ЛЕКЦИИ ПО КОМПЮТЪРНИ АРХИТЕКТУРИ НА СПЕЦИАЛНОСТИТЕ КОМПЮТЪРНИ НАУКИ И СОФТУЕРНО ИНЖЕНЕРСТВО

ПРЕПОДАВАТЕЛ
ДОЦ. АНТОН ПОПОВ

2008/2009

1.Основни архитектурни принципи на изчислителните системи. Изпълнение на инструкциите

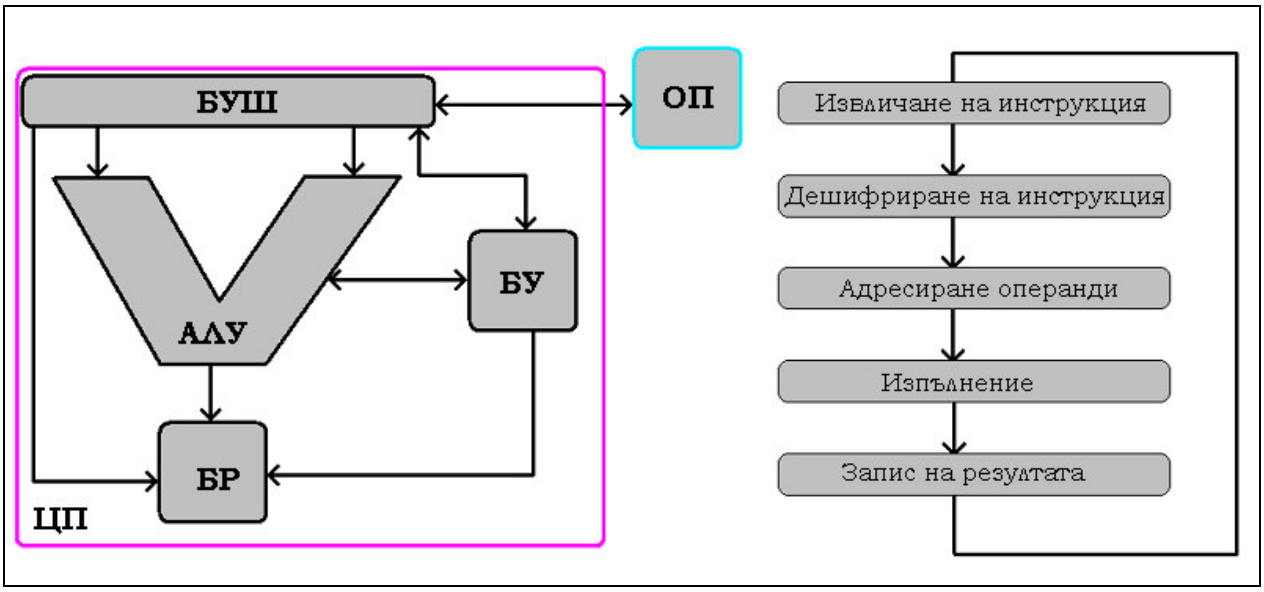
1.Увод и история

Предметът компютърни архитектури цели да обясни как работят блоковете на компютъра и каква е тяхната конструкция.

През 40-те години на 20-ти век е измислен принцип на работа на компютъра от група инженери и математици от Пентагона, които направили мощно изчислително устройство за военни цели. До тогава съществували доста опити и регистрирани патенти в тази сфера, но не били добре разработени като проекти. Проектът ENIAC на колектив с водещ унгареца Джон фон Нойман се счита за първата сериозна разработка. Той предлага принципа, по който да работи компютъра (запазил се и до днес). През 60-те и 70-те години на 20-ти век (до 1985) е имало опити да се променят основните архитектурни принципи на фон-Нойманова архитектура, но те не са били особено успешни (въпреки че имало интегрални схеми които стигат до пазара). Преди Нойман е стигнато до идеята, че нужното устройство е трябвало да бъде от тип краен автомат с множество от устойчиви състояния и начини за преминаване между тях. Сега се използват електронни елементи с 2 състояния – така наречения електронен ключ – които са включено и изключено.

Проблемът е бързо и недвусмислено да се преминава между тези състояния без междинни варианти (сега това се извършва за части от наносекундата). Не е възможно да се постигне скорост на превключване < 1ps (1/3 от скоростта на светлината в меден проводник). Преди са били използвани нажежаеми лампи (катод и 2 състояния – запушено и отпушено). Заради тези 2 състояния се работи с двоична аритметика в компютърните устройства. Имало и опити на Бебич за механичен компютър с механично колело с 10 зъба, но то е правило много бавни

механични движения => бавни пресмятания. До тогава е имало механични сметачни машини с перфокарти, но те още на са компютри.



АЛУ

В проекта ENIAC е измислено АЛУ (Аритметично Логическо Устройство), което има 2 входа и 1 изход. За вход постъпват операции с 2 операнда и се извежда 1 резултат. АЛУ е като калкулатор и извършва аритметични действия с двоични числа при познат алгоритъм на операцията. Преди компютърната ера извършваната операция е била подавана външно (чрез жакове). Операциите на АЛУ-то се кодират като двоични числа(числото се нарича код на операцията) а самата операция е една машинна инструкция. Операцията която се изпълнява над операндите се подава от БУ(блока за управление). АЛУ-то изпълнява всички аритметични и логически функции – събиране, изваждане, умножение, деление и сравняване на две числа (A>B, A≥B, A=B, A≠B, A≤B, A<B). Това устройство контролира скоростта на изчислителния процес. При по-старите микрокомпютри времето за изпълнение на една инструкция се измерваше в милисекунди, а при новите в наносекунди или в пикосекунди. Изградено е от логически елементи ИЗКЛЮЧВАЩО ИЛИ,ИЛИ,ИЛИ-НЕ и НЕ.АЛУ е комбиниран компаратор с пълен суматор. АЛУ може да бъде 2разреден,4разреден,8 и т.н. 8разредните са изградени от над 300 отделни компонента. Разряда е боря на битовете постъпващи едновременно (тоест колко бита е един операнд), освен разряд може да се нарече още размер на шината на входа, големина на машинната дума. Последователност от инструкции в паметта наричаме програма на машинен език, което всъщност е програма за АЛУ-то.. Така се получава дълго двоично число, което съдържа код на операцията и данни за операндите. Инструкцията (двоична последователност , разбираема от АЛУ) се записва в ОП, която е електронна и достъпна. В ОП се записва последователност от инструкции, които образуват компютърна програма на машинен език. Възможно е двата операнда да съществуват в самата инструкция със своите стойности. Тогава кодът на операцията се отделя и отива в АЛУ , а операндите се подават отделно на входовете, но този начин не е препоръчителен (дървено). Компютърната програма трябва да може да се изпълнява многократно върху различни данни, затова се препоръчва стойностите на операндите да са разделени от инструкцията. Въвежда се адресируема памет, която представлява номериран линеен масив от запомнящи клетки от определен брой битове. Клетките се номерират с естествените числа, започващи от 0, които се наричат адреси. В адресируемата

памет вместо съдържанието на операнди има достъп до адреса на операнда като в ОП на съответното място стои стойността. Така в инструкцията има адреси, а не стойности на операндите. Новото е, че и инструкциите се записват в адресируемата памет (първи принцип на фон Нойман).

БУ – Блок за управление

Основен елемент на ЦП е блока за управление. Негови функции са - дешифрация, определяне адреса на операндите, адресиране, пресмятане, записване на резултат.

БР – Блок регистри

С разработката на проектите за изчислителни машини се установява, че за бързодействие на АЛУ-то трябва да има достъп до малка бързодостъпна памет(това е блокът регистри). Има няколко "слота", в които се записва информацията. Един от тях е PC(IP) това е програмният брояч(или указателят към текущата инструкция), тоест той показва на кой адрес се намира инструкцията която трябва да се изпълни. За фон-Нойомановия процесор задължително трябва да има програмен брояч (наричан още указател на инструкциите - записан в регистър). В този регистър е записан адресът на инструкцията, която предстои да бъде изпълнена от ЦП. В началото външно трябва да се запише мястото от което процесорът да изпълни първата инструкция(иначе няма как да се "инициализира" компютърът при включването му). Чрез програмния брояч се извличат толкова байтове, колкото е инструкцията; обновява се програмния брояч с толкова байтове колкото е дълга инструкцията(тоест преместваме се на следващата инструкция). И така се обновява постоянно този регистър.

ОП – Оперативна памет

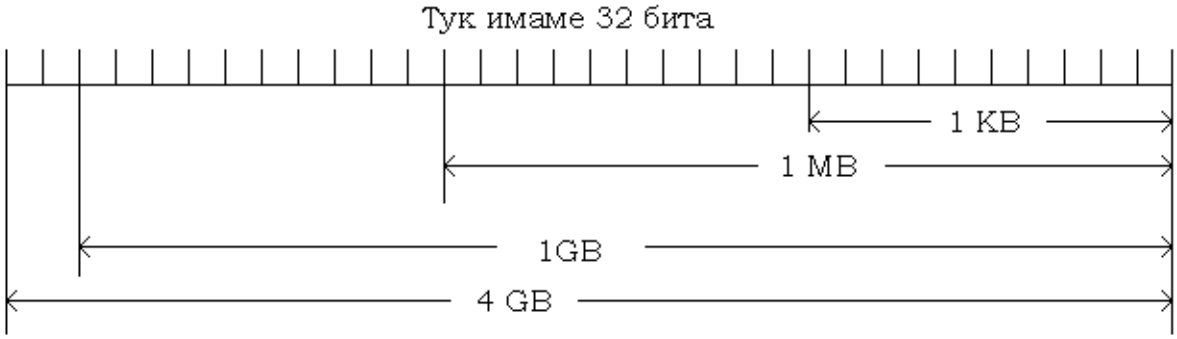
Основна характеристика на паметта е размерът и. Той се измерва в байтове. За по-големите памети се използват KB(килобайт), MB(мегабайт), GB(гигабайт), TB(терабайт). Понятието байт е въведено от IBM(преди това всеки си е взимал колкото бита си иска за основна единица памет) един байт е 8 бита, байта е основната мерна единица за памет.1 KB 1024 Byte

1 MB 1024 KB 1 048 576 Byte

1 GB 1024 MB 1 073 741 824 Byte

1 TB 1024 GB 1 099 511 627 776 Byte

Характеристика на паметта има адресната дума. Адресната дума е, грубо казано, последователност от битове, в които можем да запишем всеки адрес на клетка от паметта. На картинката по-долу са показани колко големи думи (колко битови думи) са необходими за различните размери памет.



Връзка между адресна дума на паметта и размера на паметта

Ако, да речем, адресната дума ни е с големина 32 бита, то в поле (регистър, променлива и др.) можем да запишем всяко число от 0 до (2 на степен 32) - 1 . Когато това число представлява адрес на клетка от паметта, то е ясно, че можем да адресираме клетки по 1 байт, или 4GB. Това е нашето адресно пространство. Сами се сещате, че ако физически имаме повече от 2 на степен 32 клетки памет, то губим всички останали клетки, тъй като не можем да ги адресираме.

Така големината на паметта е пряко зависима от размера на машинната дума. Не може - не искаме - да съществува адрес, който не може да бъде адресиран. А дали ще може да бъде адресиран, зависи от машинната дума. На кратко - централен процесор с 32 битова дума не може да управлява памет по-голяма от 4 GB.

2.Основни принципи на фон Нойман

Първи принцип: Съхраняват се инструкциите и данните в една обща оперативна памет, като няма формален принцип, по който двоичното съдържание да се определи дали е данни или програма. IBM определя размерът на основната адресируема единица (клетка на паметта) да е 1 байт = 8 бита.

Втори принцип: Инструкциите се изпълняват последователно. Това е изключително важно: например нека имаме операция x с операнд с адрес b и резултат с адрес a и операция y с операнд с адрес a и резултат с адрес c в следната последователност:

x : a b

y : c a

това означава, че y използва като операнд резултата от x и за да се изпълни y, x трябва да завърши ;

цялото програмиране се базира на това, че инструкциите се изпълняват последователно и ако поставим инструкция y след инструкция x, това означава, че y ще използва резултата от x.

Работата на компютъра се определя от програмите и наредбата на инструкциите в тях. Правени са опити да се изгради архитектура, при която данните също да имат роля – например, ако някакъв операнд е готов, програмата, използваща този операнд, сама да се изпълни. Оказва се, обаче, че е непостижимо за човешката мисъл да използва такива архитектури.

Обособяват се 2 основни блока – ЦП и ОП. Двата блока са свързани с шина. ЦП се управлява от инструкциите в паметта и централният процесор изпълнява инструкциите последователно от оперативната памет. Двата блока са свързани чрез шина.

3.Изпълнение на инструкциите

Разяснения по втора схема (конвейра)

Чрез програмния брояч се извличат толкова байтове, колкото е инструкцията; обновява се програмният брояч. След като е извлечена инструкцията, тя трябва да се дешифрира, пресмятат се адресите на операндите. Следва адресирането на операндите. Инструкциите много рядко работят с явно зададен адрес, в тях по-често има начини, по които ЦП да изчисли този адрес. Адресите се пращат на ОП и съдържанието на клетките чиито адреси са изпратени постъпва като вход на АЛУ. АЛУ изпълнява операцията която трябва над операндите и после резултатът се записва на определено място, определено от инструкцията. АЛУ изпълнява инструкциите асинхронно (чака постъпването на операндите на входовете му). Въведена е специална инструкция за преход(иначе процесорът просто щеше да минава еднократно линейно през ОП). Инструкцията представлява изчисление на адреса, който след това се записва в програмния брояч. Вкарването на инструкция. за преход нарушава линейността на извършваните операции от ЦП, в известен смисъл програмите се накъсват.

2. Структурна схема на изчислителните системи. Начини за предаване на данни. Кодови таблици

Една изчислителна система се състои от 4 компонента:

1. ЦП – осъществява обработка на информацията;
2. Памет – осъществява съхранение на информацията;
3. Входно-Изходна система – осъществява информационен обмен между компютъра и външния свят;
4. Вътрешни системни връзки – осъществяват комуникацията между предните 3.

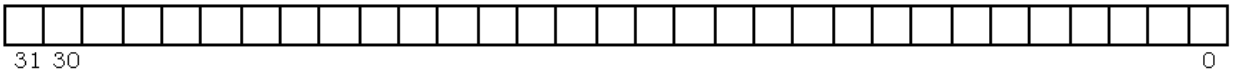
Компютърът се състои от 2 блок свързани с шина – ЦП и ОП.

