**ТЕМА 1.** **Компютърните архитектури и необходимостта от изчисления. Основни компоненти и организация на компютъра. Исторически бележки и тенденции в развитието.**

**Компютърните архитектури са дизайн от абстрактни слоеве: алгоритми, програмни езици,операционни системи, виртуални машини, ISA, микроархитектури, RTL, мрежи,устройства, физика**. Програмният език прави връзката между алгоритмите и компютъра. Компилаторът, който е част от програмните езици, превръща операторът от програмен език от високо ниво до куп инструкции, изпълними от съответния процесор.

* **RTL** представляват инструкции към процесора за изпълнението на различни задачи. Технологиите са определящ фактор за развитието на компютърния дизайн. Развитието на компютъра зависи от развитието на транзисторите, интеграционните мрежи, флаш паметите, core паметите, ROM и RAM паметите, VLSI.
* **VLSI** представляват големи чипове с по 1 милиард ключове на 1 пластинка.

Софтуерът също влияе на компютърните архитектури, но не толкова драстично. Въпреки това софтуерът и компилационните проблеми не могат да се пренебрегнат при развитието на компютъра.

Компютърните архитектури трябва да се съобразяват с много неща:

* производителност на цялостната система;
* цена за разработка на чиповете и поддържащите системи;
* цена за дизайн на чиповете;
* цена за разработка на приложения и софтуерни системи;
* нужната енергия за работа със системата;
* надежност на системата.

 Във времето различни неща са влияели върху компютърните архитектури и са влияели с различна тежест, защото приложението на компютъра се е изменяло с времето и продължава да се изменя до днес.

**История**

Първият компютър е измислен от Charles Babbage. Неговото изобретение е изцяло механично и е можело да прави различни изчисления. Самото устройство не е било приложимо, но то е дало началото и е положило основите на компютъра. Първият програмист е била Ada Byron, тя е била ученичка на Babbage и е програмирала на първия механичен компютър. Влиянието на това изобретение е било голямо. През 1944 година в Харвърд успяват да направят електронен аналог на компютъра на Babbage. Макар по- голямата част от Mark I да е механична в него има и електроника, която управлява компютъра. При него се налага да се премине към битова аритмерика и започва ползването на двоичната бройна система. Този компютър е тежал 5 тона и е бил изграден само от 750 000 части. За събиране са му били нужни 0,3 секунди, за умножение цели 6 секунди. Това го е правил неприложим напрактика, но това е било напредък в правенето на компютри.

За първи компютър се смята **ENIAC**, който е разработен 1943-1945 в университета на Пенсилвания. Той е изцяло електронен и се прави по поръчка на Пентагона. Ползвал се е за правенето на аналитични изчисления за военните. Тежал е 30 тона и е заемал 72 кв.м. той е 1000 пъти по- бърз от Mark I и вече е приложим, тъй като извършва операциите за по- прилично време. Дели за 6 милисекунди, събира за 200 микросекунди. Програмите за него са се пишели 2-3 часа и са били външни. Трябвало е за всяка нова програма да се намесва човек, който да наглася кабели така че да работи само тази програма. Този проблем се решава с втората версия на ENIAC, която се нарича **EDVAC**. През 1944 Джон Фон Нойман прави много важно откритие. Той намира начин да пази и маниполира програмата като информация(данни). Това е едно от най- важните открития за

компютърните архитектури. С този модел на Фон Нойман вече програмите не са външни. Те се пазят в операционната памет заедно с данните. (схема от лекция 1 CPU-MEM) Самият модел представлява затворена система състояща се от централен процесор и операционна памет, като в операционната памет се пазят програмите(поредици от инструкции, които процесорът може да изпълни) и данните. Процесорът изпълнява зададената програма и след това връща резултатите, като ги записва при данните в паметта.

Същност на идеите на Фон Нойман:

**Първи принцип**: Компютърът е електронно устройство и всички операции в него се извършват от електронни схеми, изградени на основата на електронни лампи, транзистори и интегрални схеми.

**Втори принцип**: Компютрите изградени с електронни елементи трябва да работят с двоична, а не с десетична бройна система.

**Трети принцип:** Компютърът е програмно управляемо устройство. Програмата, която управлява работата се съхранява в паметта на машината, заедно с междинните числови резултати.

**Четвърти принцип**: Компютърът трябва да изпълнява инструкциите последователно, една след друга, след прочитането им от паметта.

Универсалността на фон ноймановите компютри се явява в това, че за смяна на програмата паметта не е необходима промяна на схемите на свързване на компонентите на компютъра, а

може единствено да се смени програмата записана в паметта.

**Компютърът се разделя на две основни части – централен процесор и оперативна памет.** Всички компютри след EDVAC се наричат компютри с Нойманова архитектура. Проблемът на този модел е, че процесорът е от 10 до 50 пъти по бърз от паметта и това е водило до забавяне на процеса, тъй като се налага процесорът да чака паметта да свърши работата си. Този проблем го има и днес и се решава до някъде с използването на кеширана памет. След 1960 искат смяна на последователната Фон Нойманова архитектура.

Постепенно идеите се разпространяват и развиват и много университети започват да разработват свои собствени компютри. Първият комерсиялен компютър е **UNIVAC**. С развитието на компютрите започва и развитието на софтуера и програмните езици. Fortran е първият алгоритмичен програмен език. В началото централният процесор е било едно чисто изчислително устройство, но с времето се слага памет в него. Компютрите стават все по- мощни и се ползват за повече неща, но тези големи машини изискват много енергия. Друг проблем в миналото е, че хардуерът е бил много скъп, а също така и компютрите са имали малко памет.

**ТЕМА 2. Хардуерните елементи изграждащи компютърните системи. Основни логически схеми. Дешифратори, мултиплексори, суматори, цифрови компаратори.**

Основните хардуерни компоненти, които изграждат компютърните системи са **процесора, вътрешна памет, външна памет, дънна платка, видео, звукова и мрежова карта.**
Процесорът се подчинява на инструкции от програмите, обработва данните и притежава пълен контрол върху всички останали компоненти. Той е основната част на компютъра, която определя възможностите му. Бързодействието на процесора се определя от неговата тактова честота.
Вътрешната памет съхранява информацията, която е необходима на процесора. Видове външна памет са:

**ROM (Read Only Memory)**– постоянна, енергонезависима, съдържа служебна информация, предназначена само за четене.

**RAM (Random Accesses Memory)** – оперативна памет с произволен достъп; съдържа използваните в момента програми и файлове с данни. От нейния капацитет до голяма степен зависи бързо­действието на компютъра. Капацитетът на оперативната памет може да бъде 128, 256, 512MB. RAM паметите са на модулен принцип и може да се добавя още памет чрез допълнителните слотове на дънната плат­ка на компютъра.

Външната памет се реализира чрез различни устройства и служи за дълготрайно съхраняване на различната по обем информация, според капацитета на устройството. Тя може да бъде няколко вида, като например Hard Disk Drive – устройство с несменяем магнитен носител, голям капацитет и висока скорост при четене и писане. CD ROM - компактдисково устройство, за четене на информация, което работи на оптична основа, с капацитет 650 – 700 МВ. DVD (Digital Versatile Disk) - универсален цифров диск, с капацитет над 7 GB. Отличава се с високо качество на възпроизвеждане предимно на графична, звукова и мултимедийна информация.
**Дънна платка (**main board) е електрическа платка, към която се свързват всички устройства на компютъра. Тя се избира от вида на процесора и необ­ходимостта от бъдещо дооборудване на компютъра.
**Звуковата карта** е за възпроизвеждане на звук чрез компютъра. Звуковата карта е вътрешно устройство и се поставя в свободен слот на дънната платка, както и съществуват решения, при които звуковата картина е вградена в дънната платка.
**Видеокарта** или още графична карта, е компютърен елемент, който свързва дънната платка с монитора. Притежава собствена оперативна памет. От нейния обем зависят възможностите за настройка на честотата на опресняване, стабилността на картината, цветовете и др.
Мрежовата карта е вътрешно устройство, което се поставя в свободен слот на дънната платка. Позволява на компютрите да функционират в мрежа.

Логическите схеми са импулсни устройства, при които входните и изходните сигнали имат две възможни стойности - логически нива. Тези нива се означават с двоичните цифри 0 и 1, а поведението на схемите се описва със законите на двоичната логика. Когато на високото ниво е съпоставена логическа единица, а на ниското – логическа нула, логиката е положителна и обратно.
Към комбинационните логически схеми спадат дешифраторите, мултиплексорите, суматорите и цифровите компаратори.
Дешифратора е комб. логическа схема, която при определена комбинация на входните променливи активира определен изход. При n входа максималният възможен брой изходи е m=2n . Дешифратори, които притежават максималния брой изходи, се наричат пълни. Обикновено активното изходно ниво на интегралните дешифратори е логическа 0.
Мултипрексорът е комб. логическа схема, която притежава няколко информационни входа и един изход. Той се управлява от адресна логика, определяща кой от входните сигнали ще бъде пропуснат към изхода. При n адресни входа, максималният брой информационни входове е m=2n . Мултиплексори, които притежават максимален брой информационни входове, се наричат пълни.
Цифровият компаратор е комб. Логическа схема, която сравнява две числа А и В. В резултат на сравняването е възможно едно от трите състояния: А=В, А<В и А>В. Най-простият компаратор на равенство е схемата „изключващо или“. Компаратори, които могат да регистрират и трите състояния, се наричат магнитудни.
Суматотът е комб. логическа схема, които извършва сумиране на двоични числа. Ако суматорът не отчита входящия пренос, той се нарича полусуматор.

**ТЕМА 3. Памети. DRAM – принцип и особености. Interleaving. Начини за адресация на операндите. Граници на адресите.**

Операндите от паметта могат да бъдат адресирани по различни начини. Един начин за адресация
на операндите се **нарича непосредсвена адресация**. Полето от инструкцията съдържа стойността на операнда, който направо се подава на входа на АЛУ-то. Но при този начин за адресиране на операндите те могат само да се четат.
Друг начин за адресация **е регистровият**. При него физическият адрес на операнда се съхранява в регистър, чийто номер се подава като стойност на полето в инструкцията. При изпълняване на инстукцията стойността на операнда се взима от определения регистър и се подава на входа на АЛУ-то.
Следващ начин за адресация на операнди се нарича **регистров с отместване(**displacement). При
него в инструкцията имаме номер на така наречения базов регистър, от който подаваме
съдържанието на регистъра на суматор заедно с disp полето от инструкцията, което представлява
отмесване - обикновено цяло 16 или 32-битово число със знак. Полученото от суматора се записва в паметта и от паметта се подава операндът на АЛУ-то.
Друг начин на адресация е **косвено регистровият**. При него в кода на инструкцията имаме адреса на регистъра, в който се намира реалния адрес на операнда в паметта. Стойността на операнда се взима от оперативната памет от адреса, който се пази в регистъра, и се подава на входа на АЛУ-то.
Паметно абсолютен начин за **адресация на операнд**. При него в инструкцията се намира адресът на към операционната памет, на който се намира стойността на операнда и той се подава на АЛУ- то направо от паметта.
**Индексната форм**а е друг начин за адресация. При него в инструкцията се подават два регистъра, чиито съдържания се сумират. Идеята е единия регистър, да играе ролята на базов, а другия да съдържа относителното отместване спрямо базовия. Този начин на адресация наподобява начинът с отместване, но тук отместването се пази в регистър, а не се подава изрично от инструкцията. Индексната форма за адресация води до по-гъвкава програма, защото променяйки базовия адрес, можем да сменяме комплекта данни, които програмата използва, без да правим много промени по кода.
Друг начин за адресация на операнди е **косвено паметеният**. При него в кода на инструкцията се намира адрес на регистър, в който се съдържа адрес в ОП, в който адрес се съдържа накакво отместване, което ни праща на друго място в паметта. И на това място се съдържа операнда, който се подава на АЛУ-то.
**Scaled** е друг начин за адресация на операнди. При него в кода на инструкцията имаме адресите на два регистъра и непосредстев адрес. Съдържанието на регистрите и третият адрес от кода на инструкцията се събират в суматор и формират адреса в ОП на нашия операнд. След което операндът се подава на АЛУ-то от ОП.

