмент, информацията се изважда директно от невидимия регистър, а не от съответната дескрипторна таблица.Полученият след сегментацията адрес се нарича линеен адрес и той е част от линейното адресно пространство. Линейното адресно пространство е 4GB; оттук са възможни два случая: 1) Линейното адресно пространство директно се изобразява върху физическото адресно пространство. 2) Линейното адресно пространство е виртуално – извършва се странична преадресация; страниците имат размер 4KB. Кой от двата случая е налице, зависи от бит, разположен в контролния регистър CR0 – този бит указва дали се използва странициране или не.

В архитектурите IA-32 и Intel-64 са вградени два механизма за организация на виртуалната памет – сегментация (segmentation) и странициране (paging). В така наречения защитен режим могат да се използват и двата механизма за адресация. При 4KB страници за адресиране на 4 GB линейно адресно пространство са необходими 2^{20} страници. Архитектурите IA-32 и Intel-64 са пример за двуетапна адресна транслация. Ако в CR0 регистъра битът paging е установен, то следва, че освен сегментация се използва и странициране (segmentation and paging). Линейният адрес се дели на 3 части: CN (Catalog Number), PN (Page Number), Page Offset. В паметта имаме записан 4KB каталог (каталог на каталозите), в който са записани адресите на останалите 1024 каталога в паметта. Той се състои от 1024 реда съдържащи линейни адреси. Десетте бита в адреса, означени с CN определят с кой ред на каталога (entry) на каталозите е свързан търсеният адрес. Адресът на началото на каталогът на каталозите се записва в регистъра CR3. Описаните каталози в него се намират на произволно място в паметта и всеки от тях е по 4KB, т.е. описват се 2 на степен 20 страници. Всяка страница е по 4 KB, което прави точно 2 на степен 32 байта (колкото е цялото адресно пространство в 32-битова машина). В режим на сегментация и странициране получаваме 32-битов линеен адрес. Първите 10 бита от него (CN = Catalog Number – номерът на реда в каталога на каталозите) се умножават по ширината на полето на каталога (4 bytes) и се добавят към началния адрес на каталога (записан в CR3). Ако старшият бит на реда в каталога (entry) (P) е 1 (следователно present), значи че съществува каталог в паметта, който се сочи от реда в каталога (entry). В такъв случай се взимат младшите 20 бита от реда в каталога (entry), добавят им се 12 нули отдясно (началният адрес на страница е кратен на 4096) и се прибавя стойността на полето PN, умножена по широчината на реда в каталога (entry) (4 bytes). В реда в каталога (entry) (в младшите му20 бита) е записан адресът на търсената от нас страница, ако тя съществува (ако P = 1). Последните 20 бита на реда в каталога (entry) съдържат линейния адрес на страницата, а физическият адрес на нейното начало се получава, като отново към тези 20 бита прибавим 12 нули отдясно и се прибави 12-битовият offset. Това е реалният адрес в паметта. Този метод използва 4KB повече пространство от обикновеното странициране (4KB + 4 MB). При 4MB страници за адресиране на 4 GB линейно адресно пространство са достатъчни само 2^{10} или 1024 страници. При този размер описаната по-горе схема за странична преадресация практически остава без промяна. На следващата фигура е представена страничната преадресация при 4MB размер на страниците. Освен изброените два начина за странична преадресация, съществува възможност за смесване на представените две схеми – с размер на страниците 4KB и 4MB, съответно. Предимството на виртуалната адресация пред сегментацията във връзка с процесите (отделните големи програми) е, че смяна на процесите означава само смяна на редове в каталога (entry) в PT Root, т.е. ако се сменя адресното пространство всичко остава постоянно в паметта и всички програми работят във виртуални адресни пространства. Предимството на двуетапната транслация на IA-32 и Intel-64 архитектурите пред обикновеното странициране е, че смяната на всеки процес води до смяна на реда в каталога (entry) в главния каталог (който е само 4 KB). Работи се с активни динамични блокове по 4 KB (малките каталози). Недостатъкът е двойното адресиране. Това адресиране се прави от DLB – блок в кеш-паметта. Тези адреси не се смятат от аритметико-логическото устройство, а от специални малки суматори.

В архитектурите IA-32 и Intel-64 са вградени два механизма за организация на виртуалната памет – сегментация (segmentation) и странициране (paging). В така наречения защитен режим могат да се използват и двата механизма за адресация. Механизмът за странициране PAE (physical address extension).и поддръжка на 36-битово физическо адресиране е въведен в употреба в архитектурата IA-32. Тази употреба се указва чрез CPUID флага PAE (шестият бит в регистъра EDX, когато операндът-източник за CPUID-инструкцията е 2). Флагът PAE в управляващия регистър CR4 задейства механизмът за странициране PAE и разширява физическия адрес от 32 бита на 36 бита. По този начин процесорът предоставя 4 допълнителни бита за адресация. За да се използва тази възможност, трябва да се установят следните два флага: PG флаг (бит 31) в управляващия регистър CR0 – за задействане на страницираност. PAE флаг (бит 5) в управляващия регистър CR4 – за задействане на механизма за странициране PAE. При задействане на описания механизъм за странициране PAE, процесорът поддържа два размера на страниците – 4KB и 2MB. Както при 32-битовото адресиране, така и при това адресиране, е възможно да бъдат адресирани и двата размера таблици. За да се поддържат 36-битови физически адреси, трябва да се направят следните промени в структурите от данни, използвани при страницирането:1)Редовете в таблицата се увеличават на 64 бита, с цел да се пригодят към 36-битови физически адреси. Всеки 4KB каталог от страници и таблица от страници може да има до 512 реда.2)Към йерархията на структурите от данни за преобразуването на линейни адреси, се прибавя нова таблица. Тази таблица се нарича главен каталог и има 4 реда, всеки по 64 бита, като се намира над каталозите от страници в съществуващата йерархия.3)20-битовото поле на началния адрес на каталога от страници в управляващия регистър CR3 се замества с 27 битово поле.4)Преобразуването на линеен адрес се променя, за да се позволи преобразуването на 32-битови линейни адреси в по-голямо физическо пространство. На първата фигура е представена страничната преадресация за 36-битовите физически адреси при 4KB размер на страницата. Следва описание на преобразуването на линейни адреси при 4KB размер на страницата. При този метод за адресиране може да се адресират до 2^{20} страници, което прави 2^{32} байта или 4GB. За да се избират различните редове от таблицата, линейният адрес се разделя на четири секции:1)Ред в таблицата на главния каталог (битове от 30 и 31) – съдържа се отместването на всеки от четирите реда в таблицата на главния каталог. Избраният ред задава съответния физически адрес на каталога.2)Ред в каталога (битове от 21 до 29) – съдържа се отместването на всеки от редовете в таблицата на каталога. Избраният ред задава съответния физически адрес на каталога.3)Ред в таблицата на страниците (битове от 12 до 20) – съдържа се отместването на всеки от редовете в таблицата на страниците. Избраният ред задава съответния физически адрес на страницата във физическата памет.4)Отместване в страницата (битове от 12 до 20) – съдържа се отместването на физическия адрес в съответната страница. На втората фигура е представена страничната преадресация за 36-битовите физически адреси при 2MB размер на страницата. Следва описание на преобразуването на линейни адреси при 2MB размер на страницата. При този метод за адресиране са достатъчни 2^{11} (2048) страници за да се адресира 4GB физическо адресно пространство. За да се избират различните редове от таблицата в този случай, линейният адрес се разделя на три секции:1)Ред в таблицата на главния каталог (битове от 30 и 31) – съдържа се отместването на всеки от четирите реда в таблицата на главния каталог. Избраният ред задава съответния физически адрес на каталога.2)Ред в каталога (битове от 21 до 29) – съдържа се отместването на всеки от редовете в таблицата на каталога. Избраният ред задава съответния физически адрес на 2MB страница.3)Отместване в страницата (битове от 0 до 20) – съдържа се отместването на физическия адрес в съответната страница.