1.Основни характеристики на ЦП:

А) Ширина на входа на АЛУ (колко бита постъпват едновременно на всеки вход на АЛУ-то). Размерът на шината на входа се нарича машинна дума. На входа на АЛУ-то постъпват паралелно 2 думи и като резултат излиза 1 дума. Думите са кратни на байт. При 16 битовите

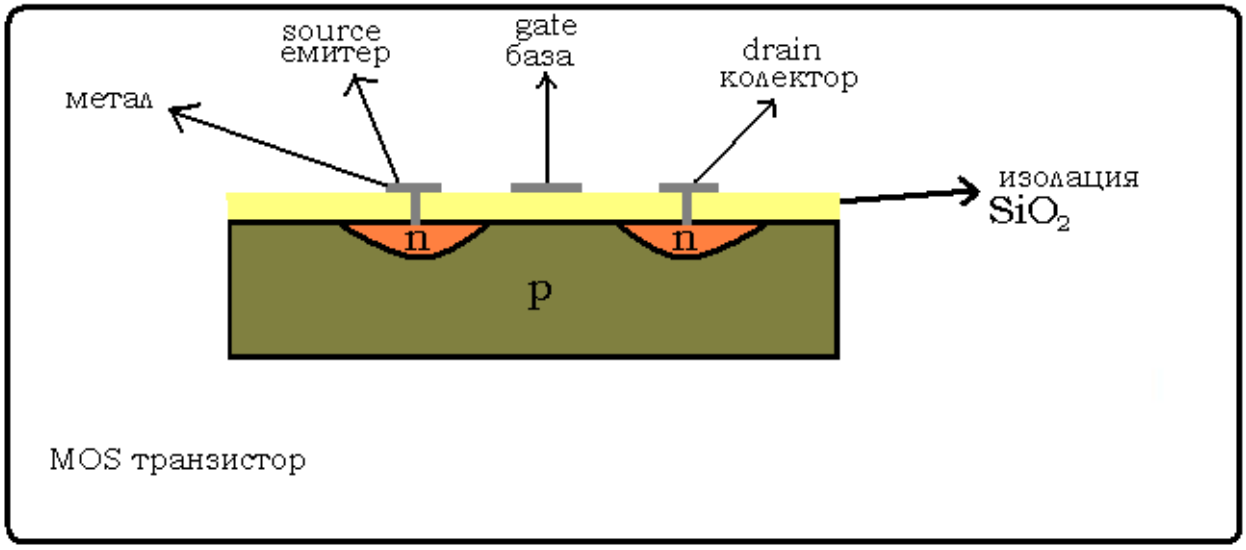
компютри думата е 16 бита, а класическата дума е 32 бита(при 32-битови компютри). Напоследък се произвеждат компютри със 64-битова машинна дума.

Б) Капацитет на паметта – измерва се в байтове, килобайтове, ... Номерът на клетката в оперативната памет е цяло число без знак (включително и 0), което се представя в двоичен вид, при което адресното пространство се разглежда в двоични разряди, така че да може да се запише максималният адрес в тази последователност – така се формира капацитетът. 32-битова дума определя 4 GB виртуално адресно пространство. Дума от 4 bytes може да адресира адресно пространство от 4 GB.



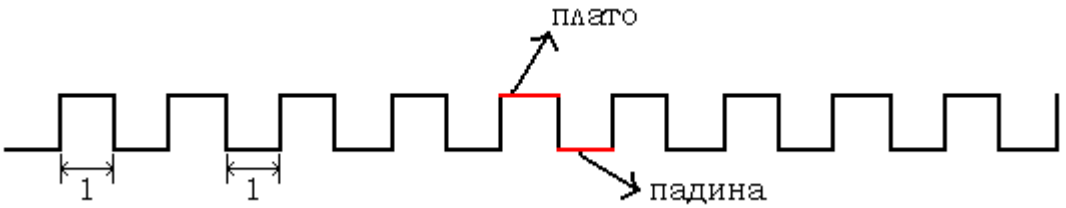
В) Гъстота на транзисторите в интегралната схема

Начин на работа на електронни блокове в компютъра (ключове) – първо за били лампи, след това полупроводникови транзистори, а сега цифрови транзистори с 2 устойчиви състояния. С времето започват да се интегрират повече транзистори в една полупроводникова пластинка, наречена чип (интегрална схема). Съвременните чипове могат да поберат до 1 милиард ключове. Преди са събирали десетки милиони. Правят се в бели стаи. Интегралната схема се състой от пластина от чист силиции (Si) покрит от SiO2. Пластината е чист полупроводник с йонна импликация се вкарват примеси и се получава n-p(p-n) проводимост. Интегралните схеми реално се състоят от различни транзистори - най-основният транзистор е MOS (metal oxide semiconductor). Ето негова схема:



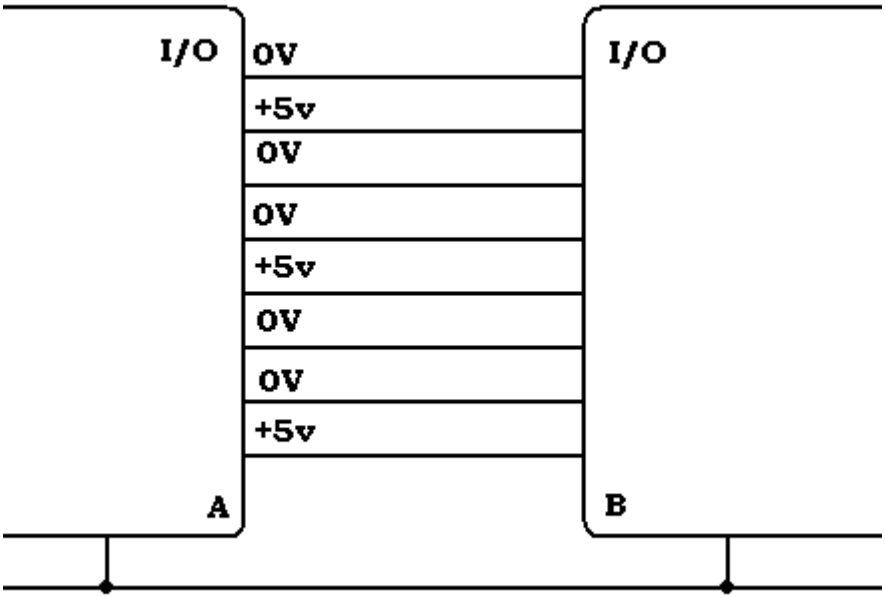
Когато на входа се подаде напрежение, проводимостта между източника и целта се променя. В случая ако подадем положително напрежение на входа, ще имаме проводимост от колектора към емитера. Ако имахме PNP транзистор, трябва да подадем отрицателно напрежение, за да имаме проводимост от емитера към колектора. В другите случаи на комбинации от напрежения и транзистори нямаме проводимост, това е идеята на транзисторите - да можем с подаване на електричен ток да контролираме предаването на електричен ток.

Всичката тази съвкупност от ключове(в интегралните схеми и нормалните схеми с много транзистори или лампи) има определен брой устойчиви състояния. Преминаването между тези състояния изисква време, зависещо от броя блокове и сложността и времето за промяна на състоянието на 1 ключ. Натрупването на ключове в блоковете изисква вътрешна синхронизация, но за да могат всички блокове да работят заедно е необходима външна синхронизация(необходимо е забавянето за промяна на състоянията да е еднакво за всички).



Г) Основна тактова поредица на процесора . Тактовата централа в централния процесор излъчва правоъгълни импулси с коефициент на запълване единица, които са кварцово стабилизирани. Така се тактува времето за преминаване в ново състояние. Периодът на тактуване е важна характеристика на ЦП. До края на такта АЛУ изкарва резултат. Времето за тактуване зависи от данните. При floating point се използва отделен изчислителен блок. Най-късата инструкция отнема минимум 3 такта като може да отнеме до 30 такта. Тактът зависи от качествата на схемите. Изчислява се с определен запас – обикновено АЛУ-то се справя за 60% от такта. OVERCLOCK – скъсяване на такта на процесора на свой риск. Периодът се измерва в наносекунди и части от наносекундата. 1 наносекунда = 1 GHz. Синхроимпулсът определя момента, в който се отварят врати, пускащи битове да влязат и излязат, през останалото време се смята. Добрите АЛУ-та работят за 1 такт. V = 1 / T.

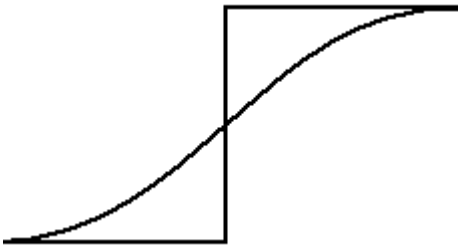
Цифрови блокове и начини на предаване на данните



Съвкупността от свързващите линии между блоковете A и B се нарича шина. В хардуерен аспект това са пътечки по които тече електричен ток(логическа единица) или не тече ток(логическа

нула). В началото логическата единица е била +5V а логическата нула - 0V, в наши дни логическата единица е +3V а нулата си е същата. Така избрани единица и нула се наричат положителна потенциална логика, всъщност положителна потенциална логика е избирането на по-голямото напрежение за единица а по-ниското за нула. Обратното избиране на нулата и единицата се нарича отрицателна потенциална логика. Официално отрицателната потенциална логика вече не се използва.

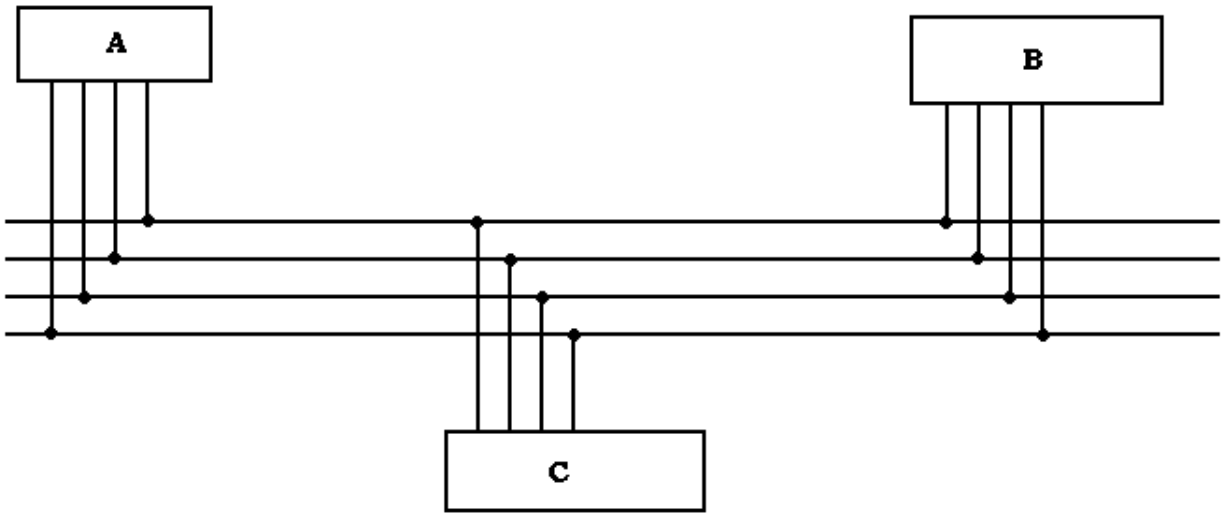
Смяната на потенциалите (напреженията) между нула и единица не става мигновено. Има необходимо време за което да се достигне до съответния потенциал. Ако измерваме с осцилоскоп съобщенията по една пътечка на шината (или казано по друг начин също толкова неясен - разглеждаме потока от битове по една от пътечките между устройство A и B) имаме "плавна" графика.



Единицата и нулата, не са отбелязани на графиката, защото в зависимост от логиката може да са различни.

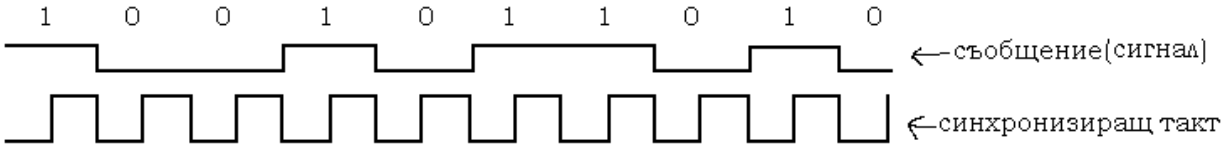
За да комуникират качествено A и B, трябва да сме сигурни, че когато B чете от A, потенциалите на всяка пътечка от шината са достигнати(тоест на всяка пътечка имаме +5V или 0V). Точно за това е необходима синхронизацията използва се синхронизиращ сигнал. При падината на сигнала A "слага" потенциалите на пътечките на шината, а по време на платото - B поема(прочита) сигнала, като A продължава да поддържа непроменен сигнала докато B го чете. Обикновено разряда на шината е равен на броя на bit-овете в машинната дума за съответната архитектура(тоест реализация на архитектура, архитектурата си е фон-Нойманова).

Комуникацията между блоковете може да е еднопосочна - винаги единия блок предава, другия приема. Двупосочна - и двете устройства могат да изпращат и приемат информация (полудуплекс), в такива случай имаме отделни устройства(намиращи се в процесора), които управляват кога съответния блок е изпращащ или приемащ. Блокът може да бъде и в състояние "изключено"(ни приема, ни предава), когато е в такова състояние блокът включва на високо съпротивление тоест, за да отчете нещо на входа си напрежението трябва да е много високо(това се постига като се увеличи напрежението на входа, представете си, че електричният ток е вода и шината е улей по който имаме вода - вдигането на напрежението е все едно да вдигнем стените на блока над водното равнище). Третото състояние на блок-а позволява направата на магистрала - шина която свързва повече от два блока, ето картинка:



Позволява направата на магистрала, защото ако всички устройства слушат по магистралата и приемат поставеното на шината, напрежението подадено от "говорещото" устройство ще падне твърде много и няма да се разчита като сигнал от другите(следвайки горния пример с водните улеи - ако всички устройства приемат вода - до по-далечните устройства няма да стигне почти никаква вода). BUS – шина, UNIBUS – обща шина (магистрала). Шината е връзка м/у 2 устройства, а магистралата м/у много.

Последователно предаване на данни:



Прим
ер за съобщение, предадено по 1 пътечка от шина. 1 бит се прехвърля за 2 такта. В този вариант синхросигналът е разделен от информационният сигнал, което е доста неудобно. През 1960 г. в Манчестър се измисля двоен Манчестърски код – наслагване на синхросигнал и информационен сигнал в един общ – вървят по една линия. На всеки такт сигналът се диференцира и се получава информацията – ако нараства – 1, ако намалява – 0. Друг начин за предаване на данните е асинхронният. При тази ситуация двете устройства са в покой и когато едното иска да предаде данни на другото, то се събужда, събужда другото устройство и му предава данните. Събуждането става като в началото се пусне стартов бит, след който се пускат 7 бита информация (в някои реализации може да не са то4но 7 бита) и накрая стопов бит, който е с дължина 1,5 бита. При получаване на стартов бит се включва вътрешен тактуващ механизъм, който определя комуникацията м/у блоковете. В наши дни в процесорите не се използва асинхронна комуникация. Асинхронната комуникация е приложима за външни устройства(и микроконтролерни системи), които не прехвърлят особено много информация по шината си. С напредването на технологиите всички устройства се наблъскват с много нови възможности и съответно асинхронната комуникация се измества от синхронна.

Съхранение на данни:

В началото съхранението на данни се е определяло според процесора, сега се определя в bytes (байтове) и кратни на байтовете.

Много важни понятия са старши и младши (most significant и last significant) byte/bit - това са съответно най-десен и най-ляв byte/bit.