**Memory** – основната памет, трябва да съхранява цялото оригинално линейно пространство. Кеш паметта съхранява само малка част дублирани адреси от адресното пространство. Добавена е и още една голяма памет в която се съдържа информацията от другите памети. По-късно се вижда, че този вариант повишава коефициента на производителност и тази идея се развива, като кеш паметта се реализира вече на 3 нива(L1, L2, L3).

RAM = Random Access Memory – основната памет, която се адресира. Основната памет е голяма и по размер се доближава до виртуалното адресно пространство. Cache паметта трябва да е много бърза, затова се прави от статична рам памет и за всеки бит са нужни по 6 биполярни транзистора.
За основната памет се използва **DRAM** (Dynamic RAM).DRAM паметта се представя като матрица (с равни редове и колони) от множество такива островчета. Адреса на едно такова островче се формира по следния начин: в лявата половина на адреса се намира редът където се намира то, а в дясната - стълба.

За **SRAM** е важна скоростта, нарича се статична защото не се изтрива при четене, няма нужда от рефреш, гълта повече ток и загрява повече чипа. Времето за достъп е само един такт, докато при DRAM времето за достъп е 2 такта. За да се увеличи скоростта на извличане на информация от паметта се създават така наречените банки памет – подреждат се няколко чипа памет (банки) и се използва interleaving – припокриване на работата на 4 чипа.

**ТЕМА 4.Микроархитектура. RISC и CISC.**

Микроархитектура:

Ако има много функционални блокове микрокомандата ще стане много по- дълга. И тук се сблъскваме с проблема за паметта, в която се съхраняват микроинструкциите, а именно ROM паметта. Тя трябва да бъде много бърза, което от своя страна значи, че тя трябва да е малка, тоест да не е обемиста, което я оскъпява допълнително. В една микропрограма имаме височина и дължина. Височината представлява броят на микрокомандите, които винаги са с една дължина. А дължината представлява колко бита е всяка микрокоманда.Нека си представям тази памет като матрица с наредени последователните инструкции, които са подадени от микропрограмата. Всеки ред представлява нова микрокоманда. А колоните са различните видове контролни сигнали, които може да подава **микроинструкцията** (микрокомандата), като за всяка микроинструкция имаме еденица само в колоните които трябва да се изпълнят. Има и една колона, която съдържа номера на следващата инструкция. За да се постигне желаната бърсина на ROM-a, той трябва да е възможно най- малък и това може да се постигне като намаляме височината или дължината му.
Технологията **CISC** (Complex Instruction Set Computer) е свързана с традиционните процесори, при които се поддържат множество инструкции, изпълнявани за различно време. Това време е в зависимост от типа на инструкцията, дължината, метода за адресиране на операндите и пр. В някои процесори изчисляването на резултата става на части поради разликата между дължината на реалните операнди и размера на входовете на АЛУ. Като правило в традиционният CISC- **процесор инструкцията се реализира като група от няколко байта в паметта, като управлението се извършва от микропрограмно устройство**. Последното извлича конкретна микропрограма, съответстваща на изпълняваната инструкция, след което реализира последователност от действия - елементарни операции, определяни от включените в микропрограмата микроинструкции. Тази последователност се нарича **“вътрешен цикъл”.** Така всяка инструкция се изпълнява за различен брой стъпки (тактове) в зависимост от броя на микроинструкциите в тази микропрограма.

Технологията **RISC** (Reduced Instruction Set Computer) е развита в началото на 80-те години като алтернатива на универсалните процесори с микропрограмна архитектура и разширен състав на системата от инструкции. **RISC-процесорите работят с ограничено множество** от инструкции(голям брой инстукции) и се характеризират със следното:

* Броят на инструкциите и различните видове адресиране на операндите са намалени (ограничени са до най-необходимите) и са въведени явни команди за зареждане и запис (Load/Store архитектура) на регистрите;
* Използват се прости инструкции с еднаква фиксирана дължина, равна на дължината на машинната дума и общ формат (определя еднакъв и минимален брой тактове за изпълнение на инструкциите);
* Операциите са от тип регистър-регистър с триадресна структура от типа (R1)+(R2) R3, които се изпълняват за един машинен такт (това налага голям брой регистри);
* Премахва се микропрограмното управление (използва се твърда логика за всяка инструкция, има твърд хардуерен контрол); 
* Комуникациите с паметта са на основата на инструкции за зареждане и съхраняване на данни;
* Избегнати са закъсненията при преходи;

 Възможност за изпълнение на 1,5 команди на такт чрез прилагане на конвейеризация на микро ниво и вградена кеш-памет (cache).

 Зависимост от оптимизацията на компилатора
При RISC-**процесорите се търси ефективно натоварване (запълване) на вътрешния конвейер, което води до по-висока производителност.** За целта управляващото програмно осигуряване трябва да поддържа ефективна конвейеризация на RISC-инструкциите, което се подпомага от опростеното предварително извличане и изпълнение.

**ТЕМА 5. Паралелизми на ниво инструкции. Конвейерно изпълнение на инструкциите. Хазарти в инструкционните конвейери – ресурсни конфликти, процедурни зависимости и зависимости по данни. Производителност на инструкционния конвейер и методи за ускоряване. Инструкции за отложен преход.**

При създаването на процесорите целта е била просто да има компютър, който да смята и да е

удобен за работа. Но постепенно компютрите се развиват и започват да се използват за все повече неща. Развиват нещата свързани с ОС, входно изходните системи, периферните устройства.
Правенето на по-производителни машини тръгва в две посоки:

1) да се направи нова архитектура. Правят се предложения, които не намират достатъчно потенциал за развитието си.

2) Търсене средства за ускоряване на класическата фон-Нойманова архитектура. Алгоритмите на фон ноймановите машини се базират на последователността на изпълнение на инструкциите. Тази класическа архитектура се сблъсква с ускоряването на процесорите.

Сега ще дадем обяснение как се ускоряват процесорите и как това създава конфликт с фон ноймановите архитектура. Хенри Форд създава революционен фабричен конвейер за сглобяване на автомобили. На поточната линия се слага едно шаси, върху него се работи на

пунктове. На края на линията излиза готов автомобил. Това се опитва да се приложи и при процесорите.

 Идеята е да се вкара конвейер в изпълнението на инструкциите. Тук е дадена схемата на изпълнението на идеалния конвейер. Всички обекти минават през едни и същи етапи. Конвейера трябва да се движи синхронно и се движи със закъснението на най-бавния етап. Няма взаимно влияние между обектите, които са в различните стъпки. Времето за минаване през един етап е едно и също за всички етапи. Тук разглеждаме един обект да е една инструкция.

 Всички инструкции минават през 3 етапа:

* извличане (изпълнява се от блока за управление на шината);
* дешифриране и изпълнение (АЛУ),
* записване и write back.

По-бърз конвейер се получава при 5-стъпково разделяне на изпълнението на инструкциите:

5 етапа:

* Извличане (изпълнява се от блока за управление на шината) – fetch;
* Дешифриране (управляващо устройство);
* Изпълнение (АЛУ);
* Записване;
* write back.

И петте инструкции се обработват едновременно на различните етапи. Резултатът от изпълнението и на петте след по-малко време, отколкото при предишния вариант с конвейер с 3 етапа и още по-бързо от изпълнението на петте инструкции без конвейер. Докато ако нямахме конвеирно изпълнение на инструкциите трябваше да се изчаква да завърши първата инструкция, за да се започне втората. За конвейерния процесор трябва да се добавят контролни точки и регистри за инструкцията на всеки отделен етап от конвейера, тъй като всеки етап трябва да знае с коя инструкция работи в момента. **Това усложнява процесора, тъй като за инструкциите на конвейера са нужни повече ресурси.** Друг проблем, който се среща при конвейерните процесори е възможността ресурсите от които се нуждае една инструкция да се използват от друга(структурен риск). **Също така може една инструкция да зависи от резултата на друга инструкция. Това се нарича риск на данните**. И при двата случая се получава **забавяне на конвейера**, тъй като едната инструкция трябва да изчаква другата. Когато инструкциите зависят една от друга от страна на изчисление на данни стратегии за разрешаване са няколко. Едната е да се изчака да стане готов резултата, като се замръзват предните етапи на конвейера. По време на това замразяване продължава работата само на предните етапи и това продължава докато чакащата инструкция не получи нужните ресурси(interlocks). Друг метод за решаване на проблемите с чакането при конвейера е така нареченият байпас метод. При него резултатът се подава директно на чакащата инструкция, по вътрешни пътища, веднага след изчисляването му. Този трансфер на данни прескача през стъпалата за паметта и write back, за да осигури нужните ресурси по-бързо на чакащата инструкция. Има и още един начин за решаване на тези проблеми, чрез спекулация върху зависимостта на една инструкция от друга. По този начин се гледа избягването на пускането на две зависими инструкции непосредствено една след друга по конвейера.

**ТЕМА 6. Организация и йерархия на кеш паметта. Повишаване на производителността.**

**Кешът** се състои от регистрова памет и/или статична памет, а ОП се състои от динамична памет (DRAM). При поискването на информация от ЦП се проверява първо дали тази информация я има в кеш, ако я има то имаме hit. Това означава, че ще имаме достъп до данните с малко закъснение. Ако не може да се намери търсената информация в кеш паметта, то имаме miss. Това означава, че ще трябва да се обърнем към DRAM и да търсим информацията там, което от своя страна означава, че ще имаме голямо закъснение при получаването на данните.

 Кеш паметта е съставена от множества. Всяко множество е съставено от равен брой фреймове, които пък от своя страна са съставени от идентификатор, блок и два бита за състояние. Address tag – а е част от адреса в ОП и еднозначно идентифицира разположението на данните в ОП. Фреймовете в кеша са разбъркани от големи и малки адреси. Ако трябва да **сложим копие на някакви данни** от ОП (блок) в кеша има три начина това да стане в зависимост от организацията на кеш паметта:
1. **При пълно асоциативен кеш (fully associative)** – имаме едно множество от всички фреймове и там може да се сложи на произволно място.
2. **При двупътно асоциативен кеш (set associative)** – тогава имаме няколко индексирани множества с по равен брой фреймове където по индексът на множеството се ориентираме къде трябва да се постави новия блок.
3. **Кеш с директно изобразяване (direct map)** – тук отново имаме едно множество от индексирани фреймове и блокът се поставя на точно определеното място.