Задачата е единица работа, която процесорът може да разпредели, изпълни и прекрати. Задачата може да се състои в изпълнение на програма/процес, компонент на операционната система, прекъсване и т. н. Архитектурата IA-32 предоставя механизъм за запазване на състоянието на съответна задача, за нейното разпределяне и изпълнение, както и за превключване от една задача към друга. Всяка задача е изградена от два компонента – пространство за изпълнение на задачата и сегмент за състояния на задачата – Task State Segment (TSS). Пространството за изпълнение на задачата се състои от сегмент за кода, стеков сегмент и един или повече сегмента за данни. Task State Segment (TSS) задава сегментите, които съставят пространството за изпълнение на задачата и предоставят място за съхраняване на информацията за състоянието на задачата. В режим на многозадачност, Task State Segment (TSS) предоставя механизъм за свързване на задачите. Всяка задача се идентифицира със сегментния селектор на своя TSS. Когато дадена задача се зареди за изпълнение в процесора, сегментният селектор, началният (базовият) адрес, границата и сегментния дескриптор на TSS се зареждат в регистъра на задачата – Task Register (TS). Следва представяне на основните структури от данни, използвани при управлението на задачи, а именно TSS, дескриптор на TSS, TSS-gate и регистър на задачата (TR). TSS е системен сегмент, който се използва за съхраняване на необходимата на процесора информация за възстановяване на прекъснато изпълнение на дадена задача. В него се включват регистри за обща употреба (EAX, ECX, EDX, EBX, ESP, EBP, ESI и EDI), регистри за избора на сегмент (ES, CS, SS, DS, FS и GS), флагов регистър (ES, CS, SS, DS, FS и GS), указателен регистър (EIP), поле за връзка към предходната задача, поле за избор на сегментния селектор на LDT и на CR_3, поле за нивото на привилегия. Дескрипторът на TSS дефинира (задава) TSS. Този дескриптор може да бъде разположен единствено в GDT, а не в LDT или IDT. TSS-gate осигурява възможност за косвено, защитено обръщане (извикване) на задача. TSS-gate може да бъде разположен в GDT, LDT или IDT. На фигурата е представен неговият вид. Регистърът на задачата (TR) съдържа 16-битовия сегментен селектор и целия сегментен дескриптор на текущата задача (32-битов базов адрес, 16-битова граница на сегмента и атрибути на дескриптора). Тази информация се копира от TSS-дескриптора в GDT за текущата задача. Следва описание на концепцията за управление на задачите в архитектурата IA-32. операционната система има модул, наречен loader (зареждащ модул), който, получавайки съответна команда, извършва зареждане на задача за предстоящо изпълнение, като при това зареждане в оперативната памет за тази задача се създава споменатия вече (TSS). Освен този модул, при управлението на задачите участва и още един – планиращ модул (модул за планиране), който решава коя от задачите, заредени в оперативната памет, да активира. Той работи на следния принцип: 1) Издава команда към процесора за задействането му след даден квант време. 2) Обхожда таблицата на задачите, взима първата, която е в режим на готовност и я стартира, след което продължава по следния начин – записва състоянието на предишната и започва да работи по новата задача. 3) След като изтече квантът време, процесорът е бил задействан от таймер и този вид прекъсване води до активиране на планиращия модул, който отново издава команда за задействане на процесора и отново сканира таблицата. Има статистическа информация, която пренарежда таблицата, така че да не се позволи една и съща задача да се изпълнява многократно постоянно.

В настоящата тема е описано управлението на работата на задачите: изпълнение, превключване, влагане, рекурсия, изобразяване в линейното и физическо адресно пространство. Изпълнението на задача се назначава от софтуера или процесора по един от следните начини:1)Явно извикване на задача CALL инструкция2)Явно прескачане към задача чрез JMP инструкция3)Неявно извикване (от процесора) към задача, обработваща прекъсване4)Неявно извикване на задача, обработваща изключение5)Връщане към задача (чрез IRET инструкция), когато флагът NT в регистъра EFLAGS има стойност 1. Всички изброени методи за назначаване (dispatching) на изпълнение на задача дефинират съответната задача със сегментен селектор, който сочи към TSS-gate, или към TSS на тази задача. При назначаване на задача за изпълнение чрез CALL или JMP инструкция, селекторът на инструкцията може да избере или директно TSS, или TSS-gate, който съдържа селектора на TSS. При назначаване на задача, обработваща прекъсване или изключение, IDT на прекъсването или изключението трябва да съдържа TSS-gate, който от своя страна да съдържа селектора на TSS на задачата, обработваща прекъсване или изключение. Когато бъде назначено изпълнението на дадена задача, става автоматично превключване между текущо изпълняваната задача и новоназначената задача за изпълнение.

Ако текущо изпълняваната задача (извикващата задача) е извикала за изпълнение новоназначената задача, сегментният селектор на TSS на извикващата задача се съхранява в TSS на извиканата задача, за да се предостави обратна връзка (“link back”) към извикващата задача. Превключването към дадена задача се осъществява в един от следните четири случая:1)Текущата програма, задача или процедура изпълнява JMP или CALL инструкция към TSS дескриптора в GDT на дадената задача.2)Текущата програма, задача или процедура изпълнява JMP или CALL инструкция към TSS-gate дескриптор в GDT или текущата LDT на дадената задача.