Представяне на цели числа:

Числовите данни се съхраняват в 4 байта. Числата се пазят в паметта като цели двоични числа, без знак с максимална стойност 2 на степен 32. Отрицателните и положителните числа се различават по първия си(старшия) бит - ако числото е положително старшия бит е "0", ако е отрицателно - "1".В зависимост от това дали са положителни или отрицателни, числата са записани в прав или допълнителен код:

-1 е записано като 1 11111111111111111111111111111111 - допълнителен код

0 е записана като 0 00000000000000000000000000000000

1 е записано като 0 00000000000000000000000000000001 - прав код

а сега съберете -1 и 1 - получаваме 0 нали, заради тази сметка е избрано да се използва обратния код. Основен формат на данните е двоичното число със занк и със запетая (но фиксирана запетая, различно от плаваща запетая)

Операции с цели числа:

Възможните операции с тези числа са събиране, изваждане, умножение и делене. За тези операции има правила. Операцията изваждане се извършва чрез събиране, като единият операнд се обърне в допълнителен код и след това се извърши действието. При умножението, разредноста на произведението е равна на сбора на разредноста на двата множителя(разредноста е броя битове необходими за представяне на числото в двоичен вид). Деленето на числа с фиксирана запетая е се прави в процесора, целочислено - резултатът е частно и остатък. Размерността на делимото минус разредността на делителя е равна на размерността на частното. Разредността на остатъка е равна на размерността на делителя. Остатъкът има знака на делителя. При деление не е позволено делителят да е 0. Друг проблем на аритметическите операции е препълване на разрядната решетка – когато разредносттна на резултата стане по-голяма от 32 бита. Ако съберем числата 0 11111111111111111111111111111110 + 0 00000000000000000000000000000011 = 1 00000000000000000000000000000001. Това очевидно е проблем, тъй като при събиране на 2 положителни числа получаваме отрицателно. При извършване на операциите се гледат преносите на 1-ци от старшия към знакоявия bit и от знаковия bit навън. Няма препълване, ако и двата преноса са само на 0 или само на 1. Други операции които можем да извършваме са побитовите логически - AND, OR, NOT, XOR. Побитовите операции се прилагат на всеки два съответстващи по ред в думата бита - тоест ако имаме AND на операдн1 и операнд2 и 6-я бит в операнд1 е 1 а в операнд2 - 0 в резултата на 6-та позиция имаме 0.

Операции над битовите количества

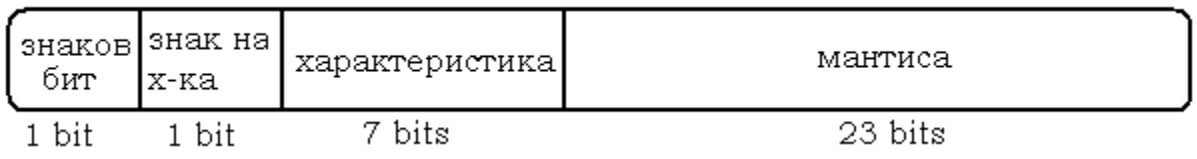
Операзии за битови количества(опрации премстващи битове), те са три типа - алгебрично(знаково), обикновенно(побитово) и ротационно. Разликата е, че ако преместим алгебрично число, винаги си запазваме старшия(знаковия бит), докато при обикновенното просто имаме преместване на битовете(без да се съобразяваме със знакови битове). Ротационното просто премества последните битове отпред или първите няколко бита отзад, в зависимост в каква посока е. Върху битовите количества са позволени набор от логически операции - and, or,xor и not. Първите три се извършват побитово за съответните битове на операндите, not е унарна - изпълнява се над еди операнд. Като резултат от тези операции не можем да получим препълване или делене на 0.

Това са операциите, които АЛУ-то може да извършва.

Нецели числа:

Представяне на нецели числа:

Други важни за компютърна обработка са числата, които не са цели - числата представяни като обикновена дроб(примерно 1/3 = 0,33333..). Представянето на нецели числа в дигиталния свят е доста сложно, в началото компютрите не са можели да извършват действия с нецели числа(не че сега го правят добре, но поне се опитват). За да се извърши операция над нецяло число, то трбява да мине в двоична дроб с плаваща запетая - числото се състои от две части - характеристика и мантиса. Характеристиката е степента на дойката, мантисата - има две форми нормализирана(може да има 0 след запетаята примерно 0,000341) и ненормализирана(веднага след запетаяра има значещо число примерно 0,341). Разбира се има операция нормализиране - мести се манитсата надясно а степента на 2-ката се овеличава. Ето и схема на представянето на числото:



Проблем на това представяне е, че имаме твърде малко памет за представяне на мантисата - 3Bytes(включваме и знаковия бит). За товв се решава да се увеличи размерът на числата с плаваща запетая на 8 Bytes с това се увеличава значително диапазона на тези числа.

Аритметически операции с нецели числа:

Аритемтиката на тези числа е сложна и стандартното АЛУ неможе да я извършва. За да се извършват операции с fp(floating point) числа е необходим блок за плаваща запетая (floating point unit) - това е отделно изчислетлно звено (подобно на АЛУ-то само за fp числа). Машинните инструкции за тези числа имат специфични особенности и съответно не се използват в масовите програми. Като цяло, за програмиране на ниско ниво(асемблер и подобни) не се използват fp числа. За програмиране на високо ниво има библиотеки, който се използват за работа с тези числа(и библиотеките са написани на асемблер, но горе долу до там стига използването на асемблерски код за fp числа) .При работата от високо ниво се използват доста често, но тези библиотеки са твърде големи за да се изпозлват при програмиране на микроконтролери и/или драйвери за микропроцесори.

Десетично представяне на числовите данни:

Вариант за преставяне на числата за компютъра е 10-тичния, той не е удобен за компютъра. За това се преминава към BCD (binary coded decimal). При BCD всеки байт се разделя на 2 парчета по 4 бита, във всяко парче се записва по едно число от 0 до 9

(има 6 комбинации в парчето от 4 бита, които не се използват).

Десетично: 0 1 2 3 4 5 6 7 8 9

BCD: 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001

Съответно преставянето на 127 в BCD е

0001 0010 0111

Очевидно този начин за представяне на десетично число е доста неефективен.

Представяне на символи:

В началото компютрите са предвидени с изчислителна цел, но с времето се е развила тяхната "псименост" и е включила в себе си и букви(преди това е имало цифри и означения на математическите операции). В началото се отделя по един байт за символ(буква). Буквите на английската азбука (големите и малките(без шрифтовете)) се събират в 7 бита. Тъй като имаме 8 бита идва въпросът какво да правим останлите 128 комбинации, които можем да използваме. В началото те не са били използвани и в кодовата таблица на символите е имало само английската азбука. В IBM правят 8 битова таблица, но за това малко по-късно сега следва разяснение как действат азбучните таблици.

Символни таблици

Азбучните представяния се базират на таблица с определена големина , като символите се групират в дълги последователности(низове /стрингове) и се представят в код, който се разбира от компютъра. Когато кодът се разпознае от компютъра му се съпоставя съответното число/буква.

Въвеждане и извеждане на символи:

За кодовите таблици в наши дни има две основни.

ASCII

ASCII(American Standart Code Infomrationlinterchange) - първоначално е 7 битова(128 символа), съдържа само английската азбука, цифрите и помощни символи. По времето когато се е използвала тази таблица се е използвало асинхронно трансфериране на данни(старт-стоп трансфер), като първият бит е оказвал, че се изпраща съобщение(данни) което било следващите 7 бита (за край се използва бит и половина), за това тази система е била много удобна и бърза. В последствие обаче се наложило да се въведат и другите писмености в символните таблици. Така на базата на старата таблица се определя нова ASCII таблица която вече е 8 битова. В началото се слага 0 или 1 в зависимост от това дали символът е към старата или новата ASCII таблица(старата таблица е била на "нулевата страница" а новата на "страница

първа"). Първата страница съдържа 128 символа, които определят една азбука, съществуват много 1-ви страници, за различните азбуки(тоест 1-та страница не е стандартизирана, дори за една писменост може да има някоко таблици).

Възприето е разделянето на ASCII таблицата да се разделя на две колонки(страници)(без причина, просто ей така са го решили).

И разбира се всички ASCII таблици са за символи, НЕ ПОДДЪРЖАТ йероглифи.

Unicode

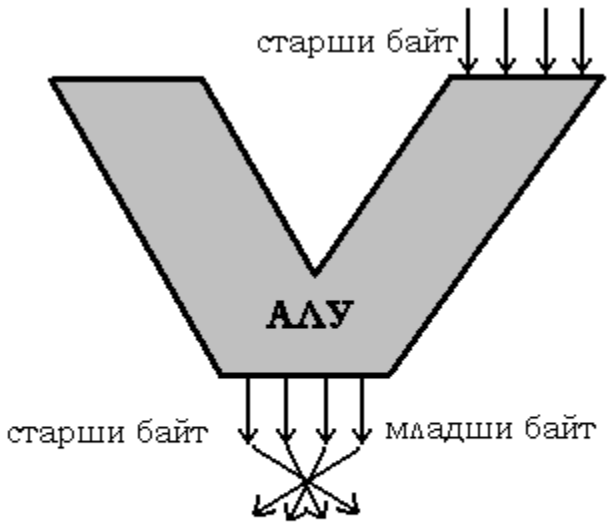
С времето се е оказало много трудно да се комуникира качествено с многото ASCII таблици, за това се преминава към Unicode - една таблица в която кодът на всеки символ се записва в 2 Bytes(което прави 64 хиляди възможни комбинации). Използват се 8-10 хиляди места в таблицата за буквите на различните азбуки, специални символи и препинателни знаци, остават 50-55 хил. места, за които се борят кои йероглифи да сложат.

Съхраняване на данните в паметта:

При съхраняването на битовете и байтовете идва въпросът как да се разположат в паметта. При запис на 4-байтово число по принцип старшият байт е нулев и след него следват 1,2 и 3. Така имаме номерация от старши към младши байтоев, като номерацията на битовете също е от старши към младши. Таза подредба се нарича Big Endian. Съществува и друга подредба – Little Endian. При нея старшият байт е последен и байтовете са разположени така: byte3 byte2 byte1 byte0.

Подаване на информация на АЛУ:

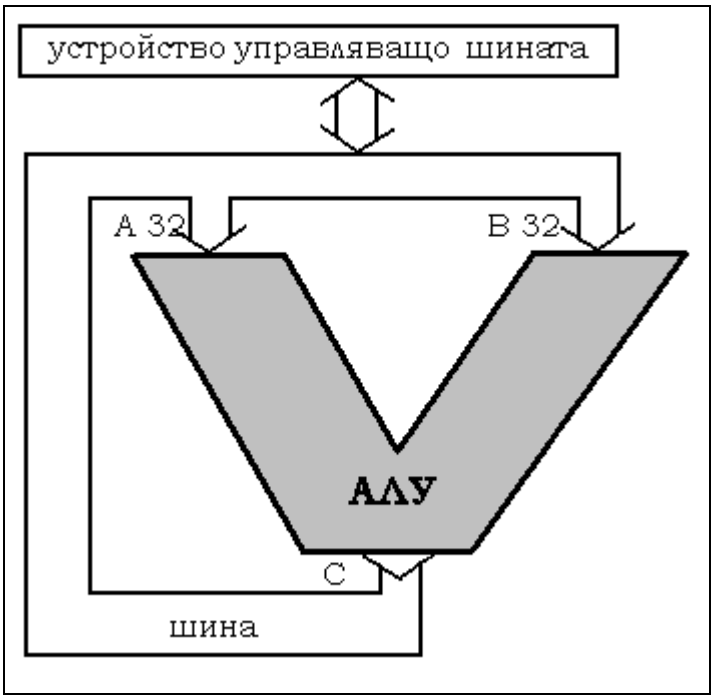
На входа на АЛУ-то числото постъпва от дясно на ляво. След извършване на операцията числото е в същата подредба, следователно при записа най-дясната част трябва да се запише първа в паметта.



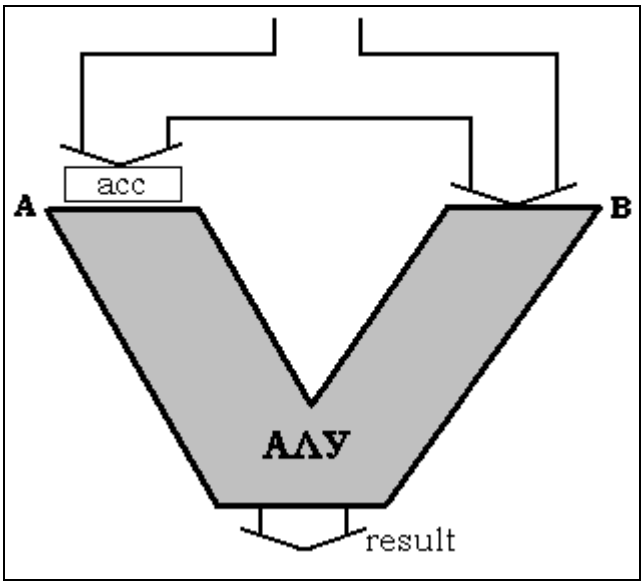
при Big Endian се разменят
както трябва байтовете
излизащи от АЛУ

3.Процесори – вътрешна структура и конвейри. SIMD и MIMD процесори.

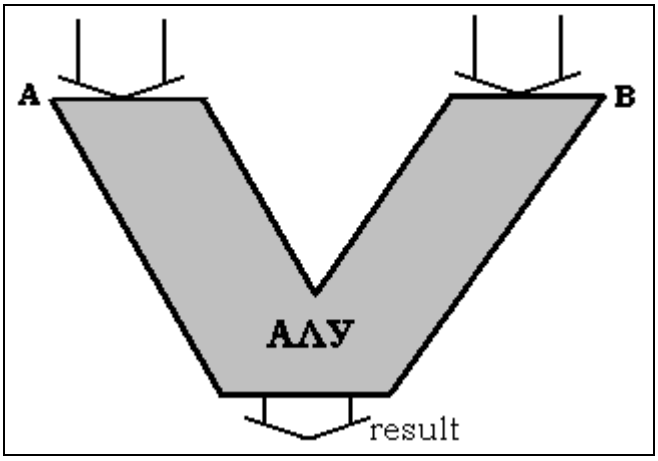
Структура на процесора – около АЛУ се изгражда процесорът, АЛУ изпълнява набор от фиксирани стандартни операции, АЛУ има 2 входа (А и Б), един изход и се конструира за асинхронна работа. Потенциална логика – по 32 потенциала на двата входа. Отварят се входовете на АЛУ при потенциалния входен (нарастващ) фронт, по време на платото се зареждат потенциалите и в зависимост от сложността на операциите до края на такта (до следващия нарастващ фронт) трябва вече резултатът да е излязъл по шината (до края на падината трябва да има резултат на изхода).



С една обща вътрешна шина за всичко, която обикаля двата входа и изхода, е възможно да се обслужи АЛУ, но тогава: на първи такт БУШ-ът трябва да предаде единия операнд на вход на АЛУ и той да се запази там във временен регистър, на втори такт излиза втори операнд на шината, отваря се временен регистър на втори вход на АЛУ, на трети такт двата регистъра си затварят входовете, отваря се изходът, в края на третия такт АЛУ е извършили изчисленията и в началото на 4-тия такт излиза резултатът, който приема отново от устройството за управление на шината.



Тази схема може да се направи и с 2 шини. Един регистър на вход А(акумулатор), на вход Б няма регистър. При първия такт постъпва операнд А и запълва акумулатора на А. На втори такт вторият операнд влиза във вход Б и се извършва операцията. В началото на 3-ти такт се отчита резултат.

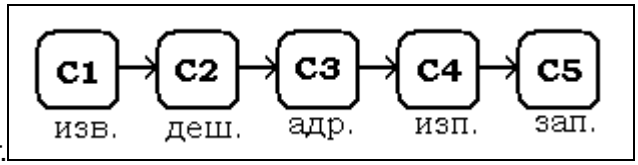


Ако имаме 3 шини – на първа шина се подава първият операнд, на втората – вторият, прави изчислението и в края на първия такт имаме резултат на третата шина. Тази реализация е най-ефективна, но изключително скъпа за изпълнение.