За подобряване на кеша трябва да се:

* намали времето за hit,
* да се намали възможността за miss (например като се направи по- голям кеш),
* да се намали miss penalty-то (например да се сложи втори кеш между първия кеш и ОП, който да е по- голям от първият кеш, но по- малък от ОП).

Различните варианти на кеш паметта имат различни преимущества и недостатъци. А**ко размерът на кеш паметта е по – голям, това ще намали miss-овете но ще увеличи времето за hit.** Ако има по- голяма асоциативност отново може да се увеличи времето за hit. По-големият размер на блока също увеличава miss-овете.
Кеш паметта се реализира вече на 3 нива(L1, L2, L3).

* L1 е най-малка като обем и най- бърза от кеш паметите.
* L2 е малко по-голяма и по–бавна от L1
* L3 е най-голяма и най-бавна от трите кеш памети.

И трите са много по-бързи за достъп от ОП и трите съдържат само дубликати. Колкото е по-малка по обем една памет, толкова е по-бърза. Затова кеша се разделя на три нива и L1 се разделя на памет за инструкции (IS – instruction сегмент) и памет за данни (DS – data segment). Кеш паметта е направена за бързо четене. Паралелно със сравняването на тага се изтегля и информацията и при получаването и не се знае от кой фрейм е получена тя. Това не може да се приложи при писането в кеша. При писане първо се намира тагът, който ни е необходим за писане и после се пише. По този начин записът става много бавен.Решението е създаването на write buffer, който се слага на входа на кеша (вътре в него). Там стоят и адреса и данните и процесорът счита, че данните са записани. При постъпване на данни за запис в кеша първо се проверява дали има hit и ако има тогава се записват в буфера и когато им доиде реда се записват в кеша.

**ТЕМА 7. Архитектура на процесорен набор инструкции. Инструкции - формати, операции, групи инструкции и формати на данните.**

**ISA- Instruction set architecture**(архитектура на процесорен набор инструкции), е интерфейсът между хардуера и софтуера. Тази архитектура на набора инструкции определя формата на инстукциите за процесора, както и семантиката на тези инструкции.За всеки набор от инструкции има много начини за имплементация. Например една такава имплементация е архитектурата на Intel -x86. Инстукциите за програма зависят от програмния код, технологията на компилатора и ISA архитектурата -> циклите за инструкция зависят от ISA архитектурата и микроархитектурата-> времето за цикъл зависи от микроархитектурата и базовата технология. Всеки процесор работи на тактове, като тактовете на всеки процесор са с точно определена дължина. **Всеки тактов импулс има преден фронт, плато, заден фронт, падина**. Дължината на платото и падината са едни и същи. Някои процесори имат свойството да увеличават и намалят дължината на такта си. Този процес се нарича OverClocking, но не е много добре да се използва, тъй като в даден момент процесорът започва да не работи вярно. Скоростта на процесорите се определя от това колко такта имат за една секунда. **Когато има 1 такт за 1 секунда това се равнява на 1Hz**.
Един от най-важните аспекти при архитектурата на процесора е процесорното състояние. Когато една инструкция навлезе в процесора тя го заварва в някакво състояние и го оставя в друго такова. Състоянието на процесора образува един контекст за изпълнението на следващата инструкция. Основният аспект на контекста е програмния брояч. Той е един регистър който съдържа адреса на текущата, респективно на следващата инструкция. Той се увеличава с дължината на прочетената инструкция и в него се записва адреса на следващата инструкция. В началото програмният брояч съдържа адреса на инструкцията, която

трябва да изпълни. Освен програмния брояч, акумулатора или регистрите също образуват

контекста. Следваща единица са флаговете, които са възникнали при изпълнението на предишните инструкции. Тези аспекти образуват видимото състояние на процесора. Видимо за програмиста и софтуера.

Как се изпълнява една инструкция:

*  Взима се инструкцията – извлича се от паметта съдържанието на program counter
*  Декодира се – указва се кои са операндите и какво трябва да направи процесора
*  Извличане от регистъра – извлича се необходимата информация от паметта (може на части

или цялата) и се изкарва на двата входа на АЛУ-то

*  АЛУ операция – извършва се необходимата операция върху входните данни
*  Може да се установи процедура по записване на получената информация в паметта

 write back – записва се резултата обратно в някои от регистрите

 изчислява адреса на следващата инструкция(ако е нужно)

Множество от инструкции е фиксиран набор от инструкции за някоя архитектура. **Добро**

**множество от инструкции е такова множество, нa което лесно се пишат програми,** старите програми да тръгват на новите процесори и да поддържа няколко имплементации. CISC архитектури, например, са много от съвременните процесори. В публичното пространство е известно, че съществува минимален брой от инструкции, чрез които процесорът може да работи по-ефективно и има множество разработени такива проекти за процесори, но никои от тях не успява да се наложи за масовия потребител.

**ТЕМА 8. Основни програмни регистри, указатели и флагов регистър за IA-32 и Intel-64 архитектура. Системна архитектура и регистри в защитен режим: състав, предназначение и формати.**

**Архитектурата на x86 процесорите съдържа 16 регистри. Можем да ги разделим в няколко основни групи:**
1.Регистри с общо предназначение. Тези осем 32-битови регистри с общо предназначение се използват главно, за да съдържат операнди за аритметични и логически операции.
2.Сегментни регистри. Тези регистри позволяват на дизайнерите на системния софтуер да използват flat или сегментен модел на организация на паметта. Тези шест регистъра определят в определено време кои сегменти на паметта са адресируеми в дадения момент.
3.Регистри за статус и инструкции. Тези специални регистри се използват да записват и променят определени аспекти от състоянието на x86 процесора.

**Регистрите за общо предназначение на x86 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. 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 x86 процесорите дават на дизайнерите на системен софтуер гъвкавостта да избират сред множество модели на организация на паметта.
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. x86 архитектурата се възползва от това като осигурява механизми които поддържат директния acces в инструкциите и информацията не текущия модул.
Във всеки момент шест сегмента от паметта могат да бъдат достъпни за изпълняваща се x86 програма. Регистрите 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. Всеки регистър определя един определен сегмент, от сегментите които образуват програмата.
Сегментът, съдържащ в момента изпълняваща се последователност от инструкции, се знае като текущия code сегмент, той е специфициран като CS регистър**. x86 процесорите извличат всички инструкции от този кодов сегмент, използвайки като отместване(offset) съдържанието на instruction pointer.** Подпрограмни извиквания и параметри обикновено изискват регион от паметта да бъде асоцииран като стек. Всички стекови операции използват SS регистъра за да се намери стека.
Регистърът instruction pointer (EIP) съдържа адреса на отместването свързан с началото на текущия кодов сегмент на следващата инструкция която чака изпълнение. **Instruction pointer** - а не е директно видим за програмиста; той се контролира неявно от инструкции за трансфер, прекъсвания и грешки.

**ТЕМА 9. Представяне на данните. Бройни системи. Формат с фиксирана точка. Формат с плаваща точка. Обхват на представяните числа. Представяне на символи.**

Можем да представим данните под тип „форма“. Съществуват няколко форми за представяне на данни.

* **Символна форма** - служи за условно обозначаване на някакво понятие, явление или процес. Символите  могат да бъдат азбука, знаци, управляващи, графични, специални и т.н.

Символите се кодират с кодови таблици(ASCII, Unicode и др.), като за всеки символ се определя точно определена двоична комбинация, в зависимост от избрания метод за кодиране.

* **Текстова форма** - Форма за представяне на данните във вид на последователност от символи, на избран език, съдържателно разглеждана, като едно цяло. В текста основно се използва определен език и неговата азбука. При писане на програми се използват езици за програмиране.
* **Звукова форма** – форма за представяне на данните във вид на последователност от битове, получени вследствие на аналого-цифрово преобразуване на реален звук. Използва се система за аудиообработка.
* **Графична форма** - форма на представяне на информацията, предназначена за зрително възприемане. Изображението се явява  много нагледна и обемна форма за представяне на информацията. Възприемането му зависи от мозъка на човека и от възможностите на неговите органи на зрение.
* **Бройната система** е съвкупност от знаци и правила, чрез които се записват числата. Съществуват два вида бройни системи – непозиционни и позиционни.
Непозиционните бройни системи са тези, при които стойността на цифрата не зависи от нейната позиция в записа на числото. Такива бройни системи са [римската](https://bg.wikipedia.org/wiki/%D0%A0%D0%B8%D0%BC%D1%81%D0%BA%D0%B8_%D1%86%D0%B8%D1%84%D1%80%D0%B8), [гръцката](https://bg.wikipedia.org/wiki/%D0%93%D1%80%D1%8A%D1%86%D0%BA%D0%B8_%D1%86%D0%B8%D1%84%D1%80%D0%B8), милетската бройна система и др. Позиционните бройни системи са тези, при които стойността на цифрата зависи от нейното място в записа на числото.  Това означава, че стойността на цифрата в числото не е строго определена и зависи от това на коя позиция се намира съответната цифра в дадено число. ч
Основните бройни системи са двоична, десетична и 16-тична, която се ползва рядко.

**Двоичната бройна** система е системата, която се използва за представяне и обработка на числата в съвременните електронноизчислителни машини. Главната причина, поради която тя се е наложила толкова широко е, че устройства с две устойчиви състояния се реализират просто, а разходите за производство на двоични аритметични устройства са много ниски.
**Двоичните цифри 0 и 1 лесно се представят в изчислителната техника като "има ток" и "няма ток".** Двоичната система за представяне на числата в компютъра си има и недостатъци. Един от големите практически недоста­тъци, е че числата, представени с помощта на тази система са много дълги.
Числата представени в **десетична бройна** система се задават във вид, удобен за възприемане от човека. Тази бройна система има за основа числото 10. Числата записани в нея са подредени по степените на числото 10. Младшият разряд на десетичните числа се използва за представяне на единиците, следващият за десетиците, следващият за стотиците и т.н. Всеки следващ разряд е десет пъти по-голям от предния.
При **шестнайсетичните числа** имаме за основа на бройната система числото 16, което налага да бъдат използвани 16 знака/цифри за представянето на всички възможни стойности от 0 до 15 включително. За представянето на 16-чните числа се използват числата от 0 до 9 и латинските букви от A до F.

В представянето на фиксирана точка има три секции. Те са попълненото поле, целочисленото поле и дробното поле. Да приемем число като 1000.100. 1 в най-левия край е подписаното поле. Това означава дали числото е отрицателно или положително. След това 000 е цялото поле. „.“ е радикалът или десетичната точка. Числото след точката на радикала е дробното поле. При представяне с фиксирана точка броят на цифрите преди и след радикала не може да бъде променен.
Представянето с плаваща запетая може да се използва за преодоляване на ограниченията на представянето с фиксирана точка. То може да представлява много големи и много малки числа.
Разликата между фиксирана точка и плаваща точка е, че фиксираната точка има определен брой цифри, запазени за целочислената част и частичната част, а с плаваща запетая няма определен брой цифри, запазени за целочислената част и дробната част.
За целите числа без знак се заделят по 1, 2, 4 или 8 байта от паметта. В зависимост от броя на байтовете, използвани при представянето на едно число, се образуват обхвати на представяне с различна големина. Посред­ством n на брой бита могат да се представят цели числа без знак в обхвата [0, 2n-1].  Обхватът на числото 1, например, запис чрез порядък: 0 ÷ 2⁸-1, обикновен запис: 0 ÷ 255, числото 2: чрез порядък: 0 ÷ 2ⁱ⁶ -1, обикновен запис: 0 ÷ 65 535.