3)Векторът на прекъсване или изключение сочи към TSS-gate дескриптор в IDT.4)Текущата задача изпълни IRET инструкция, при условие че флагът NT в регистъра EFLAGS има стойност 1. При превключването към друга задача, състоянието на средата, в която се е изпълнявала текущата задача (състояние на задачата или още контекст на задачата) се запазва в TSS и изпълнението на задача се прекратява за някакъв период от време. След това информацията за предстоящата за изпълнение задача се зарежда в процесора и нейното изпълнение започва от инструкцията, сочена от вече зареденият регистър EIP на текущата задача. Влагането (nesting) е понятие, което се използва, за да се укаже дали изпълнението на дадена задача е вложено в изпълнението на друга задача, тоест, дали е налице влагане на задачи. За тази цел се използва флагът NT в регистъра EFLAGS, като при наличие на такова влагане, в полето в TSS на текущата задача, което съдържа връзка към предходната задача, се записва TSS селекторът на задачата, в която е вложена текущата. При това, възможно е да има няколко нива на влагания, като при такова положение, в гореописаното поле на TSS селектора се записва TSS селектора на непосредствено по-високото ниво на влагане, а не на някое от по-високите. Рекурсията (рекурсивно, повторно извикване на задача) е нежелателна ситуация. Тази ситуация се получава, тъй като TSS позволява съхраняване само на един контекст за всяка задача и поради това, при рекурсивно (повторно) извикване на дадена задача, би довело до загубване на съществуващия контекст на съответната задача. За да не се стига до такава ситуация, в сегментния дескриптор на TSS съществува флага, чрез който се указва дали дадената задача е в режим „busy”, тоест дали се намира във все още незавършен процес на изпълнение. Изобразяването на задачите в линейното и физическо адресно пространство става по един от следните два начина:1)Използване на споделено адресно пространство между всички задачи. Когато механизмът за страниране не е задействан, това е единствената възможност. При това положение, всички линейни адреси се изобразяват в едни и същи физически адреси. Когато механизмът на страниране е задействан, тази форма на изобразяване се получава чрез използването на един каталог на страниците за всички задачи. Линейното адресно пространство може да надвишава достъпното физическо пространство, при условие че се поддържа използването на виртуална памет.2)Всяка задача има собствено линейно пространство, което се изобразява върху физическото адресно пространство. Тази форма на изобразяване се постига чрез използването на различен каталог на страниците за всяка от задачите. Тъй като управляващият регистър се зарежда при всяко превключване на задача, е възможно всяка да има различен каталог на страниците.

В настоящата тема е описана челната част (Front End) на вътрешната архитектура IA-32: устройство за предсказване на преходите, устройство за извличане на инструкциите и предварителна дешифрация, инструкционен буфер и декодери. Наред с това, са представени техники за повишаване на ефективността на тази секция, а именно макро- и микро-сливане, трасировки. Челната част (Front End) на вътрешната архитектура на IA-32 е един от основните компоненти на процесорите Intel, както може да се види на фигурата. Тя съдържа кеша за проследяване, който е кеш за инструкции. При липса в кеша за проследяване в непосредствена близост до кеша за проследяване е разположен ROM с микрокода, който съхранява микрооперациите на комплексните инструкции. За комплексните инструкции кешът за проследяване изпраща указател към ROM, по който се извлича съответната последователност от микрооперации, имплементиращи тази инструкция. В този компонент се включват следните устройства:1)устройство за предсказване на преходите 2)устройство за извличане на инструкциите и предварителна дешифрация 3)инструкционен буфер 4)устройство за декодиране на инструкциите 5)Следва описание на всеки от изброените компоненти. 1. Устройство за предсказване на преходите – това устройство позволява да се предвиди коя инструкция ще се изпълни при прехода още преди неговото действително изпълнение. За предсказване на прехода се използват различни стратегии за предсказване на преходите (branch prediction), които могат да бъдат статични и динамични. Основната разлика е, че при динамичните стратегии за предсказване на прехода се следи и се взима предвид поведението на програмата до стартирането на конкретната инструкция за условен преход. Най-често използваните статични стратегии за предсказване на прехода се основават на следното: 1)Приема се, че преходът никога не се осъществява и се извличат следващите инструкции; 2)Приема се, че преходът винаги се осъществява и се извличат инструкцията-цел на прехода и последователността от инструкциите след нея; 3)Решението зависи от операционния код на инструкцията за условен преход. Основната цел на динамичните стратегии за предсказване на прехода (dynamic branch prediction) е да се повиши точността на прогнозата, като се използва историята на изпълнението на програмата на до този момент. За съхраняването на историята на всяка инструкция за условен преход (или поне на последните изпълнени такива) се използват допълнителни битове, съдържащи стойностите на ключовете за осъществени/неосъществени преходи. Таблицата, съхраняваща историята на преходите, обикновено се съдържа в малък кеш, свързан с фазата на конвейера за предварително извличане на инструкции. Всяка позиция в таблицата съдържа три компоненти: адрес на инструкцията за условен преход, битове за историята на прехода, информация за целта на прехода (адрес на инструкцията-цел на прехода или самата инструкция-цел на прехода). Таблицата, съхраняваща историята на преходите, се нарича буфер за целта на прехода – BTB (Branch Target Buffer) и е представена на втората фигура. При използването на BTB е възможно да се достигне до 85-90% прогнозиране на преходите. Предимствата на този метод са, че не се изисква рекомпилация на кода или промяна в машинните инструкции. Недостатъците му са, че апаратната реализация е скъпа и ефективността силно се влияе от контекстните превключвания. 2. Устройство за извличане на инструкциите и предварителна дешифрация. Това устройство извлича инструкциите, които е вероятно да бъдат изпълнени, запазва в кеш памет често използваните инструкции и извършва тяхната предварителна дешифрация. 3. Инструкционен буфер. Този буфер извършва буфериране между устройството за предварителна дешифрация и устройството за декодиране на инструкциите. 4. Устройство за декодиране на инструкциите. Декодиращата логика на това устройство се състои в приемане на инструкциите от инструкционния буфер и в тяхното декодиране в микрооперации. Следва разглеждане на техники за повишаване на ефективността на тази секция, а именно макро- и микро-сливане, трасировки. При макро-сливането се осъществява сливане на стандартна последователност от две инструкции в една декодирана инструкция (микрооперация), така че двете инструкции изглеждат като една по-дълга инструкция. Така се увеличава производителността на декодирането, намалява се латентността и електрическата консумация. При микро-сливането се осъществява сливане на стандартна последователност от две микрооперации в една микрооперация. Така се увеличава производителността на завършващата част (Retirement). Трасировките са техника, при която стековият указател се изчислява чрез използване на специална логика, посредством която се постига повишаване на ефективността на изпълнение на входа и изхода от процедура/функция.