В процесора има един блок регистри. Броят варира от 16 до над 100. Тези вътрешни регистри са за съхранение на вътрешни резултати и операнди. Ако трябва да се събере стойността на един регистър със себе си и да се запише резултатът на същото място, то може да се стигне до грешка (объркване на вход и резултат), затова регистрите се правят двойни (2 стъпала) – чете се от второто стъпало и се записва в първото. Когато свърши такта всички първи стъпала стават втори и така се обновяват регистрите. При няколко шини блокът регистри се усложнява. Така достъпът до тях става възможен за всички шини. При една шина регистрите са прости, т.е. не е нужно да са стъпаловидни. Процесорите се конструират в зависимост от: регистрите, шините (броя) и дължината на входовете на АЛУ. Обикновено при умножение и делени процесорите

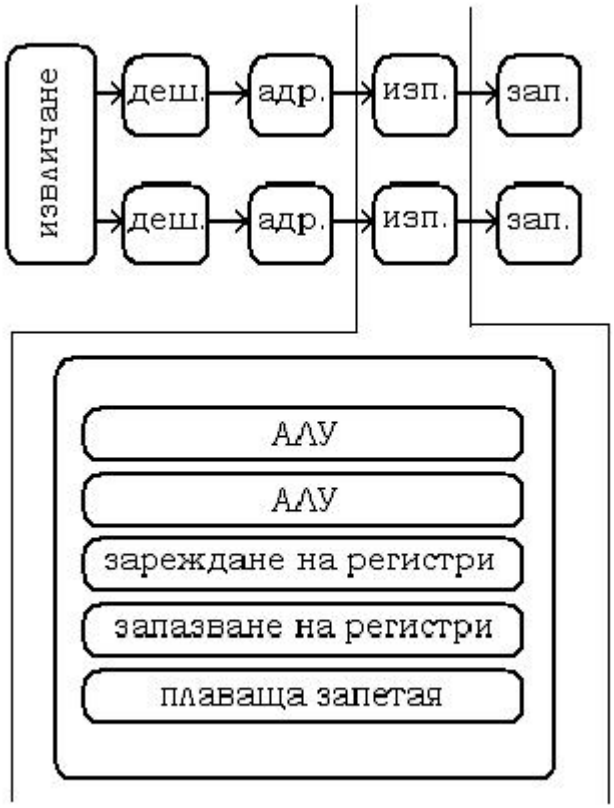
започват да работят с двойки регистри затова тези операции се извършват на повече тактове (обикновено времето се удвоява).

За да се управлява шината, т. е. кога са отворени входовете и изходите, има управляващо устройство и БУШ за работа с паметта. Има 2 варианта за процесора – 4-та шина за работа с ОП и всичко да минава през регистрите или да се предава директно от входовете на АЛУ. В стремежът за бързодействие (преди е било 1 по 1 инструкция) се развива идеята самото изпълнение да обхваща по-малко тактове. Всяка инструкция минава през 5 етапа – извличане (БУШ) , дешифрация (УУ), адресиране (адресация на ОП) , изпълнение (АЛУ) , запис (БУШ2 – пак БУШ), като всеки етап е минимум един такт.



Изпълнява се наведнъж само един блок от тези 5, а останалите чакат. Има и друг начин – влиза инструкция в C1 после минава в C2 и постъпва нова в C1. От C2->C3 , C1->C2 , ... (конвейрна обработка). За една инструкция времето остава същото, но за повече инструкции времето се съкращава (5 пъти по-бързо , тъй като се изпълняват 5 инструкции в този цикъл – от C1 до C5 (те трябва да са само линейни)). Но в тези инструкции на конвейра не трябва да има инструкции за преход, за да не се объркват операндите на различни инструкции.

Предварителна дешифрация (преди стъпка C1) е възможен допълнителен етап. Ако в конвейра следва инструкция за преход процесорът го има предвид и да не стане объркване на заредените операнди този дешефратор нарушава конвейрната линия.

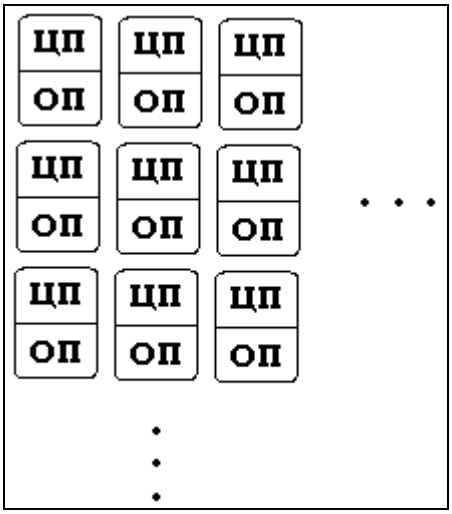


За подобряване на бързодействието се въвежда присъствието на 2 конвейра (2 физически еднакви последователности от устройства). Инструкции, които са последователни могат да минат паралелно, ако едната не използва резултатите на втората. Ако инструкциите не са последователни, а зависими една от друга, се изчакват и минават една след друга по единия конвейр. В дешифратора има специална логика, която определя дали една инструкция зависи от предходната и дали може да се пусне по паралелния конвейр. Статистиката показва, че не е голям процентът на инструкциите, които са независими и могат да минат по втория конвейр, т. е. една двойна конвейрна структура не увеличава скоростта двойно, а доста по-малко.

Въвеждане на **суперскаларен** процесор (досега скоростта се определяше по най-бавното стъпало на конвейра). Въвеждат се допълнителни елементи за процеса на изпълниние. Суперскаларната архитектура отделя в стъпалото С4 отделни функционални единици (2 индивидуални АЛУ-та например). Повечето архитектури са суперскаларни, в смисъл, че повечето имат паралелен floating point unit. Дори и първите процесори са го имали. (втората схема в горната картинка)(... да добавим нова схема ...). Целта е да има повече работещи устройства, за да се увеличи скоростта и да се разделят дейностите (отделни физически специализирани устройства, които са по-бързи). УУ решава дали да ги пуска последователно или паралелно. По-бързите и ефективни процесори отново запазват фон-Нойманова архитектура.

Тип работа на процесора:

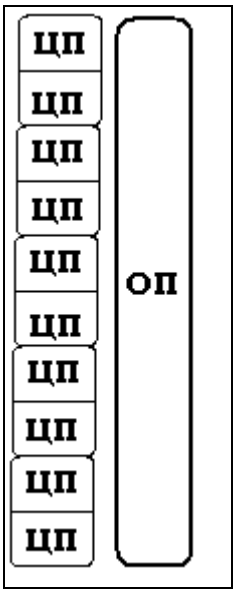
- Single Instruction Multiple DataStream (SIMD)(една инструкция – различни памети). Има една програма, която се изпълнява върху много процесори, като 1 инструкция се смята на много места с различни данни. Такива са например матричните или масивни (Array) процесори.



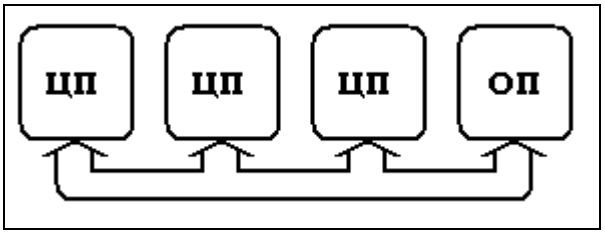
Това е масив от такива елементи, съдържащи ЦП и ОП. Една и съща инструкция се дава на всички ЦП, но върху данни от паметта на всеки от тях. Всеки процесор има копие на някакви данни. Недостатъкът е, че ако се простира върху 64x64 клетки ЦП, ОП, трябва да се смята с матрици 64x64, т.е. да се обработват на части.

Векторен Процесор – при него имаме един вектор с множество ЦП, паметта е обща, но сложно направена и всеки процесор има достъп до нея. Векторните процесори са с по-

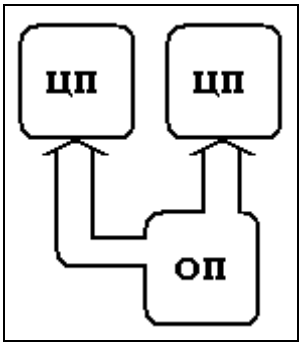
достъпна цена от Array процесорите, но все пак струват няколко милиона долара. Произвеждат се от фирмата CRAY



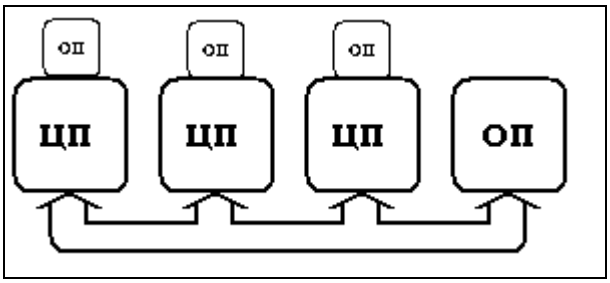
- Multiple Instruction Multiple DataStream (MIMD) – друг вариант за мултипроцесорна работа.



Това са мултипроцесорни структури – няколко ЦП свързани с магистрала към обща ОП, като процесорите не изпълняват една и съща, а отделни програми. Те могат да пресичат в даден момент работата си, но само по отношение на операндите и резултатите си. Нарича се разпределена обработка и трудно се синхронизира. Може заедно да работят 2 ЦП с общо поле на паметта – те се затварят в един корпус (двуядрен процесор). Най-често имат 2 отделни блока регистри. Обикновено първото ядро се използва са системни програми, а другото за приложни.



Друг тип мултипроцесорна архитектура е вариантът всеки процесор да има собствена памет към него и една обща оперативна памет за всички.



4.Инструкции – формати, операции, групи инструкции и формати на данните

Instruction set (IS) – набор от инструкции за даден процесор. Инструкцията трябва да има формат, т. е. възможни дължини за процесора, от какви полета се състои. Теоретично това са 4 полета – код на инструкция, адреси на операнди 1 и 2, адрес на резултата.

КОП	АО1	АО2	А на резултат

Вместо адрес в инструкцията може да присъства самия операнд, но това е неудобно, защото инструкцията е непроменяема, затова се избира начин за работа с адреси. Ако операндите са със стойности, тогава първо се чете инструкцията, после операндите и това е по-бърз начин за работа. Понякога се допуска 1 байт за някакви непроменими константи да се предава по този начин. Но 3 адресни полета са много, затова се решава адресът на резултата да се записва на мястото на първия операнд. Самото прочитане на операндите по даден адрес не влияе на стойността на самия операнд. Получаваме

КОП	АО1	АО2

Върху тази обща форма са възможни три интерпретации върху дължината на инстукцията.

- Всички инструкции за набора (IS) да се изберат с еднаква дължина (например 4 байта) . Това има предимството, че кеша ще работи много добре, тъй като ще се знае дължината на всяка следваща иннструкция без да трябва да се дешифрира тя предварително.
- Няколко определени дължини на инструкциите (2-3 фиксирани дължини 2, 4 6 или 8 байта) за дължина на инструкцията. Колко е дълга инсрукцията ще се определи от левите 2 байта на кода на операцията . Операционните кодове ще се разделят на 4 групи в този случай и по началото на кода на всяка инструкция ще се познава колко е дълга тя.

00	

01			

10					

11							

3.Кодът на операцията няма връзка с дължината . Тя е променлива за различните инструкции и се определя по време на четенето (интерпретацията). Като се стигне до определен байт ще се разбере че инструкцията е свършила.

Спецификации на операндите –определят се данни за операндите. Кодът на операцията определя само броя операнди. После имаме спецификации за първи и втори операнд и за резултат.

КОП	Спецификация O1	Спецификация O2	Спецификация Рез.

След формата на инструкцията е важно какви и колко различни операции се съдържат. Операциите са свързани с аритметичните блокове на процесора, но първичното е набор от инструкции (колко операции и какви данни). След тези определения се казва какви блокове ще са нужни на процесора.

По отношение на процесора, първите компютри са били с малък брой операции, после се увеличава броя и сложността и се стига до машините с няколко стотин (над 400) инструкции (увеличава се набора от инструкции) . Един байт е дължината на най късите, стигат до 64. При сложен набор на инструкции се програмира лесно на машинен език. Ограничен набор би бил оптимален (ортогонален – ако се махне дори само 1 инструкция вече не върши работа).

RISC – Reduced Instruction Set Computer (приблизително 50 инструкции, с които се описват всякакви обработки, но е бавно и трудоемко, трудно се пишат програми). Предимството е, че процесорът ще изпълни инструкцията много по-ефективно – за малък брой тактове (инструкционният цикъл е къс). По тази причина се решава сложността на процесора да се прехвърли към сложността на компилаторите. Те оптимизират кода (например от C към Assembler). Всички инструкции са с фиксиран размер.

5. Модели на паметта

Модели на вътрешната памет

ще разгледаме следните модели – модел със стек, модел с акумулатор, модел регистър-памет, модел памет-памет, модел регистър-регистър;

ще се спрем на триадресния случай с инструкцията add (C, A, B), която изпълнява операцията събиране на операндите, записани в A и B и записва резултатата в C;

основният проблем е как да се осъществява адресацията на операндите – стремежът е да се намали дължината на инструкцията и в нея да не участват абсолютни адреси;

с M[X] ще означаваме числото, записано на адреса X;

Модел със стек

в 60-те до средата на 70-те години имаше архитектури, които апроксимираха стековият модел на вътрешна памет;

при чисто стековия модел при адресацията се работи само със стек: (TOS – върхът на стека, S – самият стек)

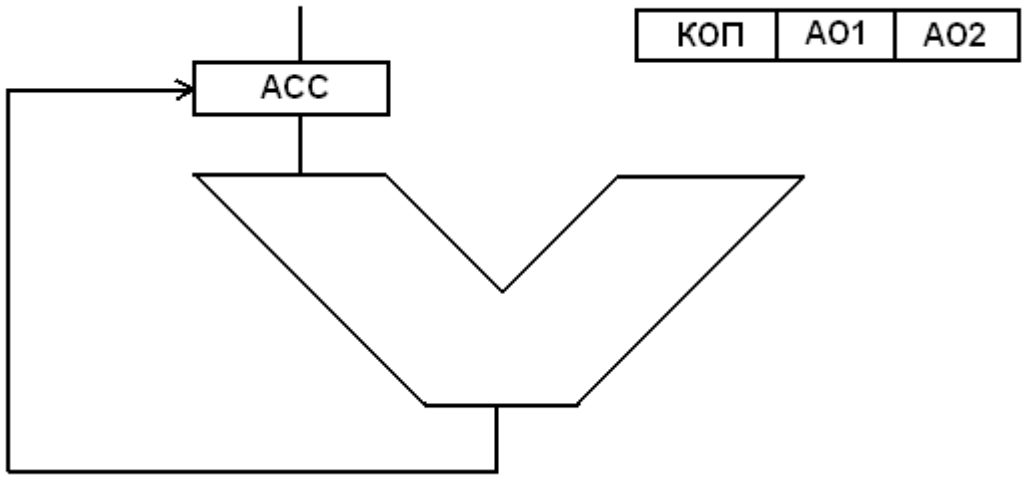
1. S[++TOS] = M[A] – стековият указател за върха се инкрементира и числото, записано на адреса A се вмъква във върха на стека;
2. S[++TOS] = M[B] – аналогично за числото, записано на адреса B;

3. при архитектура с чисто стеков модел се използва операция add без операнди; тя се изпълнява по следния начин:
    - T1 = S[TOS--] – във временен регистър се записва числото във върха на стека и след това стековият указател за върха се декрементира;
    - T2 = S[TOS--] – извлича се вторият операнд от стека и заедно с първия операнд двата влизат в ALU;
    - S[++TOS] = T1 + T2 – резултатът отново влиза във върха на стека;
4. M[C] = S[TOS--] – на адреса C се записва числото във върха, което е резултатът от операцията и стековият указател за върха се декрементира;