Символите в съвременните компютри се представят с уникална последователност от 7, 8 или 16 бита чрез кодовите таблици. Тсе преобразуват в двоични комбинации чрез кодови таблици. Примери за кодови таблици са ASCII, 7 и 8 разрядна ASCII, Unicode (UTF-16/8).

**ТЕМА 10. Модели на паметта в инструкциите: стек, акумулатор, регистрови и пр. Режими на работа и организация на паметта при IA-32 и Intel-64 архитектурите.**

Типът на вътрешната памет в процесора е най-основното разграничаване за архитектурата на набора инструкции. **Основните модели на паметта в инструкциите са стек, акумулатор и множество от регистри.** Операндите могат да бъдат посочени изрично или косвено.
При стековия модел операндите са посочени косвено на върха на стека. При него Top Of Stack register (TOS) сочи към върха на стека. Първо ще се вземе най- горната клетка на стека и ще се подаде като първи операнд на АЛУ-то, а TOS вече ще сочи към втората клетка от стека, в която седи вторият операнд. След това ще се вземе отново най- горната клетка от стека и ще се подаде като втори операнд на АЛУ-то. Резултатът, върнат от АЛУ- то, след като то свърши работа, се записва отново в стека и той застава най –отгоре на стека, тоест TOS сочи към резултата. Стекът е линейна памет на принципа FILO (first in last out) или LIFO(last in first out).
Друг модел е акумулаторният. При него се използва само един регистър - наречен акумулатор. Първия операнд се подразбира и се взима от акумулатора, втория се взима от паметта, тоест е експлицитен. Извършва се операцията в АЛУ-то и резултата се записва в акумулатора. Счита се, че при този модел има само един косвен операнд - този в акумулатора. Предварително се зарежда първият операнд в акумулатора и при събирането той се подава по магистралата между акумулатора и АЛУ-то. Това предварително зареждане съкращава дължината на инструкциите, тъй като не е нужен адрес за резултата, както и адресът за втория операнд е излишен.
Моделът **памет –** **памет**. При него всички операнди се пазят в паметта и затова няма регистри. При този модел има много възможни дължини на инструкцията, както и време за изпълнение. И въпреки че това е най-компактният модел, не е много популярен при правенето на днешните компютри. Всички операнди се намират в паметта и се подават на буферните регистри Т1 и Т2, които са невидими и са програмно – недостъпни. Резултатът от АЛУ-то също се записва направо в паметта.
Моделът **памет – регистър или регистър – памет** работи като акумулаторния, само че вместо акумулатор имаме регистри, от които взимаме единия операнд и записваме резултата на същото място. Повечето съвременни архитектури на набор инструкции използват този модел. Друг модел е моделът **регистър – регистъ**р. Известен е още и като load/store модел. При него всички операнди и резултата се записват в регистри. В този модел се използват множество инструкции, което води до по-големи програми. Ползва се при RISC компютрите. Броят на регистрите зависи от ефективността на компилатора, тъй като повечето компилатори използват определени регистри за определени операции. Регистрите се ползват, защото са бързи, има директен достъп до тях и не могат да се получат miss-ове, но има смени на контекста, с фиксирана дължина са и не може да се взима адресът на регистър, което налага ползването на паметта, в която да могат да се пазят променливи.

**ТЕМА 11. Формат на инструкциите в защитен режим на работа: префикси, кодове на операциите, Mod/rm и SIB байтове, отместване и непосредствени операнди, кодиране на адресните режими. Формат на инструкциите в IA-32e режим. Използване на REX префикс.**

Това е формата на инструкциите в 64 битов режим:


REX означава разширяване на регистрите (register extension).
Една от основните промени при 64 битовия режим е разрешението да се използват до 16 регистри. REX префиксът участва само в 64 битовия режим. Той трябва да предшества opcode полето иначе се пренебрегва. Обхватът му е от 0x40 до 0x4f. Това значи, че в 64 бита той overrides инструкцията INC и DEC, но тези инструкции имат алтернативни кодове на операциите.

1)Instruction prefixes – Чрез префикса може да се променя поведението на инструкцията, като например да се сменя сегмента, размера на думата, да се управлява цикличното изпълнение и използването на шината
2)Opcode – Кодът на операцията се кодира с 1, 2 или 3 байта. Допълнително може да се използват и 3 бита от полето ModR/M. Използването на тези полета зависи в голяма степен от типа операцията.

3)ModR/M байт –не задължителен и понякога може да съдържа част от opcode-а.
4)SIB  – – Scaled Index Base,не е задължителен

5)Displacement – Размерът на отместването се определя от полето Mod на ModR/M байта
6)Immediate - не е задължителен компонент и се използва като числова стойност от вариращи размери от байтове(byte, word, long).

**ТЕМА 12. Виртуална памет и начини за транслация на адресите.**

Виртуалната памет е множество отделни блокове от последователни адреси. Най-

разпространеният размер на една страница е 4 KB. Страниците съществуват независимо една от друга. Първоначално идеята е била необходимите в момента страници да са в реалната памет, а останалите на диска. Сега реалната (основната) памет може да е почти колкото виртуалната. Така реалното адресно пространство може да е колкото виртуалното. Адресът се състои от 2 части – **Virtual Page Number и Page Offset** – отместване в границите на

страницата. Чрез Paging Unit-a VPN-a се преобразува в Page/Physical Frame Format. Това

преобразуване се поддържа от архитектурата и чрез него преминаваме от виртуален в реален

номер на страница и съответно от виртуален в реален адрес. Виртуалната памет е част

от операционната система, тя се грижи за преобразуването на адресите и чрез механизма на

виртуалната памет OS може да manage-ва процесите. Ако процесите работят в отделни процесни пространства, то всеки процес ще се измества спрямо началото. Благодарение на механизма на виртуалната памет се реализират технологиите Multiprocessing, Multiprogramming, Security of Processes. Процесорът е този, който прави преобразуването от Virtual към реален адрес чрез вградено хардуерно устройство за Paging. Всичко останало се извършва от операционната система. Класическата виртуална памет има таблица на страниците, която се пази някъде в реалната ОП. PT Root е фиксиран регистър в процесора, в който се държи адресът на началото на таблицата. Съдържанието на този регистър може да бъде променяно. Page Table-a се състои от редове – всеки ред от таблицата съответства на номер на виртуална страница. Класическият размер е 32 бита за ред, което прави таблица – таблицата е 1 000 000 реда, което са 1М виртуални страници.

Всеки ред (entry) се състои от:

1. Present bit – показва дали е представена физическа страница, която съответства на

виртуалната, ако този бит е 1 – има, ако е 0 – няма.

2. В случай, че има(имаме hit) – трябва да има адрес на фрейма, адресът е малък. При hit се

прави Protection check. Ако тя премине се подава физическия адрес на кеша, ако не мине

извежда грешка. Ако e 0(имаме miss) - прави се проверка на ОП и ако страницата липсва

процесорът предава управлението на OS, която трябва да разположи тази страница в

реалната памет. След което отново се пуска същия адрес и пак има miss, този път

страницата се намира в ОП и се обновява P bit на 1. Процесорът отново започва да работи и

търси страницата, като прави нова, трета адресация за същата страница. Разликата е, че

този път той открива страницата и имаме hit.

3. В свободното място между P и PFN има 11 bits. Първият от тях е Dirty – ако поне веднъж е

писано нещо по страницата, този бит става 1, следователно когато трябва да се махне от

ОП, този фрейм трябва да се запише на диска(във виртуалната памет).

4. Когато в ОП няма място, операционната система взема решение коя страница да се

изхвърли. Обикновено FIFO(най-стария) или Least Recently Used(LRU). За да стане това

останалите битове се използват за брояч на обръщенията – при всяко обръщение (четене,

писане) броячът става 1. Периодично OS минава и чисти брояча, което за момент спира

работата. Ако види, че D e 1 – записва на диска, иначе – не.

**Предимството на виртуалната адресация пред сегментацията във връзка с процесите е, че смяна на процесите означава само смяна на записите в PT Root.** Тоест, ако се сменя адресното пространство, всичко остава в паметта, защото всички програми работят във виртуални адресни пространства.

**ТЕМА 13. Сегментна преадресация за 32-битов линеен адрес: таблици, селектори и дескриптори – формати и предназначение.**

В защитен режим на паметта участват два механизма на преадресация, които са включени в процесора – странична преадресация и сегментна преадресация. Сегментацията е задължителна и не може да се изключи. При сегментната преадресация логическия адрес се преобразуват в линеен адрес. При страничната преадресация, линейния адрес се преобразува във физически адрес. Тази стъпка е опционална. Тези преадресации се изпълняват по начин, който не е видим за програмиста. За да се направи сегментната преадресация процесора използва следните структури данн**и: Дескриптори (Descriptors), Таблици на дескрипторите (Descriptor tables), Селектори (Selectors), Сегментни регистри.**

Сегментния дескриптор осигурява на процесора информация която му е необходима за да изобрази логическия адрес в линеен. Дескрипторите не се създават от програмиста.

Полетата на сегментния дескриптор са:
**BASE**: Дефинира местоположението на сегмента в 4GB линейно адресно пространство. Процесора конкатенира трите фрагмента от базовия адрес за да формира една 32-битова стойност.
**LIMIT**: Дефинира размера на сегмента. Когато процесора конкатенира двете части на това поле, се получава 20-битова стойност.
**Granularity бит**: Специфицира частите, с които LIMIT полето се свързва. Когато бита е празен, лимита се интерпретира на части от по един байт.
**TYPE**: Разделяне между видовете дескриптори.
**DPL** (Descriptor Privilege Level): Използва се от защитните механизми.
**Segment-Present бит**: Ако този бит е 0, дескриптора не е валиден за използване в адресна преадресация; процесора ще издаде грешка, когато селектор за този дескриптор се зареди в сегментен регистър.

Операционни системи, които имплементират сегментирана виртуална памет изтриват този бит в един от тези случаи:
• Когато линейното адресно пространство измервано от сегмента не е изобразено в страничния механизъм.
• Когато сегмента не е в паметта.

Сегментните дескриптори се съхраняват в един от следните два типа дескрипторни таблици:
- Global descriptor table (GDT)
- Local descriptor table (LDT)
Дескрипторната таблица е просто паметен масив от 8-байтови входни данни които съдържат дескриптори. Дескрипторната таблица е с променлива дължина и може да съдържа до 8192 (2^(13)) дескриптора. Първите входни данни на GDT (INDEX=0) не се използва от процесора. Процесора намира GDT и текущата LDT в паметта с помощта на GDTR и LDTR регистрите. Тези регистри пазят базовия адрес на таблиците в линейното адресно пространство и съхраняват и сегментните лимити. Инструкциите LGDT и SGDT достъпват GDTR; инструкциите LLDT и SLDT достъпват LDTR.