В настоящата тема е описана изпълняващата част (Execution Core) на вътрешната архитектура IA-32: разпределител, суперскаларни компоненти и диспечеризацията им, резервационна станция. Изпълняващата част (Execution Core) на вътрешната архитектура IA-32 е един от основните компоненти на процесорите Intel, както може да се види на първата фигура. Изпълняващата част използва стратегията с промяна на реда на изпълнението на инструкциите, заложен в програмата и включва следните компоненти:1)разпределител (устройството за разпределение на ресурсите)2)преименуващо устройство (Renamer)3)резервационна станция4)изпълнителни устройства Следва описание на всеки от изброените компоненти. 1. Разпределителят (устройството за разпределение на ресурсите) извлича микрооперациите от опашката и разпределя ключовите буфери, които обхващат буфера за пренареждане, физическите регистри за цели числа и за плаваща точка, буферите за четене и за запис. Някои от тях са разделени по такъв начин, че един логически процесор може да използва само половината от ресурса. Всеки логически процесор може да използва 63 позиции в буфера за пренареждане, 24 позиции в буфера за четене и 12 позиции в буфера за запис. Когато единият логически процесор използва максимално всички негови ресурси и се нуждае от повече такива (например повече позиции в буфера за запис), устройството за разпределение на ресурсите (разпределителят) временно блокира работата на този логически процесор и започва да обслужва другият логически процесор. Като се ограничава максимално допустимото използване на такива ресурси като ключовите буфери, се осигурява висок коефициент на „справедливост”, така да се каже, на обслужване на двата процесора и се предотвратява възникването на така нареченото мъртво блокиране. 2. Преименуващото устройство извършва преобразуване на имената регистрите в архитектурата IA-32 в имена на физическите регистри на процесора. Това дава възможност на универсалните целочислени регистри на архитектурата IA-32 да бъдат динамично разширени, за да се използват всички налични физически регистри. Устройството за преименуване на регистрите използва таблица на псевдонимите на регистрите, за да може да се проследява последната версия на всеки архитектурен регистър за указване къде се намират операндите за следващата инструкция. Всеки логически процесор разполага с индивидуална таблица на псевдонимите на регистрите, тъй като е необходимо да поддържа и следи своето архитектурно състояние. Изброените две устройства – разпределителят (устройството за разпределение на ресурсите) и преименуващото устройство – обслужват едновременно една и съща микрооперация. След като напуснат фазата на разпределение/преименуване, микрооперациите се разпределят в две опашки: опашка за достъп до паметта (четене-запис) и обща опашка от инструкции за останалите операции. Двете опашки са разделени по такъв начин, че всеки логически процесор може да ползва максимално половината опашка с оглед да се избегне монополизирането на ресурсите от един от логическите процесори. 3. Устройство за планиране на инструкциите (резервационна станция). Процесорът използва пет устройства за планиране на различните типове инструкции от изпълнителните устройства. Максимално могат да бъдат планирани шест микрооперации за изпълнение в рамките на един машинен цикъл. Решението за активиране на една инструкция за изпълнение се взема, когато нейните операнди са готови в регистрите и съответното изпълнително устройство е свободно. Всяко устройство за планиране разполага с опашка от 8 до 12 позиции, от която избира инструкция за изпращане към изпълнителните устройства. 4. Изпълнителни устройства. Изпълнението на инструкцията се осъществява в в някое от паралелните изпълнителни устройства. Поради преименуването на регистрите, операндите се четат от общия физически регистров файл и резултатът също се записва в него.

В настоящата тема е описана завършващата част (Retirement) на вътрешната архитектура IA-32: пренареждащ буфер, зареждане и запомняне в паметта. Завършващата част (Retirement) на вътрешната архитектура IA-32 е един от основните компоненти на процесорите Intel, както може да се види на първата фигура. Завършващата част актуализира архитектурното състояние на логическия процесор съобразно реда на инструкциите, заложен в програмата. Тя проследява кога при двата логически процесора има финализирани инструкции, които се оттеглят при съблюдаване на нареждането им в програмата. Двата логически процесора се обслужват алтернативно. След оттеглянето (Retirement) на инструкциите (микрооперациите – μops) за запис, данните се записват в кеш паметта от първо ниво. Следва да се отбележи, че за един цикъл може да се оттеглят (Retire) до три микроинструкции. Подсистемата памет включва бързия кеш за данни от първо ниво L1, унифицирания кеш от второ ниво L2 и унифицирания кеш от трето ниво L3 (само за някои процесори). Тази подсистема включва и буфера за преобразуване на логическите адреси във физически. Този буфер е обща структура за двата логически процесора, като всяка позиция в него е маркирана с таг на логическия процесор. Независимо от това кой от логическите процесори е заредил данните в кеш паметта и двата логически процесора могат да използват общо всички позиции в кеш паметта на всички нива. Така, например, единият логически процесор може да извлече инструкции и данни, необходими на другия логически процесор, което е често срещан случай в кода на сървърните приложения. Заявки за достъп до паметта на логическите процесори, които не могат да бъдат обслужени от йерархията на кеш паметта, се обслужват от шинната логика чрез обръщение към главната памет на компютърната система. Заявките на логическите процесори се обслужват на принципа „пръв влязъл – пръв обслужен”, като опашката и буферите се използват общо. При мултипроцесорни конфигурации с клъстери, идентификаторът на логическия процесор се изпраща в явен вид по външната шина по време на подаването на заявката. Другите транзакции по шината, като например транзакциите за предварително извличане на инструкции, наследяват идентификатора на логическия процесор, подал заявката, която е генерирала транзакцията. Тясно свързан със завършващата част (Retirement) е така наречения пренареждащ буфер, към когото микрооперациите се предават след тяхното изпълнение. Този буфер е разделен на две половини, използвани съответно от двата логически процесора. Буферът за пренареждане разделя изпълняващата част (Execution Core) от завършващата част (Retirement), като служи за свързващо звено между тези две части. Именно с негова помощ се обновява архитектурното състояние на логическия процесор съобразно реда на инструкциите, заложен в програмата и се управлява подредбата на изключенията. Освен това, завършващата част (Retirement) извършва проследяването на преходите и изпраща обновена информация за тях към таблицата, използвана за съхраняване на историята на преходите. Тази таблица е създадена с цел да се повиши точността на прогнозата при така нареченото предсказване на прехода (dynamic branch prediction), като се използва историята на изпълнението на програмата на до този момент. За съхраняването на историята на всяка инструкция за условен преход (или поне на последните изпълнени такива) се използват допълнителни битове, съдържащи стойностите на ключовете за осъществени/неосъществени преходи. Таблицата, съхраняваща историята на преходите, обикновено се съдържа в малък кеш, свързан с фазата на конвейера за предварително извличане на инструкции. Всяка позиция в таблицата съдържа три компоненти: адрес на инструкцията за условен преход, битове за историята на прехода, информация за целта на прехода (адрес на инструкцията-цел на прехода или самата инструкция-цел на прехода). Таблицата, съхраняваща историята на преходите, се нарича буфер за целта на прехода – BTB (Branch Target Buffer). При използването на BTB е възможно да се достигне до 85-90% прогнозиране на преходите.