предимство на този модел – компактен код на програмата, тъй като инструкциите са много къси;

недостатък на модела – при въвеждане на конвейеризация (pipelining); при този подход в специално направен процесор няколко инструкции последователно влизат в процесора преди първата от тях да е завършила; основната идея е, че четирите етапи от концентуалния модел на фон-Нойман се извършват върху четири напълно различни компоненти; така в процесора могат едновременно да се обработват четири инструкции; една инструкция се изпълнява за същото време както в класическия, така и в конвейерния процесор – т.е. бързината на изпълнение е една и съща; като цяло, обаче, се повишава производителността на процесора – теоретично за едно и също време той ще изпълнява четири пъти повече инструкции; на практика това зависи от разположението на инструкциите – например, ако резултатът от първата инструкция се използва от някоя от другите инструкции, конвейерът трябва да изчака първата инструкция да завърши преди да обработи другата; оптималният случай е в процесора да влязат четири независими инструкции; при стековият модел конвейерът не може да работи ритмично – всяка инструкция ангажира стека по време на целия цикъл на изпълнението;

Модел с акумулатор



акумулаторът е подразбиращ се регистър, прикачен към процесора;

1. load A (ACC = M[A]) – акумулаторът се зарежда с първия операнд;
2. използва се операция add B с един явен операнд; тя се изпълнява по следния начин: ACC += M[B] – в ALU постъпват операндът от акумулатора и вторият операнд и резултатът от операцията отива в акумулатора;

3. store C (M[C] = ACC) – резултатът от операцията, който се намира в акумулатора се записва на адреса C;

при този модел във всички инструкции акумулаторът участва имплицитно – така всички операции за ALU имат един явен операнд;

предимство на подхода – по-малко хардуер в CPU; акумулаторът се залепя непосредствено преди левия вход на ALU и се свързват с къса, директна **магистрала (bus)**; в общия случай в CPU трябва да има поне три магистрали за да работи с ALU – по една за двата операнда и една за резултата; при модел с акумулатор схемата е следната: данните за акумулатора и за десния вход на ALU се предават по една и съща магистрала; резултатът винаги се записва в акумулатора и се предава към паметта по същата магистрала; друго предимство е, че акумулаторът не присъства в инструкциите;

моделът с акумулатор е характерен за първите микропроцесори през 60-те години по технологични причини – използват се по-малко магистрали; напоследък този модел се завръща във връзка с някои архитектурни нововъведения, които го правят удобен;

недостатъци на подхода – синхронизацията на ALU с магистралите (шините) става с помощта на врати (latch), поставени в двата му входа; в модела с акумулатор, самият акумулатор също има

врата; на първия такт вратите на ALU са затворени, вратата на акумулатора се отваря и данните от магистралата постъпват в акумулатора;

на втория такт вратата на акумулатора се затваря, вратите на ALU се отварят и по едно и също време акумулаторът предава данните на левия вход на ALU и по магистралата се предават данните на десния вход на ALU;

на третия такт ALU изчислява резултатът и той отива в акумулатора;

на четвъртия такт резултатът излиза от акумулатора по магистралата;

така резултатът излиза чак на четвъртия такт, докато при схема с три магистрали резултатът излиза още на втория такт – спестяването на хардуер води до забавяне при еднаква дължина на периода;

Модел регистър-памет

този модел е подобен на модела с акумулатор, но се използват няколко експлицитни регистъра; осъществява се по следния начин – близо до ALU се поставя блок от регистри; например нека да са 16 на брой, номерирани от R0 до R15;

1. load R1, A (R1 = M[A]) – регистърът R1 се зарежда с числото, записано в адреса A;
2. add R1, B (R1 += M[B]) – в ALU постъпват операндът от R1 и операндът на адрес B и резултат се записва в R1;
3. store C, R1 – резултатът, който е записан в R1 се записва на адрес C (в инструкцията последователността на операндите на store е R1, C);

при този метод има една основна магистрала, която излиза извън процесора; блокът от регистри се свързва с ALU с две вътрешни магистрали – те често се реализират като една, тъй като резултатът се записва в същия регистър, от който е взет първия операнд;

на първия такт вратите на ALU са затворени и по магистралата данните постъпват в регистровия блок (в случая в R1);

на втория такт по вътрешната магистрала се предава левият операнд на ALU, а по външната магистрала се предава десния операнд;

на третия такт ALU изчислява резултата, вратите му се затварят и резултатът се записва в R1;

на четвъртия такт резултатът излиза от регистровия блок по магистралата;

характерно за този модел е асиметричността на операндите – единият операнд се извлича от регистрите - четенето, писането, адресирането е много бързо, докато вторият операнд се извлича от паметта – времето за достъп до паметта обикновено е няколко цикъла;

една последователна програма, която използва междинни резултати може активно да си служи с регистрите и да не записва резултатът всеки път в паметта – по този начин се повишава бързодействието;

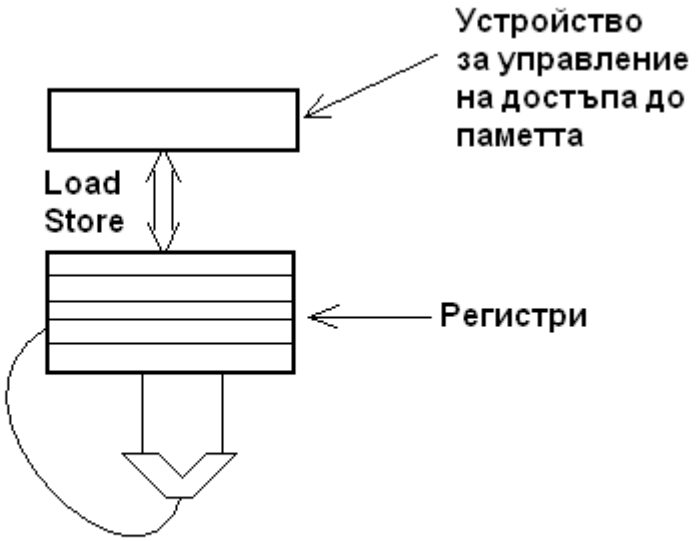
Модел памет-памет

M[C] = M[A] + M[B] – в ALU директно от паметта влизат операндите, записани на адреси A и B и резултатът се изпраща към паметта на адрес C;

такава архитектура изисква три магистрали; паметта трябва да е така направена, че да може да адресира едновременно два операнда; при класическата проста памет е нужен междинен регистър, в който да се пази първия операнд; при този модел ще има забавяне във връзка с времето на достъп; операндите са симетрични и не се използват вътрешни регистри – няма междинни инструкции; кодът на програмата е по-сбит, но процесорът се усложнява, тъй като в зависимост от вида на адресите инструкциите са с променлива дължина;

например машините VAX използват този модел;

Модел регистър-регистър



1. load R1, A (R1 = M[A]) – регистърът R1 се зарежда с числото, записано на адрес A;
2. load R2, B (R2 = M[B]) – регистърът R2 се зарежда с числото, записано на адрес B;

3. add R3, R1, R2 (R3 = R1+R2) – на ALU се подават операндите от R1 и R2 и резултатът се записва в R3;
4. store C, R3 – резултатът, който се намира в R3, се записва на адрес C в паметта;

смисълът в такава архитектура е промяната в регистровия блок – управляващият блок на регистрите и само той осъществява достъпът до паметта; ALU няма достъп до външна шина; то е свързано с регистровия блок с три вътрешни магистрали;

тази архитектура е много добра за конвейерен процесор, тъй като докато регистрите се запълват по външната магистрала, ALU може да работи по същото време с други регистри; самата операцията събиране става за един такт – в края на такта резултатът постъпва в третия регистър;

предимства на регистрите – достъпът е много бърз, проста и бърза адресация; времето за операции с регистрите е фиксирано, докато времето за достъп до паметта се движи в граници – не е точно определено;

недостатъци на регистрите – асиметричност в инструкциите и нарушаване на идеалния фон-Нойманов модел; това означава, че при всяко прекъсване на една програма (например външно прекъсване или извикване на подпрограма) има опастност да се наруши нейната цялостност, тъй като в регистрите се пазят междинни резултати на програмата; поради тази причина, целият регистров блок трябва да се запази в паметта преди да се осъществи прекъсването и когато програмата продължи състоянието на регистровия блок да се възстанови; така големият брой регистри се отразяват отрицателно на времето на изпълнение; ще отбележим, че междинните регистри не влияят на консистентността на програмата за разлика от адресуемите регистри; при моделът памет-памет, считайки че инструкциите са атомарни, програмата е винаги консистентна; друг недостатък на регистрите е че са с предварително фиксирана дължина (най-често двойна дума) – в различните набори от инструкции се използват операнди, който са символни (байтови) или битови низове; това са структури с променлива дължина, по-дълги от думата и те не могат да се държат в регистри;

при повече регистри се ускорява производителността на процесора – намалява се трафикът към паметта; колкото повече са регистрите, обаче, толкова повече се увеличава дължината на адреса на регистрите; в модела регистър-регистър най-простия модел има четири регистъра, които се адресират с два бита – по един регистър за всяка магистрала (три вътрешни и една външна);

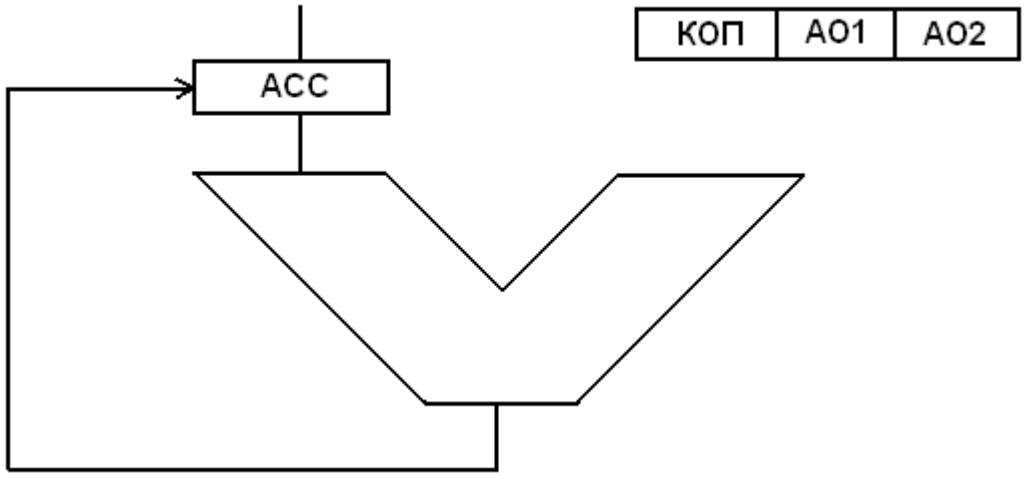
в днешно време се правят специални инструкции, които запомнят и възстановяват блока от регистри много по-ефективно отколкото изпълнение на съответния брой load и store;

(оттук надолу са го писали некадърници (ние))(нещата се повтарят но са от нашите лекции)

Модел на паметта - определя в полето на адреса какво ще има.

Първи модел:

При някои процесори, които трябва да накъсат инструкцията се въвеждат собствен вид регистри (акумулатор) на единия вход на АЛУ-то, там е операнд едно и там винаги отива резултата, следователно това е регистър по премълчаване.



Вместо двоичен код на операцията се пише мнемоничен код и се получава съдържанието на адреса A (load acc = M[A]) или

1) load A; acc = M[A] акумулаторът не присъства явно в инстр.
2) add B означава, че acc = acc + M[B]
3) store C запазване на акумулатора в C – клетка с адрес C; получава съдържанието на acc и M[C]=acc

Бинарни операции се правят с един операнд и другия по подразбиране. Всяка инструкция задължително използва acc, който непрекъснато се презарежда и изпразва. Акумулатори са се използвали в първите микропроцесори, които са били с малко регистри.

Втори модел:

Втори вариант за еднооперандна инструкция е работа със стек. Тогава за събиране се вкарва в стека операнда A (push A) , т.е. указателя към върха на стека се инкрементира и се вкарва съдържанието на клетката A: S[++tos] = M[A] ,после и B отива в стека (push B): S[++tos]= M[B]

Обработващите инструкции са без операнди – те знаят, че трябва да вземат от върха на стека.

add T1=S[tos--]

T2=S[tos--]

S[++tos]=T1 + T2

Целта е да не остават излишни неща в стека след извършване на операцията. След действията само резултатът остава в стека и той може да се вкара в паметта с pop C.

Трети модел (Register – Memory)

Следващият модел е модела на разширения акумулатор(extended acc) – вместо 1 се правят няколко набори чрез блок от регистри като те се означават с поле от ограничен брой битове(напр. поле от 4 бита - 16 регистъра)

load R1,A R1=M[A]

add R1,B R1+=M[B]

store C,R1 M[C]=R1

Адресът на първия операнд е в полето на acc, а на втория е в инструкцията. След изпълнение на инструкцията резултатът е в A. Този модел е приемлив - с компактна инструкция. Адресът на операнд 1 е с 4 бита фиксирани; остава само адреса на операнд 2 да се изчисли и да се вземе от паметта. Модел 3 и Модел 1 са с по 3 инструкции, но кодът е по-гъвкав в третия случай, може да се оптимизира и да се подобри ефективността му. Модел 2 съдържа една инструкция повече. В модел 1 всяка обработка е от 3 инструкции (зареждане, смятане , записване на резултат). В модел 3 някъде в началото се зареждат стойности, смятат се междинните и резултата си остава там, само крайният резултат се връща в паметта => само 2 обработки може да са достатъчни. Недостатъкът на третия модел е асиметричност при изпълнение. Времето за достъп до паметта е доста по-голямо и процесортъ изчаква търсения операнд, а в модел 3 за 1 такт се извършва операцията, ако е зареден първия операнд се чака 2рия. RM е удобен за ръчно програмиране.

Четвърти модел (Memory – Memory)

add C,A,B (3 адреса в паметта)

В C се записва сумата на A и B -> M[C] = M[A]+M[B]

Само една инстукция , 8 бита са нужни и се използва по-малко място в ОП като код.

Формира се адреса на 1 операнд, търси се операнд 1 и постъпва на вход 1 на АЛУ, после идва втория операнд и тогава се отварят входовете на АЛУ и до края на такта излиза резултат и той се записва в ОП. (няма акумулатори). Работи се само с клетки на паметта и не може да се обърка нещо с регистрите. Модел 4 е компактен, но не работи добре с Нойманова архитектура – работи 3 пъти с паметта.

Пети модел (Register – Register, 2 акумулатора)

Първо зареждаме единия акумулатор, след това другия, псоле ги събираме и запазваме резултата.