Частта от логическия адрес, която съдържа селектора идентифицира дескриптора като специфицира дескрипторна таблица и индексира дескриптора в тази таблица. Селекторите могат да бъдат видими за програмите като поле в указателя, но стойностите на селекторите обикновено не се поставят от програмиста.

х86 съхранява информация от дескрипторите в сегментни регистри, така заобикаляйки нуждата да се допитва до дескрипторна таблица всеки път, когато се достъпва паметта.
Всеки сегментен регистър има видима част и невидима част. Видимите части от тези сегментни регистри се манипулират от програми, като че са просто 16-битови регистри. Невидимите части се управляват от процесора. Операциите, които зареждат тези регистри са инструкциите на нормални програми. Тези инструкции са два вида:
1. Директно зареждащи инструкции; например MOV, POP, LDS, LSS, LGS, LFS.
2. Заложени зареждащи инструкции; например far CALL и JMP. Тези инструкции неявно се отнасят към CS регистъра и го зареждат с нова стойност.

**ТЕМА 14. Странична преадресация за 32-битов физически адрес: каталози, таблици и страници – формати з а 4 К и 4 М. Странична преадресация за 36-битов физически адрес: главен каталог, каталози, таблици и страници – формати 4К, 2М и 4М.**

За да поддържа 36-битовото адресиране, необходимо е да се направят някои съществени промени на страничния механизъм. 32 битовия линеен адрес все още се използва, но се превеждат на 36-битов физически такъв. Intel избра да използва три стъпален страничен механизъм, за да поддържа PAE за 4KB страници и дву стъпален за 2MB страници. С включено PAE , CR3 сочи към малка таблица на указателите на страничните каталози. Всеки входен елемент от PDPT сочи към отделна странична директория. Всеки каталог сочи към странична таблица или директно към page frame.

В**по-голямата си част, страничната преадресация работи така: линейния адрес се трансформира във физически чрез поредица от таблици. Най-значимата промяна е допълнителната таблица и промените в самите странични структури.** Когато PAE е включено, ново ниво на търсене се явява. CR3 сочи към 4- вход на PDPT, с входни данни които сочат към отделни каталози. Размерът на страничните структури се удвоява, за да оправдае допълнителните 4 бита на базовия адрес, но виртуално са идентични на предшествениците си. Значението на тези промени както следва: Размера на входовете се удвоява за всяка структура, което води до намаляване на размера на структурата на половина. Повечето от софтуера написан, за да поддържа предишните странични имплементации, няма да бъде съвместим с PAE.

Страничната преадресация е виртуално идентична на предишната имплементация. Преадресацията е идентична за 2-MB и 4-KB страници, без факта, че указването към таблицата е премахнато в единия случай. Когато 32-битов линеен адрес е предоставен на страниците, той се разбива на следните полета:

* CR3 сочи към базовия адрес на PDPT.
* Адресните битове A[31,30] от линейния адрес формират 2-битов индекс във PDPT. Данните в PDPT съдържат базовия адрес за каталога.
* Адресните битове A[29..21] формират индекс в каталога (от неговата база). Тези входни данни (PDE) сочи към базовия адрес на таблицата или директно към 2-MB page frame.
* Адресните битове bits A[20..12] формират индекс в таблицата в нейната база. Тези входни данни на таблицата сочат към базовия адрес на 4-KB част от паметта.
* Останалите адресни битове, A[20..00] или A[11..00], формират индекс в действителната страница от паметта.

**ТЕМА 15. Инструкции за управление на програмата – условни и безусловни преходи. Преходи в подпрограми и връщане. Указване адреса на прехода. Методи за намаляване броя на преходите. Преходи в подпрограми и връщане.**

Тези инструкции се наричат инструкции за управление на програмата и включват инструкцията за условен преход (branch), безусловен преход (jump), за влизане в подпрограма (program calls), връщане на подпрограми (program return) също и командите system calls и system returns. Последните 2 команди са за достъп до системни програми и системните програми са достъпни само чрез system call.

Основните аспекти на управляващите инструкции са четири:

1) дали ще има или няма да има преход

2) как се изчислява адресът на прехода

3) link return address – запомня се адресът за връщане, тоест къде трябва да се върне програмата

4) как ще се запаси състоянието на общите регистри

Тъй като устройството за предварително разпознаване не може да разпознае прехода, се налага инструкциите за условен преход прекъсват конвейера. Такъв вид инструкции за условен преход са **compare** и **batch** инструкциите. **При compare - инстукциите за сравнение, се взимат две променливи A и B, сравняват се, но не се променят.**
Друга особеност на инструкциите за прехода е преходът към подпрограма. За да може да е

динамично, програмата трябва да знае с какъв преход да се върне – това става чрез link return register.
В самата инструкция на извиканата подпрограма не се вижда адресът на връщането. Той може да се укаже експлицитно или да бъде в регистър по премълчаване. При извикването на подпрограмата, програмният брояч се обновява със следващата инструкция в паметта, новополученият адрес се запомня във указания регистър Rx или регистър по премълчаване Re или в стека. Ако адресът се запомни в регистър по премълчаване може и имаме влизане от подпрограма в подпрограма се губи имплицитно указания адрес. Затова трябва да се запише някъде.

Най-често начинът за връзка между подпрограмите се осъществява с имплицитен регистър за

връзка, в който се записва адресът за връщане. При имплицитните регистри за връзка всяка

инструкция за преход към подпрограма записва адреса на връщане в един и същи регистър – за това трябва да се внимава при последователни преходи да не се изгубят предишни адреси.

Адресът на прехода може да се формира по различни начини:

* като относителен адрес спрямо
* програмния брояч;
* чрез базов регистър и отместване;
* чрез задаване на абсолютен адрес;
* чрез вектори.

Първият начин за формирането на адреса на прехода се нарича PC- relative и представлява

относителна адресация чрез програмния брояч и определено отместване от него . В нея няма

зададен адрес само отместването спрямо текущия. Това е най-правилният и систематичен метод, но недостатъкът му е, че при фиксирана дължина на преместването не могат да се правят далечни преходи. Също така е неудобно, защото target-ът трябва да е известен по време на писане на програмата. Това го прави неуниверсален.

Втория начин за формиране на адреса на прехода се нарича RegisterBase + displacement. При него в базов регистър се държи адресът на началото на 64КВ сегмент в паметта, а displacement-ът определя отместването спрямо този адрес. Това е стандартният начин за получаване на адрес на прехода. По този начин могат да се достъпват всички части на оперативната памет. Отместването е 16 бита. Базовите регистри могат да сочат към произволно място в оперативната памет, отделено за една програма, тоест може да се скочи навсякъде. Също така target-ът е динамичен при този метод за формиране на ардеса на прехода. Недостатъкът на този механизъм е, че адресът на прехода не е свързан с текущия адрес и това затруднява конвейера, при достигане на етапа на изпълнение.

Третият начин е чрез **директно указване на адрес на преход или указване на регистър**, който го съдържа. Този метод се нарича **absolute**. Той е най-бърз, тъй като новата стойност директно се записва в програмния брояч, но абсолютният адрес е по- дълъг и полето на адреса е по- голямо и програмата не е мобилна. Това означава, че ако програмата се зареди на друг адрес в паметта, няма да работи коректно, затова този начин не е ефективен и се ползва рядко.

Четвъртият начин на адресация е **чрез използването на специализираните векторни пространства**. Те започват от известен адрес в паметта. При векторните пространства от този адрес нататък следват 256 вектора по 4 байта, като адресирането се определя само от първия чрез jmp x, където дължината на х е 1 байт. Векторът х сочи в полето на програмата, тоест към частта от ОП, която се заема от векторното пространство. В тази адресация има неявни база и отместване. Базовият адрес на началото на векторното пространство се пази в специален регистър. Векторното пространство се използва за специални нужди, например за адресиране на преходите при прекъсване. Схемата за бърза дешифрация не може да следи дали има преход при векторната система. Тя се счита за опасна, тъй като е неудобна за конвейера. Не се счита за добър начин на програмиране, предимствата й са прескачанията при прекъсванията, тъй като при този метод имаме ясно разделение на подпрограмите от програмите и това осигурява защита на подпрограмите. Хардуерната възможност процесора да прави прескачания се нарича система за прекъсвания.

Заради конвейерното изпълнение на програмата условните преходи са трудни за реализиране и тъй като устройството за предварително разпознаване не може да разпознае прехода се налага инструкциите за условен преход да прекъсват конвейера. Това може да се предотврати като се използва инструкция с предикати.

**ТЕМА 16. Прекъсвания и изключения: вектори, таблица на прекъсванията и формат на елементите й, източници и приоритети на прекъсванията и изключенията, разрешения и забрани, обработка на прекъсвани и изключения.**

Прекъсванията и изключенията са събития, указващи за възникването на определени условия в системата или в изпълняваната програма, изискващи намесата на процесора. Възникването на тези събития кара процесора да прекрати изпълнението на текущата задача и да предаде управлението на специална процедура или задача. Прекъсванията обикновено възникват в произволен момент от време. Те са апаратни и програмни. Изключенията представляват вътрешни събития за процесора и сигнализират за някакви грешки при изпълнение на дадена инструкция: деление на нула, нарушение правилата на защита и др.
Всеки вектор на прекъсване представлява указател към подпрограмата, която обработва прекъсването. Допустими са максимум 256 прекъсвания. Векторите на прекъсванията съдържат началните адреси на програмата за обслужване. В реален режим дължината на вектора за прекъсване е 4 байта. 2 байта са предназначени за определяне началото на сегмента и се зареждат в регистъра на кодовият сегмент. Другите 2 байта определят отместването и се зареждат в instruction paint-ера. На всяко едно прекъсване се присъединява уникален идентификационен номер.
За да бъде свързан адресът на програмата за обработване на дадено прекъсване с неговия номер се използва таблица на векторите на прекъсване. В реален режим тази таблица заема първия един килобайт от оперативната памет. Тя съдържа 256 елемента с адреси на програми за обработване на прекъсвания. Всички адреси са във формат сегмент:отместване с големина 4 байта. При поява на прекъсване процесорът автоматично изтегля началния адрес на неговата стандартна процедура за обработване от таблицата. Процесорът пресмята началния адрес на даден елемент чрез умножаване на номера на прекъсването по 4. След това последователно се прочитат младшата и старшата дума от елемента в таблицата. След като в кодовия сегмент CS се зареди така получения адрес на сегмента, а в регистър IP - отместването, управлението се насочва към програмата за обслужване на прекъсването.

**Източници на прекъсване – машинна грешка, входно/изходно прекъсване, програмни прекъсвания, външно прекъсване и изкуствени прекъсвания.**
**Машинна грешка**: Прекъсването се предизвиква от схемите за контрол на хардуера. При извършване на изчисления, освен изчислителни блокове като броячи и суматори, работят и други хардуерни елементи - схеми за контрол. При overclocking се повишава вероятността за машинна грешка. Прекъсването по машинна грешка е най-привилегированото прекъсване.
**Входно/изходно прекъсване**: Източник на такова прекъсване е външен контролер. Подава се, когато завърши входно-изодна операция.
Програмни прекъсвания: Програмни прекъсвания възникват от невъзможност да бъде изпълнена някоя стъпка от цикъла на командата. Практически всяка стъпка може да не бъде изпълнена, затова ЦП преди изпълнение на предстоящата прави проверка дали има условия за нейното изпълнение. Ако не - емулира сигнал за програмно прекъсване.
Външно прекъсване: извън процесора (при натискане бутона RESET на компютъра)
Изкуствени прекъсвания: Указани в програмата.