В настоящата тема е описана организацията на кеш-паметта: зареждане и запомняне, логика за предварително извличане на данни и инструкции в L1 и L2 кеш, изпреварващо четене и запазване на консистентността. Кеш-паметта представлява памет с малък капацитет и бързодействие, сравнимо с това на процесора и по-високо от това на ОП. Първоначално кеш-паметта била съставена само от едно ниво, а в днешно време има три нива – L1 (най-малък обем и най-голяма скорост), L2 (среден по големина обем и скорост) и L3 (с най-голям обем и най-ниска скорост). Тези нива образуват йерархия на кеш-паметта. L1-кешът се намира в процесора и е разделен на две независими части – в едната се запазват инструкции, а в другата – данни. L2-кешът също се намира в процесора, за разлика от L3-кешът, който е извън него. Кеш-паметта изцяло се управляват от хардуер и се реализира с бърза статична памет SRAM, която няма нужда от опресняване. Кеш-паметта представлява структура, състояща се от множество от рамки (frames). Всяка рамка се състои от данни, етикет (tag) и състояние. Етикетът е характеристика на рамката, която е съществена при търсене в данните в кеша. Състоянието е два бита – бит valid, който показва дали данните в кеша са валидни и бит dirty, който показва дали в рамката са били писани данни. Търсенето и зареждането на рамки в кеша става по следния начин: проверява се дали входящият етикет съвпада с етикета на рамката. Ако не съвпадат, рамката не се намира (cache-miss) и в такъв случай блокът данни в рамката се изхвърля и се заменя с блок от по-долното ниво в йерархията, а след това блокът се подава на предното ниво в йерархията. По принцип блокът от данни в рамката съдържа няколко думи и от кеша се търси част от този блок. Основната логическа организация на кеша е групиране на няколко рамки в множество (set). Кешът се реализира като матрица, всеки ред на която представлява едно множество от рамки. Адресът за кеш-паметта се състои от три полета – етикет (tag), индекс (index) и отместване (offset). Индексът определя множеството, в което се намира търсената рамка. При търсене след като се определи това множество, данните и етикетите на всички рамки от него влизат в асоциативна схема (с търсене по съдържание), като на другия вход на тази схема влиза входящия етикет. В схемата се прави паралелно търсене на входящия етикет и ако той съвпадне с някой етикет на рамка от множеството, на изхода на схемата излизат данните в намерената рамка, заедно със съвпадението. Освен това, на изхода на схемата има отделен сигнал, който показва дали има cache-hit или cache-miss. Полето отместване (offset) показва мястото на търсеният байт в блока на търсената рамка. Всяко множество се състои от един и същ брой рамки. Ако този брой се означи с k, се казва, че кешът е k-асоциативен (k-way associative). Най-простият случай е при k=1 – кешът е директно изобразим (direct mapped). Тогава индексът показва определена рамка от кеша и не е необходим етикет, но се губи принципът за темпорална локалност. Някои съществени за кеш-паметта характеристики са: асоциативност (associativity) – използват се 2-6, 8, 16-асоциативни кешове; размер на блока (cache-block) – при малък няма пространствена локалност, при голям има излишно прехвърляне на байтове; обем на кеша – с голям кешът е по-бавен, а с малък няма темпорална локалност. При кеш-паметта възникват четири основни въпроса: 1)Разполагане на блоковете от данни в кеша. Разполагането на блоковете е инвариантно, тоест всеки блок попада в точно едно множество. При директно изобразими кешове блокът попада в точно една рамка, а при кешове с по-голяма асоциативност блокът попада в някои от рамките на едно от множествата на кеша. 2)Механизъм за търсене на блок от данни в кеша (зареждане от кеша). Индексът адресира множеството, в което се намира блока и след намирането му се извършва паралелно търсене на блока във всички валидни рамки на множеството. 3)Механизъм за изхвърляне на блокове от данни. При cache-miss, ако всички рамки в множеството са валидни, то трябва някоя от тях да се изхвърли, за да се освободи място за търсената, която ще се изтегли от по-долно ниво в йерархията. Има няколко стратегии за изхвърляне на рамки: стратегия LRU (least recently used) – при нея се изхвърля рамката, която е най-рядко използваната напоследък; стратегия random – при нея се изхвърля случайна рамка от множеството; стратегия NMRU (not most recently used) – определя се коя рамка от множеството е използвана най-често напоследък и се изхвърля случайна от останалите; теоретично оптимална стратегия – изхвърля се тази рамка, която ще се използва след най-много време (което, обаче е трудно постижимо). 4)Начин на записване на блок от данни в кеша (запомняне в кеша). При записване на данни в кеша стои въпросът дали да се записват едновременно и в оперативната памет. Според това има два вида кешове: write-through, при които данните се записват в оперативната памет се наричат кешове, като техен недостатък е намаляването на производителността в работата на кеша при много операции за запис; write-back, при които данните не се записват в оперативната памет, като при тях данните се записват в налична рамка и нейният бит dirty става 1. Това е съществено, тъй като при опит за изхвърляне на тази рамка по един от горните алгоритми, тя трябва да се запише в паметта преди да се изхвърли. Предимството на тези кешове е, че се намалява трафикът към ОП, ако имат голям обем, а недостатъкът е, че паметта губи консистентност – данни, които са записани в кеша в един момент може да не присъстват в оперативната памет.