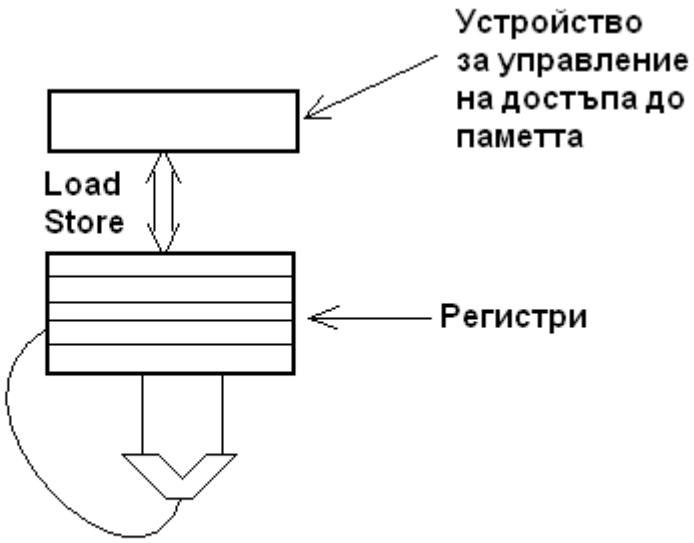
load R1, A R1=M[A];

load R2, B R2=M[B];

add R1, R2 R1+=R2;

store C, R1 M[C]=R1;

Този код е от 4 инструкции, изпълнението не е по-бързо но е симетрично, може да се оптимизира програмата и този модел да стане по-бърз. Може да действа благоприятно на конвейрната структура на процесора



Оптимизиране на програмата - след инстуркциите от модел 5 се добавя:

load R3,D

add R1, R3

а инструкцията store може да се пропусне. Инструкциите може да се преподредят като

load R3,D

да отиде на място номер 3, така докато се чака зареждането на D в R3, конвейра започва да извършва R1+R2, което е по-бързо. Веднага след първото събиране започва да се изпълнява и второто. При RISC процесорите има само R-R формат и работят затворено с регистровия блок. Отделно са инструкциите за load и store. Предимството на регистровия модел – симетрични детерминирани операции. За всяка операция се знае колко точно ще продължи, позволява се оптимизация чрез промяна на реда на операциите. В сравнение с M-M, R-R е котекстно зависим, тъй като в регистрите има начални операнди и върху тях се изпълняват множество инструкции и програмата се стреми да работи почти изцяло върху тях. Ако в даден момент програмата трябва да се прекъсне, значи, че контекста на програмата трябва да се запази, да се запази блока регистри и след това отново да може да се зареди, тоест модела е контекстно зависим. Ако работи с модела M-M, може да се прекъсне във всеки момент, тъй като всеки междинен резултат също е в паметта. (в R-R междинните резултати се пазят в акумулатори). M-M е контекстно независим модел. Друг недостатък на модел 5 е, че операндите могат да бъдат най много с размера на регистрите, тоест низови операции не се обработват с регистри. Трети недостатък е работата с променливи, които са указатели към променливи, защото промяната в

указателя означава промяна в адреса на операнда в паметта. Архитектурата Интел има 8 алкумулаторни регистъра. Други архитектури като SPARC, MIPS, ALPHA, имат по 32 регистъра. Новата архитектура Itanium на Intel е със 128. Тя е RISC архитектура. Ние използваме архитектура x86. Има стремеж за увеличаване на броя на акумулаторните регистри. Има по-високо бързодействие заради по редки обръщения към паметта (по-малко трафик). От 3 бита за 8 регистъра отиваме на 7 бита поле за адресация на 128-те регистъра. Следователно спецификаторът на ACC расте, но с повече акумулатори блокът става по-бавен, тъй като те не влизат моментално в АЛУ (повече регистри и по-бавен достъп). Много по-бавно става смяната на контекста. Повечето регистри спомагат за по-бързи програми, но по-опасни при смяна на контекста.

6.Начини за адресация на операндите. Граници на адресите

На схемата

КОП	АО1	АО2

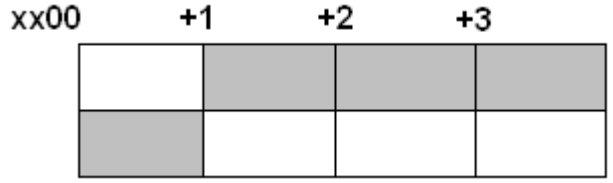
Операндите могат да се вземат чрез адреси в паметта. Но те не винаги могат да са разположени навсякъде в паметта.

изравняването означава операндът в паметта да е разположен на естествената си граница, т.е. адресът му да се дели целочислено, без остатък на размера на операнда – например, ако се работи с 32-битови думи и изравняването е по дума, това означава, че най-десните два бита на адреса на всички операнди от

една дума да са 00;

когато се конструира дадена архитектура трябва да се определи дали тя ще допуска работа с неизравнени операнди; много по-евтино и бързо е да се работи с изравнени операнди – това опростява работата на процесора и на паметта;

Ако не са изравнени и например тип данни с големина 4 байта започва от адрес кратен на 1 в процесора ще се приемат първите 4 байта, но ще се прочетат само последните 3



а последният байт ще се прочете при следващата адресация, т.е. имаме два цикъла. Неприятен фактор за изравняването е, ако не всички данни са по 4 байта, тогава се губи информация и се получават дупки в оперативната памет, които се движат заедно с пълните блокове.

Нека, например, имаме двубайтов след четирибайтов операнд. Ако има изравняване по дума, то за да се разположи четирибайтовия операнд трябва да се изпуснат двата байта след двубайтовия операнд.

към RISC машините се прилага изравняване по дума; при машините VAX няма изравняване; машините PENTIUM могат да работят с или без изравняване – процесорът може да преминава между двата начина на работа;

при архитектури които използват само изравнени операнди, адреси които не са изравнени не се възприемат и текущата инструкция се прекратява;

Режими на адресация на операндите

ще посочим начини за адресация на операндите – по какъв начин се посочва къде се намира операнда;

Непосредствен операнд (immediate)

операндът се намира вътре в тялото на инструкцията, т.е. самата стойност на операнда е поместена в нея; бележим #n; когато прочете кодът на операцията в началото на инструкцията, процесорът трябва да разбере, че става дума за непосредствен операнд – ако има такъв той директно влиза в ALU; предимството на този подход е бързодействието – избягва се формиране на адресите на операндите и извеждане на адрес върху адресната шина; недостатъци – с непосредствен операнд не може да се оперира като получател на резултати; на практика на мястото на непосредствения операнд може да се вмъкне резултат, но той няма да е достъпен за следващите инструкции; поради тази причина непосредствените операнди се ползват най-често за константи, които не се променят в хода на операциите; правилата на структурното

програмиране изрично забраняват програмите сами да променят кода си – компилаторът никога няма да пише в константата, която е непосредствен операнд;

Регистров операнд (register)

операндът се намира в регистър; бележим Rj; j не е много дълъг номер (до 7 бита в различните архитектури); предимство – адресът на операнда е много къс, освен това той моментално може да постъпи на входа на ALU; регистърът е удобен за запомняне на междинни резултати; недостатъци – регистрите са малко на брой, също те не са удобни за битови и символни низове, тъй като имат фиксиран размер и поемат числови машинни думи – 4B или 8B за числа с плаваща точка;

Памет (регистър) с отместване (displacement)

този подход е най-разпространен за изтегляне на операнд от паметта и той е най-удобен при съставяне на програми; бележим

M[Ri + #d], където Ri е регистър, d е отместване (обикновено цяло число без знак); отместването е спрямо текущата стойност на Ri;

в класическата архитектура IBM, регистърът Ri се нарича базов регистър и той съдържа базовия адрес на програмата (обикновено адресът на началото на програмата); така адресът на всеки операнд в програмата се задава с отместване спрямо базовия адрес; при писането на машинен език това е много важно – предварително не се знае от кой адрес програмата ще се изпълни, тъй като тя не се асоциира с определен адрес; програмата предварително е решила кой е базовият регистър; грижа на програмиста е когато започне да се изпълнява програмата първата инструкция да зареди базовия регистър с текущия адрес в PC; оттам нататък другите инструкции могат коректно да адресират с отместване спрямо съдържанието на базовия регистър; оказва се, че чрез този начин за адресиране могат да се изразят всички други начини за адресиране в паметта; когато отместването d е със знак може да се моделира стек;

Индиректно регистрово адресиране (indirect)

бележим M[Ri]; съдържанието на регистъра Ri определя адресът в паметта, на който се намира операнда; този начин е подобен на предния с отместване 0; разликата е, че базовият регистър

при памет с отместване е фиксиран в рамките на програмата, докато при индиректното регистрово адресиране стойността на Ri всеки път се преизчислява;

Абсолютна памет (absolute)

бележим M[#n]; при този подход в инструкцията има абсолютен адрес и той директно се използва за да се търси операнда в паметта; подхода има голяма приложимост при така наречените вектори на прекъсване – това са адреси в началото на паметта, съдържащи информация, която не се променя;

Индексна адресация (index)

бележим M [Rb + Ri]; адресът на операнда се получава като сума на два регистъра – базов регистър (Rb) и индексен регистър (Ri); в полето за адрес на операнда в инструкцията се задават адресите и на двата регистъра; смисълът на тази схема е, че базовият регистър съдържа адрес на началото на масив, а индексният регистър съдържа отместване спрямо началото на масива; съдържанието на базовия регистър е стабилно при работа с масива – променя се само индексният регистър; съществено за масивите е, че елементите им имат еднакъв размер;

Памет индиректно

бележим M[M[Rj]]; адресът на операнда се преизчислява през междинен адрес; съдържанието на регистъра, който присъства в инструкцията е адрес на стойност, която е адрес на операнда; този подход е приложим при работа със списъчни структури и други по-сложни структури за които се изисква работа с указатели;

Индексно с отместване (scaled)

бележим M[Rb + Ri*s + #d]; тук s е коефициент, който показва размера на елементите на масива; този начин е подобен на индексната регистрация, но стойността на регистъра Ri е истинският индекс на елемента в масива; в общия вид има и отместване #d;

Регистрово с обновяване (update)

бележим M[Ri = Ri + #n]; този подход е въведен най-вече за нуждите на стековете и при него има два режима:

- автоинкремент (autoincrement) – при него първо се адресира със стойността на Ri и след това Ri се инкрементира автоматично с дължината на операнда в рамките на същата инструкция; това спестява увеличаване на Ri (например при индиректната регистрова адресация);
- автодекремент (autodecrement) – при него първо се намалява регистъра Ri с дължината на операнда и след това се адресира с получената стойност на Ri;

#n не присъства явно в инструкцията, тя само го определя; този режим за адресация се използва в архитектурите VAX – инструкциите съдържат режим на адресация (автоинкремент или автодекремент) и адрес на регистър;

Основните начини за адресация в 93% от случаите са непосредствен операнд, регистров операнд и памет с отместване; всички останали начини могат да се изразят чрез тези три в рамките на две инструкции:

чрез инструкцията load Rj = M[Rb + #d] всеки адрес и всяка стойност от паметта може да се запише в регистър;

чрез инструкцията store M[Rb + #d] = Rj стойността на всеки регистър може да се запише в паметта;

останалите сметки могат да се извършат с регистрова адресация;

това е важно за машините RISC, където стремежът е ALU да работи само с регистри;

например ще моделираме индиректната адресация с памет;

load Ri = M[Rj];

load Rk = M[Ri];

бързодействието е почти същото - в оригиналния вид на първия такт междинният адрес се записва в неадресуем междинен регистър и на следващия такт стойността на този регистър се изважда на адресната шина;

7.Инструкции за управление

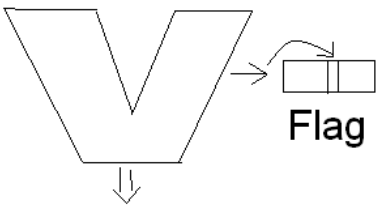
Управляващи инструкции – това са специална група инструкции, които променят съдържанието на програмния брояч, като нарушават концептуалния ред на последователно изпълнение в Нойманова архитектура. инструкциите за управление имат следните аспекти:

- дали има преход или не;
- къде се намира целта на прехода;
- дали се свързва инструкцията с адрес за връщане;
- дали се запазва и възстановява регистровото състояние;

Инструкциите за управление са

- **branch** за условен преход (conditional)
- **jump** за безусловен преход (unconditional);
- за управление са извикване на подпрограми (**call**) и връщане от подпрограми (**return**)
- инструкции за системно извикване и връщане (**system calls, system returns**);

jump винаги променя съдържанието на програмния брояч, докато при **branch** има условие и PC се променя единствено ако това условие е изпълнено. Как ще се вгради условието при **branch** е въпрос на реализация на архитектурата. Условните преходи могат да се направят по такъв начин, че в тях да се изчислява стойността на условието; най-често условието е сравнение – преход се извършва, ако резултатът от сравнението е положителен; така най-често в инструкциите branch се описва какво се сравнява и адресът на прехода; сравнението се изпълнява в ALU; това е единият тип архитектури с условни преходи с инструкции за сравнение; при другия тип архитектури инструкциите за сравнение са отделени от инструкциите за условен преход и те се извикват непосредствено преди това.



Условието често тества флагов регистър.

Признаци на резултата се записват във флаговия регистър. Всяка аритметична операция слага някакви флагове. Те са особени правила, по които се разпознава условието при прехода, в зависимост от състоянието на флаговете.

Недостатък на кодовете за условен преход е, че чрез тях се получава неявна връзка между инструкциите – примерно условен преход задължително трябва да се изпълни след изпълнение на аритметична операция с ALU за числа с фиксирана точка; при pipelining(конвейер) това може да наруши бързодействието – при инструкциите за преход целият конвейер се изчиства;

call съдържа адрес на преход с който се сменя съдържанието на PC и по този начин се отива в друга програма. При фон Нойманова архитектура извиканата подпрограма задължително трябва да се върне в извикващата

трета група инструкции са **system call** и **system return**;

system call означава, че програмата се обръща към някоя особена системна програма, която е различна от обикновените програми; операционната система е съставена от такива системни програми; system call трябва да запази цялото състояние на процесора (при обикновен call се запазва стойността само на общите регистри (AX, BX, CX, DX, SI, DI, BP – Base Pointer);

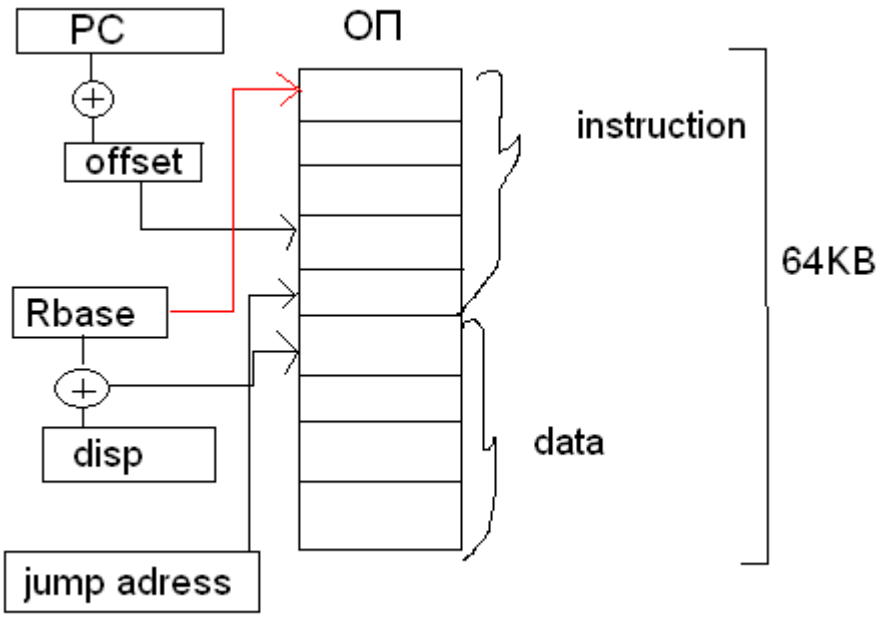
това е необходимо, тъй като при system call се напуска режима на обикновена програма и се влиза в режим на системна програма; при връщане със system return трябва да се възстанови цялото състояние на процесора;

един начин за изпълнение на system call е прекъсване, инициирано от програмата; при него в PC се зарежда адрес на фиксиран вектор на прекъсване – всеки такъв вектор съдържа начален адрес на системна програма; векторите са номерирани и system call с операнд номер на вектор означава, че по този номер се адресира вектора и се прави преход по неговата стойност;

при връщане от прекъсване се възстановява състоянието на целия процесор (включително стойността на стековия указател);

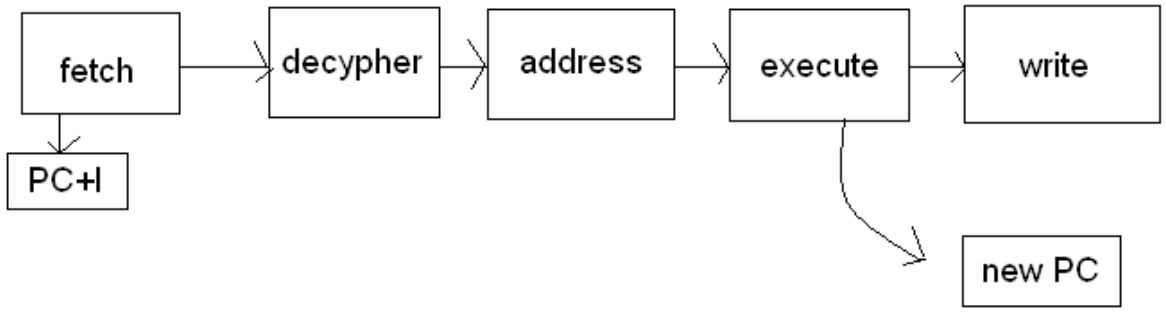
Адресът на прехода се формира по 4 възможни начина.