**ТЕМА 17. Структура на задачите за IA-32 - формати на: TSS, дескриптор на TSS, TSS-gate и регистър на задачата (TR). Управление работата на задачите: изпълнение, превключване, влагане, рекурсия, изобразяване в линейното и физическо адресно пространство.**

Задачата е единица работа, която процесорът може да разпредели, изпълни и прекрати. Тя 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 адрес, границата и сегментния дескриптор на TSS се зареждат в регистъра на задачата – Task Register (TS).

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, регистри за избора на сегмент, флагов регистър, указателен регистър, поле за връзка към предходната задача, поле за избор на сегментния селектор на LDT и на CR3, 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. Регистърът на задачата (TR) съдържа 16-битовия сегментен селектор и целия сегментен дескриптор на текущата задача. Тази информация се копира от TSS-дескриптора в GDT за текущата задача.
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.

**ТЕМА 18. Челна 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.**

Front End на 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. 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 c 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:
• устройство за предсказване на преходите
• устройство за извличане на инструкциите и предварителна дешифрация
• инструкционен буфер**

**• устройство за декодиране на инструкциите**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: **•** Приема се, че преходът никога не се осъществява и се извличат следващите инструкции; **•** Приема се, че преходът винаги се осъществява и се извличат инструкцията-цел на прехода и последователността от инструкциите след нея; **•** Решението зависи от операционния код на инструкцията за условен преход.Основната цел на динамичните стратегии за предсказване на прехода (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 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. 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.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 - 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.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.

**ТЕМА 19. Изпълняваща част (Execution Core) на вътрешната архитектура IA-32: разпределител, суперскаларни компоненти и диспечеризацията им, резервационна станция.**

Изпълняващата част се състои от една или повече изпълнителни единици, контролен блок и различни други части. В съвременните процесори всяко ядро обикновено се състои от повече от една изпълнителна единица. Четириядрен процесор има четири ядра. Ако е хипер-нишка, тогава изпълнителните единици във всяко ядро могат да бъдат споделени едновременно от две нишки. Единиците за изпълнение изпълняват действителните операции като разклоняване, математически операции, операции с памет и т.н. Всяко ядро има контролен блок, който разпространява инструкции към различните изпълнителни единици в ядрото.

Изпълняващата част пренарежда микрооперациите по такъв начин, че тези чиито операнди са готови (и има налични изчислителни ресурси) да се изпълнят възможно по-скоро. Изпълняващата част може да обработи (issue) няколко микрооперации за един цикъл.

Терминът "изпълняваща част" обикновено се използва за описание на ALU, FPU, планиращ инструкции, регистри и т.н., като единица. ALU и FPU са единици за изпълнение**.**

**ТЕМА 20. Завършваща част (Retirement) на вътрешната архитектура IA-32: пренареждащ буфер, зареждане и запомняне в паметта.**

Завършващата част се грижи за това, резултатът от изчисленията да съответства на реда на постъпване на инструкции и че състоянието на системата се обновява правилно. 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.

Отделението на Завършваща част (Retirement) получава резултатите от изпълнените микро-операции от ядрото за изпълнение извън поръчката и обработва резултатите, така че архитектурното състояние да се актуализира в съответствие с оригиналния програмен ред. Когато микрооп завърши и запише резултата си, той се пенсионира. До три микрооперации могат да бъдат завършени на цикъл. Буфер за пренареждане (ROB) е единицата в процесора, която буферира завършените микро-операции, актуализира архитектурата състояние в ред и управлява подреждането на изключенията. Отделът на Завършваща част (Retirement) също следи клоновете и изпраща актуализирана информация за целевия клон на BTB. След това BTB пречиства предварително събрани следи, които вече не са е необходими.

**ТЕМА 21. Магистрали за връзка процесор-памет и вход-изход. PCI магистрала. USB магистрала. Ethernet. Осъществяване на входно-изходен обмен.**

**Магистрала на паметта** (Memory Bus) - между кеша и ОП. По този начин ОП става достъпна от три места: ЦП, контролер за директен достъп (DMAC, Direct Memory Access Controller), и адаптер за входно-изходен обмен. Входно-изходният адаптер прехвърля данни между магистралата на паметта и втора магистрала - входно-изходна магистрала, към която се свързват различни входно-изходни устройства. Тази магистрала позволяват разнообразни устройства да се включват в системата по сходен начин. Именно поради различията в устройствата обаче, те се включват посредством входно-изходни контролери, които седят между устройството и магистралата. Устройството може да има собствен контролер, който да бъде реализиран по различни начини, или може няколко групови устройства да бъдат управлявани от стандартизиран контролер.

DMA контролерът позволява трансферът на данните между устройството и ОП да се осъществява, без да се ангажира централният процесор. В този случай ЦП подава команда на DMAC, след което DMAC извършва цялата работа по прехвърлянето на информацията. Информацията минава по същия път (входно-изходен контролер, входно-изходна магистрала, входно-изходен адаптер, магистрала на паметта), но ЦП е частично или напълно свободен през това време и може да изпълнява други задачи.

**Магистралите имат 3 компонента: адресни линии, линии за данни, линии за управление.** Магистрите може да се синхронизират, но може и да не се. Ако се синхронизират: транзакцията се регулира от тактов сигнал и всички нейни етапи - заявката, потвърждаването на готовност, трансферът на данни, потвърждаването за получаване - а следователно и цялата транзакция, трябва да завършат за определен брой тактове. Магистралата може да поддържа повече от един вид транзакции, които завършват за различен брой тактове. Не се налага да се изчаква потвърждаващ сигнал, поради което, като цяло, синхронните магистрали са по-бързи от асинхронните. Повечето компютърни магистрали са синхронни.Една примерна тактова честота може да бъде 233 MHz, като тя няма нищо общо с тази на ЦП.

Ако е асинхронна: едното устройство (активното) заявява, че иска да

комуникира с друго устройство (пасивното), след което изчаква второто да заяви готовност,

и едва тогава започва трансферът - независимо дали от активното към пасивното, или

обратно. Потвърждаването за получаване на информацията в края на транзакцията

повишава сигурността на магистралата с цената на известно забавяне за потвърждаващия

сигнал. Транзакцията не протича за фиксирано време, което има известни предимства -

например обемът на предаваната информация. Примери за асинхронни магистрали са

FireWire и USB 2.0.

Транзакциите, въпреки, че се осъществяват през магистрала с много точки на достъп, са от типа point-to-point (точка-точка). Единият субект на транзакцията е ОП, а другият - устройство, свързано към шината.

Как се прехвърля управлението между устройствата, които осъществяват транзакции?

 Атомарно - двата субекта заемат магистралата само за пренасяне на една порция данни.

При 64-битова магистрала, например, две устройства се включват, предават си 64 бита

информация и се изключват. След тях се включват други две устройства, предават си 64

бита информация и се изключват, и така нататък. Тук се налага въвеждането на арбитър за

да се установи кои устройства да се включат при следващото освобождаване на

магистралата.

 Split transaction (припокриване) – междинен вариант. Два субекта се включват, предават си

1 блок информация за 1 такт време, след което се изключват. Включват се нови два

субекта, предават си 1 блок за 1 такт, изключват се, и отново. По този начин транзакциите

се припокриват във времето и могат две двойки абонати да работят едновременно.

Двойки субекти могат да бъдат ЦП : ОП, ОП : входно-изходен контролер. Пример за

реализация са VAX.

При входно-изходни транзакции, трябва да се внимава с кои области от ОП се работи. Ако се

припокриват транзакции, свързани с една и съща област от паметта, тя може да остане в

неопределено състояние. Често се случва няколко субекта по едно и също време да прехвърлят данни с паметта. Кой субект ще използва магистралата, се определя от специално устройство - арбитър. Арбитърът може да е прост и сложен. При простият приоритетът на абонатите зависи от това как са подредени и съседните абонати са с по- голям приоритет. В сложният арбитър се съдържат схеми на приоритет на субектите. Той е свързан към магистралата на паметта и всички конкуриращи се устройства имат линии за заявка към него.

Арбитражът се извършва на две стъпки.

На първата стъпка, по време на все още извършваща се предишна транзакция устройствата, на които се наложи да предават, изпращат заявки до арбитъра. От своя страна той определя източниците и приоритетите на заявките, избира едно устройство и му изпраща сигнал за избиране.

На втората стъпка избраното устройство заема магистралата за транзакция. Заемането може да бъде „късо“ (атомарно) - за 1 ширина данни, „монополно“ (цяла транзакция) - за цялата транзакция, или „split transaction“ - няколко устройства за по няколко дължини.

Кой извършва входно изходния обмен? Има три варианта.

Първият вариант е централния процесор да извършва всички операции. Но това води до допълнителното му натоварване и възможно замърсяване на кеш паметта, но пък нямаме кохерентни проблеми.

Вторият начин е да се сложи още един входно/изходен процесор. Тогава обработката става по-бърза, ЦП не се натоварва допълнително, но могат да се появят кохерентни проблеми.

Третият вариант е да се сложи DMAC контролер, който да поеме трансфера на информация от и за паметта при даден адрес. Това е бърз и обикновено лесен начин за справяне с проблема. Все пак и тук можем да имаме проблеми с кохерентността и този контролер трябва да бъде сложен на шината на паметта.

**PCI магистрала.**

PCI магистралата е високо производителна 32 или 64 битова магистрала с мултиплексирани адресни и информационни линии. **Магистралата е замислена като свързващ механизъм между контролери на периферни устройства и процесор/паметни системи.** PCI представлява хибрид между добрите страни на ISA и VL-Bus. PCI осигурява директен достъп до паметта на устройствата, които са закачени за нея, но се свързва с процес ора през FSB(front side bus) шината използвайки отделен чип. FSB свързва процесора с паметта и останалите устройства. Друга магистрала свързва процесора с L2 кеша. Тя работи на честотата на процесора. L2 кешът е интегриран към процесора, което увеличава производителността и намалява цената. Тъй като L2 кешът се намира върху процесора, тази магистрала всъщност вече дори не е магистрала. PCI може да свързва до 5 устройства външно или 10 върху платката. Също така е възможно да има повече от една PCI шина на компютър, макар това да се среща рядко. PCI чипът контролира скоростта на шината.
Интерфейса на магистралата PCI Local Bus изисква минимум от 47 пина за target устройство и 49 за master устройство за успешното предаване на информация между тях.

Описание на част от най-важните линии в 32-битова реализация (скобите обозначават броя линии от страната на устройството):

 CLK(1) (clock) – по нея се задава основният синхронизиращ сигнал;

 AD(32) (address/data) - по тези линии се движат данните и адресите. Линиите са мултиплексирани - ползват се и за адреси, и за данни;

 PAR(1) (parity) – проверка по четност на AD - xоr на всички битове от AD + PAR дава винаги 0 (или 1, зависи от реализацията)

 C/BE(4) (command / byte enable) – линия за команди. Мултиплексирана - по време на

предаването на адреси по линиите AD, тук се предават команди; по време на предаване на

данни по AD, тук се подава кои байтове, от подаваните общо 4 по AD, трябва да бъдат

възприети

 FRAME(1) - 1 линия. Чрез този канал се определя кога точно започва и кога точно завършва дадена транзакция - в момента на започването нивото на токовия сигнал по FRAME става „ниско“, в края на транзакцията FRAME става „високо“.