В настоящата тема следва да бъдат разгледани многоядрените процесорни структури и принцип на НТ. Мултиядрените структури са представени в процесорите Intel Pentium D и Pentium Processor Extreme Edition. Тези процесори разширяват хардуерната поддръжка на многонишковата технология чрез предоставянето на две процесорни ядра в един физически процесорен корпус. Двуядрените процесори Intel Xeon и Intel Core Duo също предоставят две процесорни ядра в един физически корпус. Многоядрената топология на процесорите Intel Core 2 Duo е подобна на тази при процесорите Intel Core Duo. Процесорите Intel Pentium D предоставят два логически процесора в един физически корпус, като всеки логически процесор отделно изпълнително ядро и йерархия на кеш паметта. Процесорите Dual-core Intel Xeon и Intel Pentium Processor Extreme Edition предоставят четири логически процесора в един физически корпус, в който има две изпълнителни ядра. Всяко от тези ядра предоставя два логически процесора, които споделят тях и йерархията на кеш паметта. Процесорите Intel Core Duo предоставят два логически процесора в един физически корпус. Всеки от логическите процесори има отделно изпълнително ядро (включващо кеш памет от първо ниво) и „интелигентен" (smart) кеш от второ ниво. Тази кеш памет се споделя между двата логически процесора и е оптимизирана, така че да бъде намален трафикът в шината, когато едно и също копие на кеширани данни се използва от двата логически процесора. Пълният капацитет на кешът от второ ниво може да бъде използвано от един логически процесор, ако другият логически процесор не е активен. Най-общо казано, всяко ядро в една мултиядрена структура наподобява реализацията при едноядрен процесор по отношение на микроархитектурата. Реализацията на йерархията на кеш паметта в двуядрен или многоядрен процесор може да бъде същата или различна от нейната реализация при едноядрен процесор. При многонишковите процесори едновременно се обработват множество нишки чрез превключване на изпълнението им. При многонишковата обработка с времеделене процесорът превключва между софтуерните нишки на фиксирани интервали от време. При нея могат да възникнат празни слотове за изпълнение, но се минимизират отрицателните ефекти от голямото време за достъп до паметта. Многонишковата обработка с превключване при събитие се използва при сървърни приложения, при които две нишки изпълняват подобни задачи и възниква голям брой липси в кеша. Събитията, предизвикващи превключването на нишките, имат голяма латентност и най-често това са липсите в кеша. Недостатък е неефективното използване на ресурсите на процесора. При едновременната многонишкова обработка множество нишки се изпълняват от един процесор без превключване. При нея се постига най-ефективно използване на ресурсите на процесора и най-висока производителност. Съвременните приложения за сървъри съдържат множество „нишки" или процеси, които могат да бъдат изпълнени паралелно, т.е. те съдържат паралелизъм на ниво нишки (TPL – Thread Level Parallelism). Обработката на (on-line) транзакциите и Web услугите изобилстват от програмни нишки, които могат да бъдат изпълнявани едновременно, за да се повиши бързодействието. Дори съвременните приложения за настолни компютри се характеризират с повишена степен на паралелизъм. Едни от първите процесори, поддържащи хипернишковата технология, са процесорната фамилия Xeon и процесорът Pentium IV на фирмата Intel. Съвременните високопроизводителни (high-end) сървъри, както и тези със средна мощност (mid-range), имат мултипроцесорна архитектура. При тях се изпълняват паралелно множество нишки в множество процесори. Тези нишки могат да бъдат от една и съща приложна програма, от различни приложни програми, изпълнявани едновременно, от сервизните програми на операционната система или от програмите на операционната система, осъществяващи фонова -поддръжка. При архитектурите с хипернишкова технология един физически процесор изпълнява функциите на два логически процесора, всеки от които има собствено архитектурно състояние и споделя ресурсите на физическия процесор. Следователно процесите и нишките на приложните програми и програмите на операционната система могат да се изпълняват от двата логически процесора както в реална двупроцесорна система., за да се осигури това, инструкциите на двата логически процесора се изпълняват едновременно в общите ресурси на физическия процесор. Първата имплементация на хипернишковата технология е в микропроцесорната фамилия Xeon™ на Intel® за двупроцесорни и мултипроцесорни сървъри, при които един физически процесор обслужва два логически процесора. Увеличаването на размера на кристала в случая е не повече от 5%. Архитектурният статус на всеки логически процесор включва съдържанието на програмно достъпните регистри, управляващите регистри и регистрите на програмируемия контролер за прекъсване. Всеки логически процесор разполага със свой собствен контролер за прекъсване, като прекъсванията на даден логически процесор се обработват ексклузивно от собствения контролер за прекъсване. Логическите процесори ползват общо всички ресурси на физическия процесор като изпълнителни устройства, кешове, управляваща логика и шини.

В настоящата тема следва да бъдат разгледани прекъсванията и изключенията: вектори, таблица на прекъсванията и формат на елементите и, източници и приоритети на прекъсванията и изключенията, разрешения и забрани, обработка на прекъсвани и изключения. Нарушаването на нормалния цикъл на изпълнение на командата от процесора се нарича особена ситуация. Събитието, което поражда тази ситуация, се нарича аварийно събитие. Има два вида особени ситуации – прекъсване и изключение. Изключението възниква в тялото на текущата програма, където работи процесорът, и се обработва в нея. Обикновено това събитие е породено от невъзможността на процесора да извърши текущата команда. Например в резултат на прехода program counter (програмният брояч) да сочи някъде извън реалната памет или да се окаже, че командата е наред, но има адрес на една от данните, която сочи извън паметта. Други такива команди са: опит за делене на 0; умножение на две големи числа, при което битовете не достигат; опит на програмата да пише в област, в която няма право и др. Прекъсванията възникват винаги извън тялото на текущия процес, т.е. възникват от входно-изходните устройства, схемите на централния процесор или от други схеми. Тези събития се обработват от операционната система, извън тялото на текущия процес. И в двата случая обработката на аварийната ситуация се извършва първо от апаратурата, като централният процесор преминава в авариен цикъл на изпълнение, и след това от софтуерен модул, който обработва данните с цел да намали щетите от възникналото събитие. Централният процесор на всяка стъпка от нормалния ход на командата проверява преди реализацията на стъпката дали има условията за нейното изпълнение – ако стигне до последна стъпка от нормалния цикъл, проверява дали на неговите портове има сигнал за прекъсване. Линиите се обхождат от горе на долу, което определя приоритета на тяхната обработка. Аварийният цикъл се състои в следното: 1. Запазва се векторното състояние на текущата програма; 2. Определя се видът на аварийното събитие или чрез сигнал от линиите за прекъсване, или като код за линиите за данни към процесора; 3. По кода на вида прекъсване се извлича от оперативната памет нов вектор на състоянието, т.