1) като относителен адрес спрямо програмния брояч(PC+offset)
2) Чрез базов регистър и отместване (Rbase + displacement)
3) Чрез задаване на абсолютен адрес (jump address)
4) Чрез вектори



Формирането на адреса на прехода чрез програмния брояч и определено отместване от него е относителна адресация. В нея няма зададен адрес само отместването спрямо текущия. Това се нарича PC-relative адресация. Според статистика в 94% от случаите отместванията са на не повече от 256 байта спрямо текущата позиция, тоест достатъчно е дължината на полето на offset-а да е 8 бита. Това е най-правилния и систематичен метод(наистина така пише), но недостатъка му е че при фиксирана дължина на преместването не могат да се правят далечни преходи. Това го прави неуниверсален.

При втория случай в базов регистър се държи адреса на началото на 64KB сегмент в паметта а displacement-а определя отместването спрямо този адрес. Това е стандартния 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. Недостатъка на този механизъм е че адреса на прехода не е свързан с текущия адрес и това затруднява конвейера, при достигане на етапа на изпълнение.



В програмния брояч е заредена следващата инструкция и ако по време на екзекуция се достигне до преход това накъсва работата на конвейера.

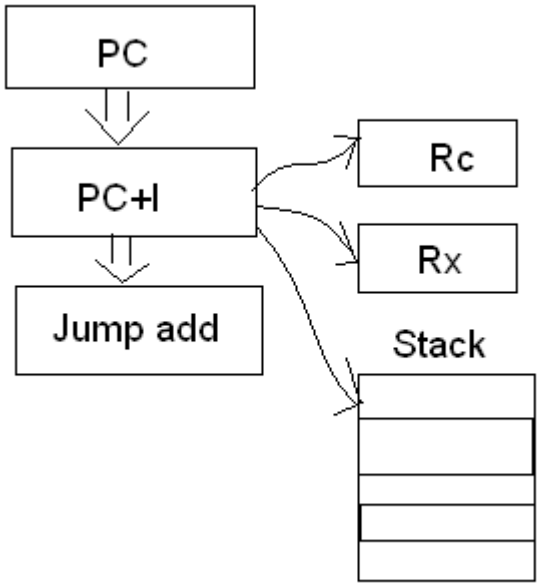
При адресиране от първия тип брояча се обновява директно със offset.

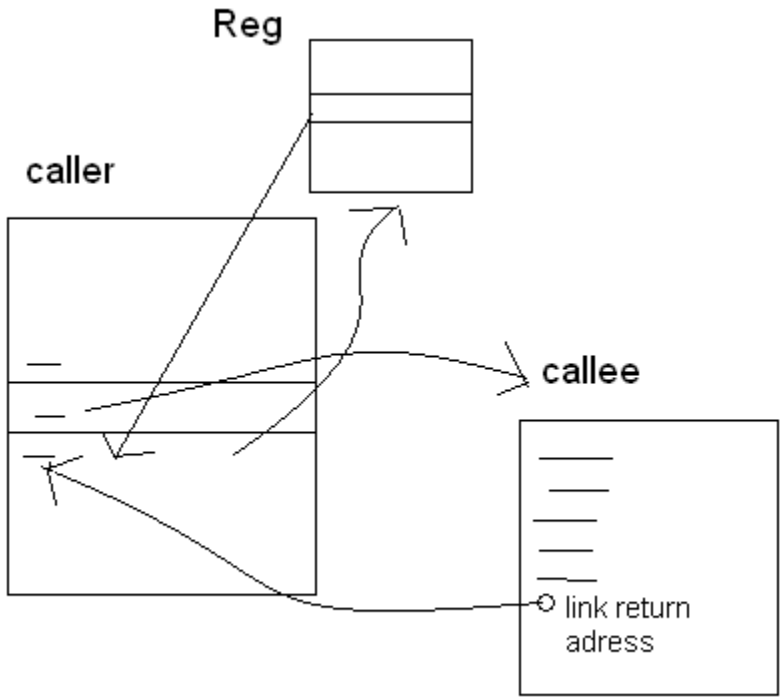
Третия начин е чрез директно указване на адрес на преход или указване на регистър който го съдържа. Този метод е най-бърз тъй като новата стойност директно се записва в програмния брояч, но полето на адреса е по голямо (32 бита) и програмата не е мобилна. Затова този начин не е ефективен (дървен е).

Четвъртия 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 първия чрез jmp x, където дължината на x е 1 байт (x е от 0 до 255). Вектора x сочи в полето на програмата. Частта от ОП която се заема от векторното пространство. (При Intel архитектурата то е от адресите 0x0000000 до 0x00000FF). В тази адресация има неявни база и отместване. Базовия адрес на началото на ВП се пази в специален регистър. ВП 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.

Хардуерната възможност процесора да прави прескачания се нарича система за прекъсвания. Това е модификация на архитектурата на фон Нойман.

Друга особеност на инструкциите за прехода е прехода към подпрограма. За да може да е динамично(извиканата подпрограма да се връща там откъдето е извикана), 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 адреса на връщането. Той може да се окаже експлицитно или да бъде в регистър по премълчаване (имплицитно оказване). При извикването на подпрограмата програмния брояч се обновява с следващата инструкция в паметта, новополучения адрес се запомня във указания регистър Rx или регистър по премълчаване Rc, или в стека. Когато програмиста оказва в кой регистър се запазва адреса на връщане трябва да внимава подпрограмата да не промени стойността на този регистър.

друга възможност е адресът за връщане директно да се записва в стек – тогава при всяко извикване на подпрограма адресът за връщане се вмъква във върха на стека и при връщане от подпрограмата този адрес се изважда от върха на стека и се записва в PC. По този начин се решава проблема със извикването на няколко вложени подпрограми.

при всяко влизане в подпрограма се налага да се запазва състоянието на процесора; при извикване на call се запазват само регистрите, при извикване на system call освен регистрите се запазват указателите към стека и др.;

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 адресът на връщане в един и същи регистър – за това трябва да се внимава при последователни преходи да не се изгубят предишни адреси;

друга възможност е адресът за връщане директно да се записва в стек – тогава при всяко извикване на подпрограма адресът за връщане се вмъква във върха на стека и при връщане от подпрограмата този адрес се изважда от върха на стека и се записва в PC;

софтуерното запазване на състоянието става преди инструкцията call, така че адресът за връщане не е бил запазен в запомненото състояние на регистрите; при последователни извиквания не е проблем новият адрес за връщане да се записва в същия регистър за връзка, поради запазването на състоянието – единственото ограничение е самите подпрограми да не променят регистъра за връзка по време на изпълнението си; в края на всяка подпрограма се извиква инструкция jump с номер регистъра за връзка и по този начин управлението се връща на извикващата програма;

Последния аспект на управляващите инструкции е запазването на регистрите. За да работи по бързо една програма работи със операнди записани в регистрите, и при всеки преход към подпрограма, този контекст се губи и трябва да се възстанови при връщането. Тази програма трябва да има област на запазване, в която временно се записва този контекст. Софтуерната конвенция е извикващата функция да има грижата да запази контекста си, и при връщане от подпрограмата да го възстанови. В мястото за връщане трябва да има инструкция за връщане на контекста, тоест всеки път се прави multiple load и multiple

store, за запазване и зареждане на всички регистри. Има регистър Rsa, в който се пази адреса на тази safe area.

Регигистрите се разбиват на 4 групи – in/out/local/global

in-аргументите които приема една подпрограма

out – резултата от изпълнението (той може да се връща на извикващата програма)

local – локални променливи

global – глобални променливи.

При влизане се пазят out и local, а при излизане in и local. При системните извиквания се пазят всички регистри, а при обикновените само базовите.

Заради конвейера казваме, че условните преходи са „убиец”. Устройството за предварително разпознаване не може да разпознае прехода, поради което инструкциите за условен преход прекъсват конвейера. Това може да се предотврати като се използва

инструкция с предикати

Условно преместване

ако имаме фрагмента if(a>0) c=b*a, това със стандартни инструкции би изглеждало така

#0 Blez r1 (branch Less than ot Equal to Zero)

#1 mul r3,r2,r1

#2

#1 се прескача.

При условната инструкция юе се разкъса конвейера и целия се изчиства за да се почисти само стъпка #1

Ако има инструкция с условно местене

#0 mul r4,r2,r1

#cmovgt r3,r4,r1

conditional move greater than

За да е коректен резултата той не трябва да отиде в r3 ако не е изпълнено условието. В този случай няма нарушение на конвейера. Няма прескачане, записа не е свързан с реда на инструкциите.

При по дребни условия е по-добре да се направят ненужни изчисления отколкото да се изчисти конвейера.

Втория начин е чрез създаване на предикатни дейности. При предиката трябва да се определя процесорна променлива която е самостоятелна и еднобитова. Но те не са адресируеми. Ако има предикати трябва всички инструкции освен от операндите си да зависят и от определен номер предикат, който ако е 1 се изпълнява инструкцията, а ако е 0 не се изпълнява. В зависимост от него се определя дали ще има действие на стъпка изпълнение, но това не нарушава конвейера. Програмирането с предикати е много сложно, трябва те да се set-ват.

#0 sgtzp p1,r1 ;set greater than zero predicat

#1mulp r3,r2,r1,p1

Разлики между RISC и CISC процесори:

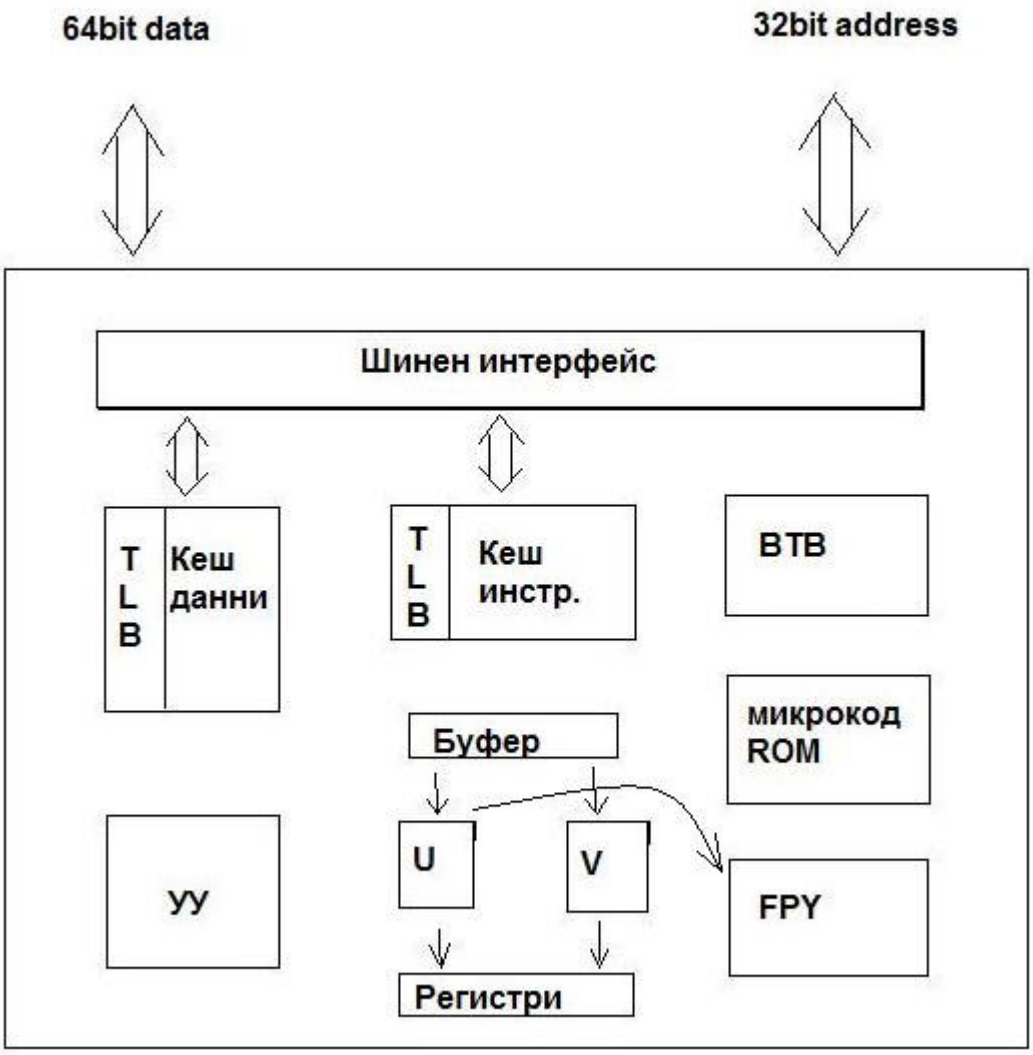
1. При RISC всички операции се извършват за 1 цикъл, при CISC има операции които отнемат няколко цикъла
2. При RISC има само хардуерно управление а при CISC е необходимо и микропроцесорно управление на работата на процесора
3. При RISC четенето и писането става само през регистрите
4. При RISC инструкциите имат точно фиксиран формат
5. RISC има малко режими на работа

Идеята за RISC идва от учени в университета в Бъркли. Те публикуват манифест за това какво представлява RISC-машината. Няма определен манифест , по който да се определя дали една архитектура е CISC, счита се че тя е такава, когато не е RISC. RISC – фактора определя колко пъти са по-бързи RISC машините – между 2.7 и 3.7 пъти.

9.Вътрешна структура на процесор Pentium, блокове и конвейери

Процесорите Pentium са с така наречената X86 архитектура. Тези процесори изглеждат външно като CISC , но са реализирани на базата на RISC-процесор (програмиста може да

подаде на процесора сложна(CISC) инструкция, а той вътрешно я реализира чрез повече по-прости (RISC) инструкции).