 IRDY(1) (initiator ready) – задаващо устройство е готово да започне транзакция с подчиненото устройство (изпраща се само от задаващото устройство)

 IDSEL(1) (initialization device select) - всяко устройство, включено към PCI шината, има

собствена IDSEL линия към PCI моста.
DEVSEL(1) (device selected) - сигнал от подчинено устройство към задаващото; означава, че

адресираното устройство е разпознато;

 TRDY(1) (translation ready) – подчиненото устройство е готово да изпълни транзакцията;

 STOP(1) - подчиненото устройство иска спиране на транзакцията;

 PERR(1) (parity error) – грешка по четност;

 SERR(1) – грешки по четност или системна грешка;

 REQ(1) (request) - индивидуална линия за заявка за транзакция - използва се от задаващото

устройство;

 GNT(1) (granted) – индивидуална линия за удовлетворяване на заявката (поискана чрез REQ) - използва се от арбитъра;

 RST(1) (reset) - Reset сигнал; например, при натискане на бутон RESET.

**USB магистрала.**

Тази магистрала е направена за лесно включване на бавни периферни устройства. Условието е кабела да е прост и не по-дълъг от 5 метра. Затова USB кабела се прави с 4 проводника, по единия тече земя по другия захранване и 2 проводника, по които върви сигнала. Данните, които текат попроводниците са последователни – бит по бит. Целта на USB-то е да се направи една магистрала, която е универсална и позволява да се работи с най-различни периферни устройства. За USB-то се поставят определени условия:

1. Потребителя не трябва да слага на устройството джъмпери или някакви ключове или устройства при опита си да го включи;

2. Потребителя не трябва да отваря компютъра, за да включи ново периферното устройство;

3. Кабелът за свързване е един и същ за всички USB устройства;

4. Трябва да има възможност устройствата да получават захранващ ток по кабела;

5. Трябва да има възможност да се включат до 127 USB устройства към един компютър;

6. Трябва да могат да се включат устройства които работят в реално време- телефони и тн;

7. USB шината трябва да позволява безпроблемно включване на ново устройство по време на

работата на останалите без да е необходимо прекъсване;

8. При включване на ново устройство не трябва да се налага ново зареждане на системата(което не е вярно при PCI);

9. Трябва разходите за производството на тези устройства да не са много големи;

USB магистралата не може да свърже два компютъра. В нея имаме един компютър, който е хост. ОС трябва да има специален модул за мениджмънт на USB магистралата. Сега всички ОС имат такъв софтуер. Хъба е вече хардуерно устройство което съдържа изводи, от които тръгва дърво за разклонение. Той е електрическо устройство, в което имаме вход и повече от един изход. В хъб-а може да се включи крайно устройство, което е потребител(USB устройство – таргет). USB-то се командва изцяло от хоста. Хостът идентифицира, когато се включи нов възел и му дава адрес от 1 до 127. USB manager- а на хоста дава адреси на устройствата включени към магистралата. Централния хъб е в мадърборда на компютъра, но може да има и допълнителен извън мадърборда. Поставянето на нов hub може да наложи допълнително захранване. Обикновено не се стига до 127 устройства.

Осъществяване на входно-изходен обмен.

За да има смисъл от компютърната система, трябва информацията да може да се въвежда за

обработка, а обработената вече информация да се извежда. Тези дейности се извършват

от входно-изходната система. Имаме 3 типа устройства: входни (Input) – например клавиатура, мишка; изходни (Output) – например принтер, монитор; съхраняващи (Storage) – например магнитен диск. (Еднократните компактдискове не са чисто нито първото, нито второто, нито третото).

Има два принципни структурни начина за включване на входно-изходна система в конфигурацията на компютъра:

Първият начин е да се постави магистрала на паметта (Memory Bus) между кеша и ОП. По този начин ОП става достъпна от три места: ЦП; контролер за директен достъп и адаптер за входно-изходен обмен. Входно-изходният адаптер прехвърля данни между магистралата на паметта и втора магистрала – входно-изходна магистрала, към която се свързват различни входно-изходни устройства. Тази магистрала позволяват разнообразни устройства да се включват в системата по сходен начин; именно поради различията в устройствата обаче, те се включват посредством входно-изходни контролери, които седят между устройството и магистралата. Устройството може да има собствен контролер, който да бъде реализиран по различни начини или може няколко групови устройства да бъдат управлявани от

стандартизиран контролер. DMA контролерът позволява трансферът на данните между устройството и ОП да се осъществява, без да се ангажира централния процесор. В този случай ЦП подава команда на DMAC, след което DMAC извършва цялата работа

по прехвърлянето на информацията. Информацията минава по същия път (входно-изходен

контролер, входно-изходна магистрала, входно-изходен адаптер, магистрала на паметта), но ЦП е частично или напълно свободен през това време и може да изпълнява други задачи.

Вторият начин е подобен на първия, но до оперативната памет има достъп през две различни

шини - едната, между ОП и кеша (тоест, ЦП), и втора, между ОП и входно-изходен процесор.

Входно-изходният процесор е подобен на централния, но изпълнява собствени програми,

управляващи входно-изходните операции.

Данните се пренасят по следната схема: ЦП привилегировано се обръща към входно-изходно устройство, със специални инструкции прочита от него данни в свой регистър (Read) и ги съхранява в кеша (Store), а след това, според стратегията на запис, може да се запише в ОП. Въпреки примитивизма на този начин, негово хубаво свойство е, че запазва консистентността на кеша - данните в кеша съответстват на тези в ОП.

Програмата на ЦП трябва да изчака входно-изходната система да свърши работата си, за да може да използва данните, върху които е работило външното устройство. Затова се налага да има някаква програма между ЦП и входно-изходната система, която да определя дали външното устройство е свършило работата си. Тази програма се нарича driver. Нейната работа е да следи регистъра за управление във входно-изходния контролер и да включи външното устройство. След това driver-а чака докато устройството работи и накрая проверява регистъра за състояние.

**ТЕМА 22.Външна памет. Твърд магнитен диск. Организация на данните и операции изпълнявани от дисковете. Повишаване производителността. Дискови масиви. Външни запомнящи устройства на CD-диск и DVD-диск и флаш-памет.**

Най-важното устройство в конфигурацията на компютъра е магнитния диск. Той е част от

йерархията на паметта и като такъв е задължителен. Състои се от метални дисковидни плочи.

Повърхността на плочата е покрита със специално метално-оксидно покритие с феромагнитни свойства, които имат цикъл на намагнитване. Феромагнита има 2 устойчиви състояния. Молекулите на този материал се разпределят по повърхността на метала и образуват домейн с осезаем магнитен поток. Промяната в магнитния поток може да индуцира слаб електричен сигнал. Когато се намагнетизира един домейн той помни посоката на намагнетизирването.

Когато тока се движи се създава една или друга посока на магнитния поток. Ако над домените се движи една намотка, то в краищата на намотката се индуцира слаб ток и протича слаб електричен поток. Той се получава при преминаване от един домейн при обратно намагнетизиран домейн. Тази намотка всъщност е главата. Състои се от подковообразна структура, около която има тънък проводник. Шпинделът е оста, около която се въртят дисковете. На всеки диск може да има до 2 работни повърхности. Към всяка работна повърхност има четяща/записваща глава. При твърдите плочи главите са летящи, т.е. профилът на главата е със специален радиус и при въртенето на диска плочата увлича газовите молекули под нея и в близост до главата се образува вятър, който при съприкосновението си с нея образува подемна сила, която отблъсква главата нагоре от плочата.

Що се отнася до въртенето - най-бавните дискове са 2400 оборота. След това следват 3600-5400.... до15000 RPM. Всички глави са свързани в карета и се движат заедно. Когато се изключи захранването на диска, главите се отдръпват навън, за да се избегне стържене. Остава само шпинделът, който след известно време спира. Ако глава опре в плочата, то тогава и главата и плочата се надраскват и стават наизползваеми. Диаметърът на

стандартния диск е 3.5 инча, като за нуждите на преносимите устройства навлизат и 2.5 инча.

При въртене главите заемат точно фиксирани позиции по радиуса на диска. Всяка такава позиция определя една писта. Пистите от всички дискове на една вертикална повърхност образуват цилиндър. Шпинделът винаги се върти с една и съща скорост при всички дискове. Щом се достигне тази нормална скорост, влиза и каретата с главите. Ъгловата скорост на диска е една и съща, но линейната на различните писти е различна. Външните писти са с по-бърза линейна скорост, вътрешните - по-ниска. Оттук - секторите във външните писти са по-разредени, вътрешните - по-нагъсто. На вътрешните писти има по- малко сектори. Броят сектори не е константен, но размерът на всеки един от тях е. Дисковете могат да започнат обработка на информация само от началото на пистата. остъпът до секторите е

последователен.

**CD** е устроен на оптичен принцип, създаден с цел да замени грамофонната плоча. Представлява поликарбонатна пластинка с диаметър 120 мм, дебелина 1.2 мм и вътрешна дупка - 15 мм. Върху тази пластинка има алуминиев слой. Информацията се нанася като се прави матрица с пробивания от инфрачервен лазер в шаблона, с дълбочина 0,8 микрометра. В тях се излива смола. Върху матрицата се нанася светлоотразителен слой, който образува спирала, върху която има равнини (land) и падини (pit). Четящата глава се движи отвътре навън (при грамофона е обратно). Дължината на спиралата е 5,6 км и се върти с постоянна линейна скорост - 120 см/сек и прави над 20хиляди оборота. За да се постига тази постоянна скорост, трябва да се променят оборотите в зависимост от това къде се намира лазера. Информацията се чете чрез лазера. Вълната се отразява, връща се, преминава през оптическа система и преминава през фотодиод. В падини отразеният лъч е по слаб, а в равнини той е по-силен.

Желанието да се увеличи обема на съхраняваната информация довежда до създаването

**на DVD дисковете** – разстоянието между две витки става по-тясно, pit-a се намалява и така има pit-ове по- начесто, сменя се лазера с по-късовълнов. Капацитетът логично става 4,7 GB, но се налагат 2 лазера в четящото устройство. Двуслойните дискове имат по 2 спирали от едната/двете страни – 1 вътрешна и 1 външна, отделени с полупрозрачен слой, а под вътрешната има непрозрачен слой. С едно въртене се четат и 2те спирали.

При **флаш паметите имаме** изолатор, в който се поставя метална пластинка, която е изцяло

плаваща в него. Тази пластинка се нарича floating gate и служи за получаване на така наречения тунелен ефект, при който изолаторът за момент пропуска електрони през себе си. По този начин плаващият gate се зарежда или разрежда и той остава в това състояние произволно дълго време. Не се разрежда с течение на времето, а само при процедурата изтриване, когато отново се използва тунелният ефект. Между source и drain протича ток само когато в плаващия gate има уловени електрони, тоест когато той е зареден. По този начин се пази информацията. Флаш паметта не се захабява при презаписване, но при изтриване на паметта се захабява изолатора и след около 1 милион изтривания изолаторът вече не изолира плаващата пластинка и паметта става неизползваема. Има два вида флаш памети: NOR-Flash и NAND-Flash, като NOR-Flash се използва за изтриваеми и програмируеми ROM памети и тя е удобна за четене на инфирмация.