е. се прави зареждане с ново съдържание на централния процесор, включително и program counter-а. Старите регистри отиват в друга област – старо състояние. Новото и старото състояние са области в операционната памет; 4. Преход към нормален цикъл. В резултат на новото съдържание процесорът тръгва от нова команда. Всеки код на прекъсване има програмен модул, който обработва точно тази грешка. Всички тези модули образуват ядрото на операционната система. В зависимост от вида операционна система адресите на тези модули се задават по различен начин. Стандартно за Intel при процесорите има по един вектор на прекъсване, като броят им при IA-32 е 256. Векторите от 0 до 31 се използват за архитектурно дефинирани изключения и прекъсвания (някои от тях са резервирани и не се разрешава тяхното използване), а векторите от 32 до 255 са предназначени за прекъсвания, дефинирани от потребителя. Тези прекъсвания обикновено се използват от външните входно-изходни устройства. Всеки един от тези вектори на прекъсване е 4 байта. Когато процесорът работи в Real mode, тези 4 байта се тълкуват като 2 байта код на сегмент и 2 байта instruction pointer (отместването). Чрез тази двойка се задава адресът на всеки един модул. Тази таблица е на фиксирано място в паметта – първите 1024 байта на оперативната памет. Хардуерно (апаратно), при възникване на сигнал на прекъсване, освен че старото състояние се записва някъде, го занася директно в program counter-а, което прави тяхното директно изпълнение. В Protected mode за всяка програмна част, която се записва някъде, се създава един дескриптор, който представлява таблица на прекъсванията IDT (Interrupt Descriptor Table) и за нея има един регистър, който сочи началото на тази таблица. Когато дойде код на прекъсване, той се разглежда като код на отместване в таблицата, откъдето се определя началото на съответния програмен модул. Механизмът на прекъсване дава възможност да се избягва зациклянето при проверка дали някакво устройство е включено. Ако входно-изходното устройство е получило сигнал за започване на своята работа, тогава то следва следните стъпки: 0)Попълва някъде в регистър или в паметта статуса. Записва чрез флагове данни и праща сигнал за прекъсване към процесора на предпоследна стъпка; 1)Процесорът проверява дали има сигнал за прекъсване. Ако „да", в стандартния случай се приема прекъсването, запазва се съдържанието на program counter-а заедно с регистрите, като един от вариантите на съхранение е на фиксирани адреси, т.е. тази област на съхранение е разбита на предварително дефинирани участъци и в зависимост от кода на прекъсване апаратно се занася съдържанието на регистрите; 2)Съхранение на векторния процес; 3)Разпознава се кодът на прекъсване. Това става по два начина: или по сигналната линия, по която пристига; или по линията за данни, където се задава двоично число, което представя код на прекъсване, от този, който задава прекъсването (В/И). Нека кодът на прекъсване е числото i, тогава от таблицата на прекъсванията се извършва копиране на i-тия ред в регистъра РС на процесора. Тези последователни стъпки се изпълняват хардуерно и се формира така наречения авариен цикъл на процесора, след което той преминава в нормален цикъл 5)Извличане от РС първата команда за изпълнение.

В настоящата тема следва да бъде представен APIC: предназначение, структура на локален контролер и обработка на локалните и междупроцесорните прекъсвания. С цел да се замени двойката контролери на прекъсванията 8259, Intel създават усъвършенстван програмируем контролер на прекъсванията (Advanced Programmable Interrupt Controller – APIC), като първият процесор с вграден APIC е Pentium. Предназначението на APIC е да изпълнява две основни функции в процесора:1)Получава сигнали за прекъсване от процесора, от вътрешни източници и/или от входно/изходния APIC (или други външни контролери за прекъсвания) и ги изпраща към процесорното ядро за обработка. 2)В мултипроцесорна система, APIC изпраща и получава междупроцесорни съобщения за прекъсвания (IPI) към и от останалите IA-32 процесори, свързани към системната шина. Тези съобщения за прекъсвания могат да бъдат използвани за разпространяване на прекъсванията между процесорите в системата или да изпълняват други функции (например разпределяне на работния процес между няколко процесора). Входно/изходния APIC е част от системния чипсет на Intel. Основната му функция е да получава външни прекъсвания от системата и свързаните към нея входно/изходни устройства и да ги предават на локалния APIC като съобщения за прекъсване. Освен това, в мултипроцесорни системи, входно/изходният APIC предоставя механизъм за разпространяване на външните прекъсвания към локалните APIC на избрани процесори или групи от процесори, свързани към системната шина. След като локалният APIC изпрати прекъсване за обработка към процесорното ядро, с което е той (локалният APIC) е свързан, процесорът използва съответните механизми за обработка на изключения и прекъсвания, за да ги обработи. Следва преглед на структурата на локалния APIC. Всеки локален APIC се състои от APIC регистри и свързан с тях хардуер, който управлява предаването на изключенията към процесорното ядро и генерирането на междупроцесорни съобщения за прекъсвания. Стойностите на регистрите APIC може да бъдат четени и променяни чрез инструкцията MOV. Локалният APIC може да получава прекъсвания от следните източници: 1)Локално свързани входно/изходни прекъсвания. Тези прекъсвания произхождат от устройства, които са свързани директно с изводите за прекъсвания на процесора (pins LINT0 и LINT1). 2)Външни входно/изходни устройства. Тези прекъсвания произхождат от входно/изходни устройства, които са свързани с изводите за прекъсвания на входно/изходния APIC. Те се изпращат като входно/изходни съобщения за прекъсване от входно/изходната APIC към един или повече IA-32 процесора в системата. 3)Междупроцесорни прекъсвания. IA-32 процесорът може да използва механизма за междупроцесорни прекъсвания, за да прекъсне при необходимост дейността на друг процесор или група от процесори, свързани към системната шина. Този механизъм се използва за софтуерни прекъсвания, пренасочване на прекъсвания и други цели. 4)Прекъсвания, генерирани от APIC таймера. Локалният APIC таймер може да бъде програмиран да изпраща локални прекъсвания към свързания с него процесор, когато се достигна зададена стойност на програмиран брояч. 5)Прекъсвания, произхождащи от брояч за проследяване на производителността. APIC на процесорите Pentium 4, Intel Xeon и P6 предоставят възможност за подаване на прекъсвания към процесора, към който са свързани, когато се получи препълване на брояч за проследяване на производителността. 6)Прекъсвания от температурния сензор. Процесорите Pentium 4 и Intel Xeon предоставят възможност да изпращат прекъсвания към себе си, когато вътрешният им температурен сензор установи достигане на критична температура. 7)Прекъсвания, произхождащи от APIC вътрешни грешки. Когато бъде изпълнено някакво условие за грешка в локалния APIC (например опит за достъп до несъществуващ регистър), APIC може да се програмира да изпрати прекъсваме към процесора, към който той е свързан. Прекъсванията 1, 4, 5, 6 и 7 са локални прекъсвания и тяхната обработка се извършва по следния начин. Локалният APIC изпраща прекъсването към процесорното ядро чрез използване на специален протокол, който се задава чрез група APIC регистри, наричани с общото име локална векторна таблица. За всеки източник на прекъсване се отделя по един ред в нея, което позволява за всеки такъв да се използва отделен протокол. Прекъсванията 2 и 3 – от външни входно/изходни устройства и междупроцесорните прекъсвания се обработват чрез механизма за обработка на междупроцесорни прекъсвания (IPI). При междупроцесорните прекъсвания всеки от процесорите може да генерира междупроцесорно прекъсване чрез програмиране на управляващия регистър на прекъсванията (ICR) в локалния APIC. При операция запис в ICR се генерира IPI съобщение и се предава по системната шина (при процесорите Pentium 4 и Intel Xeon) или по APIC шината (при процесорите Pentium и P6). IPI може да се изпрати до останалите IA-32 процесори в системата или към процесорът, подал прекъсването. При получаване на IPI, локалният APIC автоматично обработва съобщението, използвайки информацията, включена в съобщението) и го предава към съответния процесор, за да бъде обслужено.