**TEMA 23.Архитектури с експлицитно паралелно изпълнение на инструкциите EPIC, архитектури с много дълги инструкционни думи VLIW, динамични архитектури с много дълги инструкционни думи. Архитектурно сравнение и изпълнение на типични инструкции – CISC, RISC, VLIW, EPIC.**

 Паралелното изпълнение на инструкции може да се раздели на три основни задачи:

(1) проверка на инструкциите, връзките между тях и групиране на тези от тях, които могат да се изпълняват паралелно

(2) разпределяне на инструкциите върху отделните хардуерни изпълнителни устройства

(3) инициализиране на инструкциите и стартиране на инструкциите.

Според класификацията на Рау и Фишер задачите са: определяне на зависимости (determine dependencies), определяне на независимости (determine independencies) и свързване на ресурси. Четирите основни класа паралелни архитектури се различават именно по това каква част от задачите се изпълняват на ниво софтуер и каква на ниво хардуер.

Различават се следните видове архитектури: **суперскаларни архитектури, EPIC архитектури, динамични VLIW архитектури и VLIW архитектури**.

Суперскаларните архитектури са традиционно решение за паралелно изпълнение на инструкции. При тях, както е посочено и по-горе, е необходимо инструкциите да се декодират, да се определят зависимостите, да се групират независимите инструкции и да се разпределят върху хардуерните изпълнителни устройства, като всичко това се изпълнява хардуерно. Компилаторът генерира проста последователност от инструкции, която би могла да се изпълни върху процесор с един инструкционен конвейр.

Като противоположност може да се приеме VLIW архитектурата. Тук цялата поддръжка на паралелизма се дължи на компилаторите, които изпълняват и трите задачи. Хардуерните устройства само изпълняват групираните в слотове подоперации на всяка инструкция. Проблем се явява преносимостта на приложения. Когато компилаторът компилира приложението той го подготвя за паралелна обработка на специфична архитектура, с даден брой изпълнителни устройства, определени закъснения на отделните операции, памет, регистри и други. Така полученото приложение може да се изпълнява само върху процесори с дадената архитектура.

EPIC, или независми архитектури, е междинна архитектура между суперскаларните процесори и VLIW. Te sa с ограничени отговорности на софтуера и хардуера за осигуряване на паралелно изпълнение. При тези архитектури компилаторът открива и групира инструкциите според зависимостите им, така че да могат да се изпълняват конкурентно. Не се съставя, обаче, конкретен план на изпълнението според хардуера. По този начин EPIC архитектурите представляват комбинация от предимствата на суперскаларните и VLIW архитектурите.

Друг вид междинна архитектура между суперскаларните процесори и VLIW процесорите са динамичните VLIW процесори. При тях откриването на зависимостите между инструкциите и разпределянето на инструкциите за съответните изпълнителни устройства се извършва от компилатора, но инициализирането по време на съответните инструкции се извършва от хардуера на системата. Така се използват предимствата на компилатора за откриване на зависимости, а също така и се дава възможност на хардуерът да реагира своевременно на различни събития, които се случват по време на изпълнението и компилаторът няма как да предвиди.

Няма единен и категоричен победител сред системните архитектури и използваните инструкции. Всяка от тях има своите предимства и съответни недостатъци. За дадени нужди определена система инструкции е по-добра, за други - друга. Когато се избира какво да се използва трябва много внимателно да се анализира сложността на използвания хардуер и сложността на компилатора. Те, както става ясно от изложеното по-горе, взаимно се допълват и заместват. Комбинациите сложен хардуер – прост компилатор и обратното са двете крайности, които могат да се изберат. На практика, обаче, съществуват множество варианти, намиращи се между крайните състояния. Те са толкова, на колкото отделни и разграничими части може да се раздели процесът на превеждане на една програма от език за програмиране на високо ниво до машинен език.

**ТЕМА 24. Паралелна обработка. Хетерогенни мултипроцесори, Копроцесори**

Идеята за мултипроцесорите е възникнала като естествено следствие от необходимостта да се преодолеят ограниченията на еднопроцесорните системи. Освен подобрението на производителността, мултипроцесорите позволяват изграждането на компютърни системи с

голяма изчислителна мощност на относително ниска цена. Мултипроцесорите осигуряват възможност за непрекъснато и плавно увеличение на производителността чрез добавяне на нови процесори към компютърната система, както и постигането на по-висока стабилност на компютърните системи, тъй като при спиране на работата на един от процесорите поради повреда, останалите няма да бъдат засегнати, което ще осигури продължаването на работата на мултипроцесорната система. Основната трудност при изграждането им се състои в създаването на софтуер, който да използва техните възможности. Тя се преодолява като се създават операционни системи и софтуерни приложения, позволяващи паралелна работа.

Във всеки компютър има два основни потока – единият от команди, а другият от данни. В съответствие с това дали потоците от данни или команди са единични или представляват множество, съществуват четири големи класа:

**SISD** (Single Instruction Stream, Single Data Stream): единичен процесор (uniprocessor).

В този клас влизат компютрите, които имат един поток команди и един поток данни, тоест едно устройство за обработка на командите и едно изпълнително устройство.

**SIMD** (Single Instruction Stream, Multiple Data Stream): векторни процесори.

В тези компютри се съхранява един поток от команди, но в случая той е векторен, който инициира многочислени операции. Всеки елемент на вектора се разглежда като елемент на отделен поток от данни.

**MISD** (Multiple Instruction Stream, Single Data Stream): няма практически примери и не се

използва.

**MIMD** (Multiple Instruction Stream, Multiple Data Stream): мултипроцесори и мултикомпютри

В този клас се включват всички форми на мултипроцесорни конфигурации – от обединени

компютри до матрици от процесори. В конфигурациите са включени няколко независими и

завършени еднопроцесорни компютри, които в процеса на решаване на задачите контактуват помежду си чрез съобщения или използват обща памет. Понеже всеки от процесорите има свое управляващо устройство и може да изпълнява собствена програма, то за даден момент от време отделните процесори имат възможност да изпълняват различни операции върху различни данни.

Повечето микропроцесори са създадени главно да изпълняват различни логически операции. Прибавянето на специализиран чип, наречен аритметичен копроцесор, може да ускори сложните математически операции, изпълнявани от персоналния компютър почти сто пъти.
Копроцесорът е специално вградена схема, която работи съвместно с микропроцесора, за да ускори работата му. Най-познатите от копроцесорите са онези, които са създадени да ускорят онези операции от висок ред, които са изхвърлени от командния пакет на универсалния микропроцесор. Други копроцесори са оптимизирани да се справят с други специфични функции. Тъй като е създаден за специфично приложение, копроцесорът може да изпълнява частичните си функции много пъти по-бързо от обикновения универсален микропроцесор. Реално копроцесорът освобождава микропроцесора от необходимостта да се занимава с тежката работа. Предимствата, които носят копроцесорите, варират според работата, която се извършва с компютъра. Математическият копроцесор, например, може да ускори невероятно пресмятанията, но само определени пресмятания. Действително само 1% от работата, изпълнявана на типични персонални компютри, включва операции с плаваща запетая.

**ТЕМА 25. Компютърни клъстери и суперкомпютри.**

**Клъстерът е група компютри, обединени чрез високоскоростни връзки, представляващи от гледна точна на потребителя един компютър**. От гледна точна на мрежовите потребители клъстерът е един хост с един IP адрес.

Има различни видове клъстери:

1. Отказо-устойчиви клъстери (High-availability clusters, HA) Създават се за гарантиране на непрекъсната работа на изчислителна система. Състои се обикновено от две компютърни системи, при срив на една от системите, клъстерът продължава работата си. Позволява се включване на възстановената компютърна система в клъстера без прекратяване работата на клъстера. За изграждането на клъстер е необходимо специфично програмно осигуряване.

2. Клъстери с балансирано натоварване (Load balancing clusters) Разпределят заявките на един или повече входни възли, като ги препращат за обработка към изчислителните възли. Целта е повишаване на производителността чрез паралелно изчисляване на конкурентни заявки от различни компютърни системи (изчислителни възли). Това води и до повишаване на надеждността на системата.

3. Високопроизводителни клъстери (High-performance clusters, HPC) Използват се за решаване на специфични изчислителни задачи, например в научните изследвания и моделиране на процеси и явления. Намалява времето за изчисление, разбивайки задачата на паралелно изпълняващи се нишки.

**Суперкомпютър се нарича**[**компютър**](https://bg.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%8A%D1%80)**, който за момента на създаването си притежава най-високите показатели на капацитет на обработка на данни и в частност, скорост на изчисленията.**
Суперкомпютрите се използват за решаване на задачи, които изискват огромни количества изчисления в секунда. Такива са например проблеми от [квантовата механика](https://bg.wikipedia.org/wiki/%D0%9A%D0%B2%D0%B0%D0%BD%D1%82%D0%BE%D0%B2%D0%B0_%D0%BC%D0%B5%D1%85%D0%B0%D0%BD%D0%B8%D0%BA%D0%B0), [метеорологични прогнози](https://bg.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%B5%D0%BE%D1%80%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D1%8F) и изследвания на [климата](https://bg.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%BC%D0%B0%D1%82), моделиране на структури и свойства на химични съединения, биологични макромолекули, [полимери](https://bg.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%B8%D0%BC%D0%B5%D1%80) и [кристали](https://bg.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BB), [криптоанализ](https://bg.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7), физични симулации (например на самолет в [аеродинамичен тунел](https://bg.wikipedia.org/wiki/%D0%90%D0%B5%D1%80%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0%D0%BC%D0%B8%D1%87%D0%B5%D0%BD_%D1%82%D1%83%D0%BD%D0%B5%D0%BB), на взривяване на [ядрени оръжия](https://bg.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%B5%D0%BD%D0%BE_%D0%BE%D1%80%D1%8A%D0%B6%D0%B8%D0%B5), на [ядрен синтез](https://bg.wikipedia.org/wiki/%D0%AF%D0%B4%D1%80%D0%B5%D0%BD_%D1%81%D0%B8%D0%BD%D1%82%D0%B5%D0%B7)).

Основни потребители на суперкомпютри са големите световни университети, научноизследователски институти, военните ведомства. В днешно време суперкомпютрите са уникални машини, създавани само по поръчка от компании като [Cray](https://bg.wikipedia.org/w/index.php?title=Cray&action=edit&redlink=1), [IBM](https://bg.wikipedia.org/wiki/IBM) и [Hewlett-Packar](https://bg.wikipedia.org/wiki/Hewlett-Packard)d. Към 2012 година, най-бързият суперкомпютър е Titan с [производителност](https://bg.wikipedia.org/wiki/%D0%91%D1%8A%D1%80%D0%B7%D0%BE%D0%B4%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D0%B5_%28%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0%29) от 20 [петафлопс](https://bg.wikipedia.org/wiki/FLOPS),
Към края на 2016 г. обаче и двата най-мощни суперкомпютъра в света са китайски.