В настоящата тема следва да бъде представена системната и APIC магистрала: арбитраж, сигнализация на прекъсвания и протокол за обмен на съобщения, обработки. Следват общи сведения за системната магистрала и за APIC магистралата. При процесорите P6 и Pentium, входно/изходният APIC и локалният APIC осъществяват взаимодействие помежду си чрез APIC магистрала. Локалният APIC използва APIC магистралата за изпращане и получаване на междупроцесорни съобщения за прекъсване. APIC магистралата и съобщенията, предавани по нея, са невидими/прозрачни за софтуера и не се класифицират като архитектурни. При процесорите Pentium 4 и Intel Xeon, както и при следващите процесори, входно/изходната APIC (посредством архитектурата xAPIC) и локалните APIC осъществяват взаимодействие помежду си чрез системната магистрала. Входно/изходната APIC изпраща заявки за прекъсване към процесорите, свързани със системната шина посредством свързващ хардуер bridge hardware), който е част от чипсета (chipset) на Intel. Този хардуер генерира действителните съобщения за прекъсване, които се предават към локалните APIC. Междупроцесорните съобщения за прекъсване между локалните APIC се предават директно по системната шина. Следва преглед на арбитрацията при системната магистрала и APIC магистралата. Когато няколко локални APIC и входно/изходния APIC изпращат междупроцесорни съобщения за прекъсване и други съобщения за прекъсване по системната шина, се налага използване на арбитрация на шината, за да се определи редът на изпращане и управление на съобщенията за прекъсване. При процесорите Pentium 4 и Intel Xeon, локалните и входно/изходният APIC използват механизма на арбитрация, дефиниран за системната шина, за да се определи редът, в който се управляват междупроцесорните съобщения за прекъсвания. Този механизъм е неархитектурен и не може да бъде управляван от софтуера. При фамилията процесори P6, както и при процесорите Pentium, локалните APIC и входно/изходната APIC използват механизма за APIC арбитрация, за да бъде определен реда, според който се управляват междупроцесорните съобщения за прекъсвания. При тези процесори за всеки локален APIC се определя арбитрационен приоритет от 0 до 15, който входно/изходният APIC използва при арбитрацията, за да определи на кой локален APIC да бъде предоставен достъп до APIC магистралата. Локалният APIC с най-висок приоритет за арбитрация винаги получава достъп до магистралата преди всички останали локални APIC. След завършване на един арбитрационен цикъл, получилият достъп до APIC магистралата локален APIC намалява своя приоритет на 0, а всеки от останалите локални APIC увеличава своя приоритет с единица. Текущият приоритет за арбитрация на всеки локален APIC се съхранява в четири битов ID регистър (Arb ID), прозрачен за софтуера. По време на операция “reset”, този регистър се записва в APIC ID (който се съхранява в локалния APIC ID регистър). Прекъсването INIT може да се използва за ресинхронизиране на арбитрационните приоритети на локалните APIC чрез връщане на стойностите на Arb ID регистъра на всеки от тях към тяхната текуща APIC ID стойност. (Забележка: При Pentium 4 и Intel Xeon процесорите не съществува Arb ID регистър.) Следва описание на процеса на обработка на прекъсвания (включително протоколът за обмен на съобщения) след получаване на сигнализация за прекъсвания. При локални прекъсвания обработката се извършва по следния начин. След сигнализация за прекъсване, локалният APIC изпраща прекъсването към процесорното ядро чрез използване на специален протокол – протокол за обмен на съобщения – който се задава чрез група APIC регистри, наричани с общото име локална векторна таблица. За всеки източник на прекъсване се отделя по един ред в нея, което позволява за всеки такъв да се използва отделен протокол. Прекъсванията от външни входно/изходни устройства и междупроцесорните прекъсвания се обработват чрез механизма за обработка на междупроцесорни прекъсвания (IPI). При междупроцесорните прекъсвания всеки от процесорите може да генерира междупроцесорно прекъсване чрез програмиране на управляващия регистър на прекъсванията (ICR) в локалния APIC. При операция запис в ICR се генерира IPI съобщение и се предава по системната шина (при Pentium 4 и Intel Xeon) или по APIC шината (при Pentium и P6). IPI може да се изпрати до останалите IA-32 процесори в системата или към процесорът, подал прекъсването. При получаване на IPI, локалният APIC автоматично обработва съобщението, използвайки информацията, включена в него и го предава към съответния процесор, за да бъде обслужено. При Pentium 4 и Intel Xeon обработката на прекъсвания от локалния APIC се извършва по следния начин: 1. Определя се дали той е получателят на сигнала/съобщението и ако е, съобщението се приема. 2. Ако прекъсването е NMI, SMI, INIT, ExtINT или SIPI, прекъсването се изпраща към съответното процесорно ядро. 3. Ако е от друг вид, установява се съответен бит в IRR. 4. При няколко прекъсвания, те се подават едно по едно към процесора. 5. Когато едно прекъсване е било изпратено към процесора за обработка, нейното завършване се указва чрез инструкция в кода за обработка на прекъсванията, която записва съответна стойност в регистъра (EOI). При P6 и Pentium обработката на прекъсвания от локалния APIC се извършва по следния начин: 1. (Само при междупроцесорни прекъсвания.) Ако прекъсването е междупроцесорно, се определя дали той е получателят; ако е, съобщението се приема. 2. Ако прекъсването е NMI, SMI, INIT, ExtINT, INIT или на протокола MP (BIPI, FIPI или SIPI), прекъсването се изпраща към съответното процесорно ядро. 3. Ако е от друг вид, се търси свободен слот в една от двете опашки в IRR и ISR регистрите. Ако има свободен слот, поставя прекъсването в него, а ако не – отхвърля и изпраща обратно заявката за прекъсване. Стъпка 4 и 5 са същите като при Pentium 4 и Intel Xeon.