6. Шинния интерфейс управлява 64-битова шина за данни и 32 битова адресна шина. Архитектурата x86 е 32 битова, процесора работи само с 32 битови операнди и такива са адресите, но 64 битовата шина ускорява преноса на данни от ОП към процесора.
7. Кешовете за данни и инструкции са по 8 килобайта всеки и те правят по-бърза работата на процесора.
8. TLB(transfer location block) – това е блок към всеки кеш, който преобразува адресите. Той приема ефективен адрес, преобразува го в линеен и след това във физически.
9. BTB(branch trace block (buffers)) – като постъпи в процесора branch инструкция в този блок се записва дали е изпълнен прехода. В специален буфер се записват преходите с техните адреси и дали са изпълнени (<address> taken / not taken <destination address>). Буфера трасира последните 256 branch инструкции и има брояч, който записва колко пъти са и не са изпълнени. Този брояч натрупва данни и прави спекулация. Спрямо статистиката УУ приема инструкцията и залага дали ще има преход или няма да има. Ако не е познал се прекъсва конвейера. Това се прави при допълнителната дешифрация. Нормално влиза branch инструкция и след нея PC ще се обнови със следващата инструкция. Ако все пак по време на изпълнение стане преход се чисти целия конвейер. Статистически има 50% шанс да познае. BTB се опитва да натрупва статистика. В този буфер има място за 256

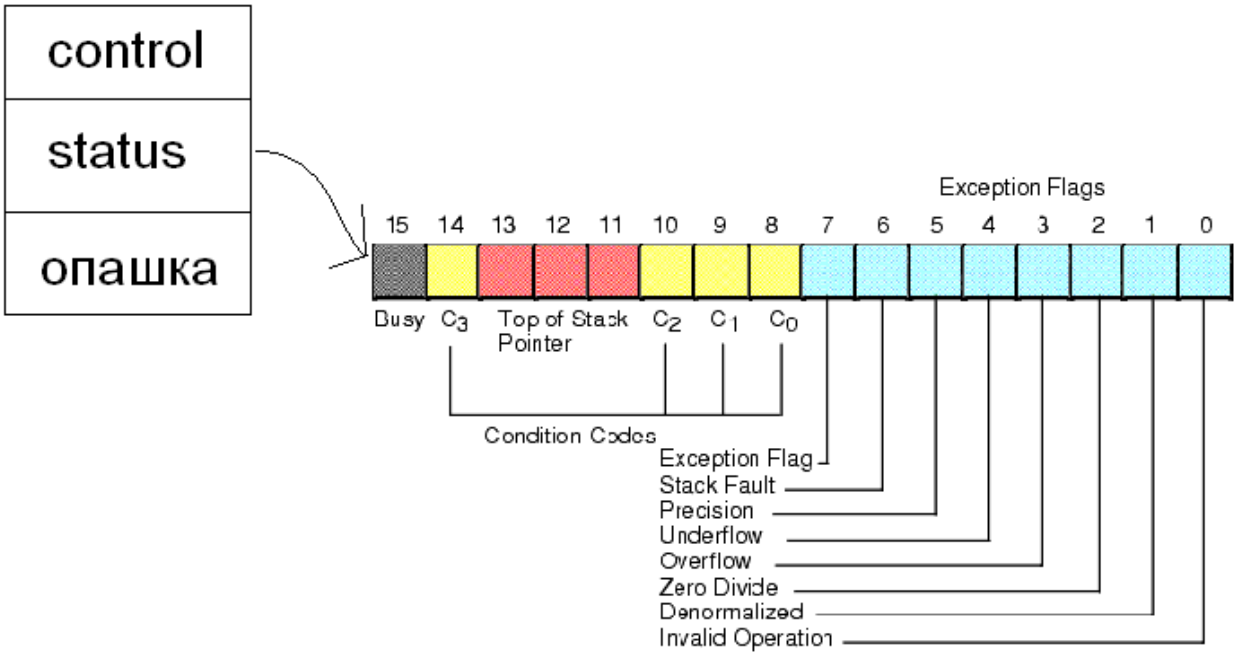
условни прехода (branch-a). Тази схема работи много бързо при цикъл. Ако например правим цикъл на 100 числа BTB ще познае 100 пъти и само на 101-вия ще изчисти конвейера.

10. Буфера за извличане и дешифрация. Буфера трябва предварително да разпознае дали една инструкция за преход е branch или jump. После той декодира инструкциите и ги гледа по двойки. Има 10 правила, по които УУ решава дали може да ги пусне по двата паралелни конвейера U и V. V е за проста аритметика. В U постъпват всички операции работещи с floating point данни. На 4 стъпка той ще ги прехвърли в FPU (floating point unit).
11. Микрокод ROM – това са микропрограми за управление (тъй като това е CISC архитектура). По отношение на изпълнението инструкциите вървят по паралелните конвейери и правят паралелно 1, 2, 3 и 5 стъпка на конвейерната обработка. Следователно това е супер скаларна схема (защото има FPU и 2 конвейера)

- FPU – процесор за аритметика с числа с плаваща запетая
  При архитектурата x86 числата с плаваща запетая са 80 битови, като във процесора за обработката на тези числа има 8 80 битови регистъра

1 b	15b	64 bit		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag
S	EXP	мантиса		tag

както и три 16 битови регистъра



tag означава състояние на регистрите, неговите разновидности са

00 – валидно съдържание на регистъра

01 – стойност 0 в регистъра

10 – празен регистър

11 – NAN – безкрайност, денормализирана стойност или невалиден формат на числото.

Вторият 16 битов status регистър указва състоянието

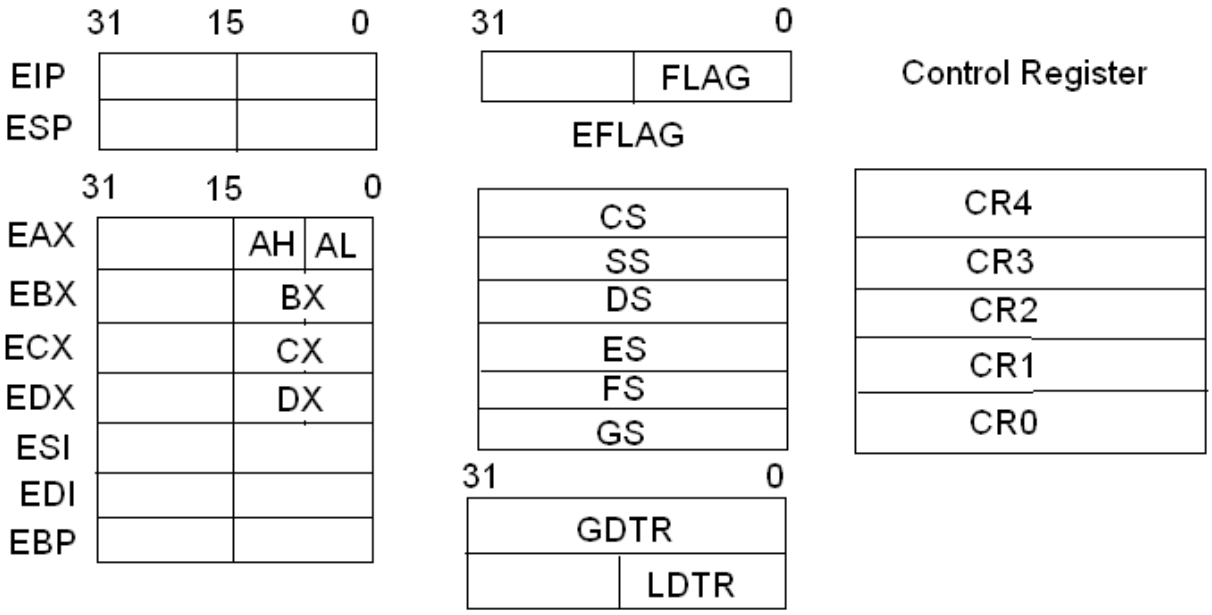
- в полето top се държи указател към върха на стека
- C0 – C3 съхраняват condition code
- PE – precision error – загуба на точност
- UE – underflow error - препълване отдолу
- OE – overflow error– препълване отгоре
- ZE – zero error – деление на 0
- DE – denormalized error – денормализиран операнд
- IE – instruction error – невалидна операция

c3 c2 c1 c0 образуват така наречения condition code

0000 означава валиден положителен денормализиран резултат

всички кодове означават специфични случаи на резултата например + безкрайност (0101)

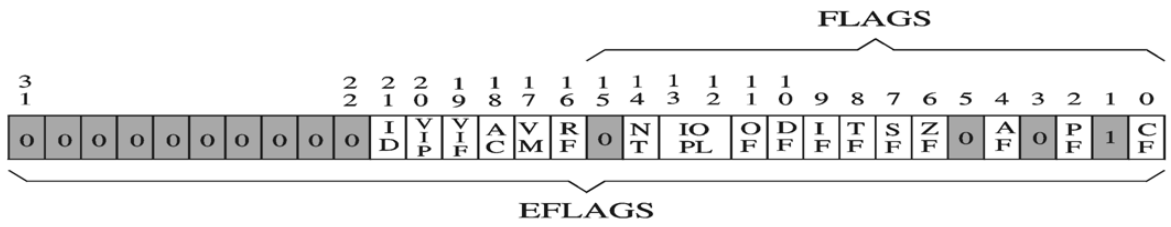
Регистров блок.



Регистрите в X86 архитектурата са 32 битови като първи в паметта е най-старшия (31вия; little endian)

- Буквата E(extended) преди наименованието на регистъра означава, че се работи с целия 32 битов регистър, а когато я няма се работи с младшата 16-битова част. Тя от своя страна може да се раздели на две 8-битови части H(high; старша част), L(low, младша част).
- IP (instruction pointer) – Регистъра, който съдържа програмния брояч //EIP(extended instruction pointer – 32 bit)
- SP/ESP (stack pointer/extended stack pointer) – указател към върха на стека.
- FLAG/EFLAG – флагов регистър. При изпълнение на инструкциите се сформират различни състояния на процесора и те се записват във 32 битовия флагов регистър EFLAG или във младшата му част FLAG.
  При процесорите Pentium флаговия регистър е със следната структура

Flags register



- C - carry flag– пази преноса на резултата от предишна операция навън от разрядната решетка
- P - parity flag– това е флаг за четност, ако резултата има четен брой единици този флаг се вдига. Той не може да се променя в програмна инструкция.
- A -auxiliary flag– флаг за пренос между младшата и старшата тетрада на най младшия байт на резултата. Използва се за десетична корекция.
- Z - zero flag – когато е 1 резултата е 0
- S - sign flag – тук се записва първия бит на резултата(знаковия бит), 1 ако резултата е отрицателен
- T - trace flag- обикновено е 0. Поставя се в 1 ако сме в режим на трасировка със командите SET T, CLEAR T
- IE - interrupt enable – разрешаване на прекъсване, Има 2 вида прекъсвания – маскируеми и немаскируеми . Когато флагът е вдигнат маскируемите прекъсвания са забранени
- D- direction – указва посоката на низовите операции, дали са отпред назад или отзад напред. Ако е 0 указателите в SI и DI се увеличат, иначе се намаляват,поставя се с SETD и CLEARD
- O – overflow – всяка аритметична операция се следи за препълване и за пренос от старшата и младшата част. Когато има препълване O=1.
- IO-PL – input – output priority level. Ниво на приоритета което се изисква. Използва се за контрол от операционната система. Има 4 нива на приоритети, като най-високо (нулево) има операционната система
- NT – nested task – ако е 1 се работи със многозадачна операционна система
- VM - virtual mode – виртуален режим на работа
- AC – alignment check – при 1 процесора проверява дали операндите са подравнени на граница. При 0 не се проверява
- VI – virtual identity
- VIP – virtual identity
- ID – идентификация на процесора. Всеки процесор има идентификатор и програмата може да се заключи за работа само с един процесор при 1

5) Сегментни регистри – има един сегмент за инструкциите (CS-code segment), един за стека(SS-stack segment) и 4 за данни (DS(data segment),ES,FS,GS). Тези регистри са програмно достъпни.
6) GDTR (global description table register) – държи указател към дескрипторните таблици
7) A и B са акумулаторни регистри. AH и AL са по 8 бита и образуват AX(A extended), който е 16 битов, а EAX е целия 32 битов регистър. Аналогично е за регистъра BX
8) C е регистър, в който се държи брояча – използва се за цикли.
9) D(data register) е регистър за данни
10) SI(source index) DI(destination index) са регистри, които държат адресите при низови операции.
11) BP(base pointer)

10.Видове инструкции и изпълнението им в процесор Pentium

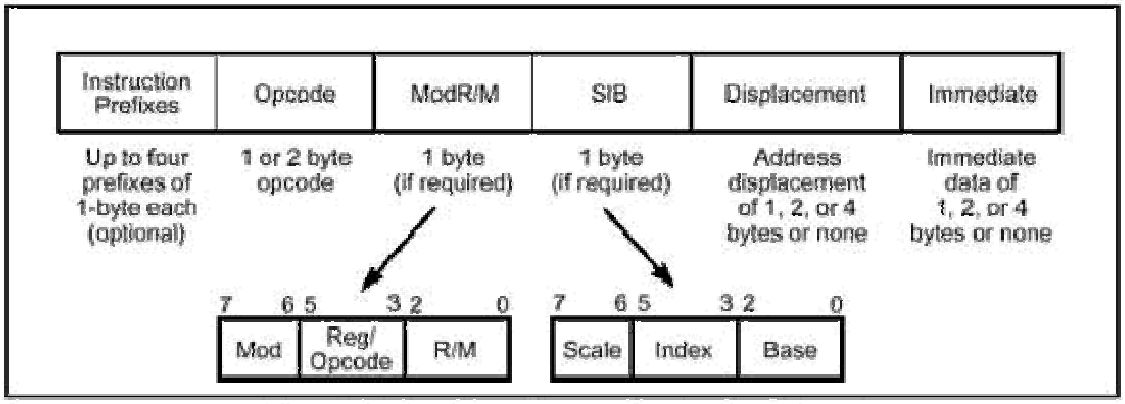


Figure 2-1. Intel Architecture Instruction Format

Формата и полетата на инструкцията се развиват с течение на времето и в днешни дни формата е много усложнена.

Инструкцията може да има префикс(предхождаща част). Възможни са максимално 4 префикса като може и да няма нито един. В един байт отделен за префикса има специфична кодова информация, която е уникална и няма подобна в кода на инструкцията. Префиксът е част от самата инструкция.

REP 0/1	OS 0/1	AS 0/1	SO 0/1

Размер на префикса в байтове

- REP – repeat; повтаряне на следващата инструкция, докато съдържанието на брояча не стане 0 (за това се използва регистъра ECX) като при всяко повторение се намалява съдържанието на брояча с единица. Префиксът се поставя пред инструкцията при писането на програма. Например за инструкцията MOVS (move string) има смисъл да се използва само с префикса repeat. Тъй като самата инструкция MOVS взима от регистъра ESI съдържанието като адрес. Там е източникът на низа, а в EDI е адресът, където се поставя резултата. Низът се мести байт по байт. След прехвърляне на 1 байт от единия низ към другия регистрите се увеличават с 1 (минават на следващия адрес). В брояча C предварително се въвежда размера на низа. Чрез комбинация на инструкцията MOVS, която работи с по 1 байт и инструкцията repeat се извършва обработване на повече байтове. Във флаговия регистър (flag) има клетка D(direction) и когато D = 0 регистрите се увеличават с единица. Когато D = 1 се намаляват.

- Следващият префикс е **OS (operand size).** В кода на операцията е името на инструкцията и всяка операция е за определен брой битове на операнда. След OS операндите се разглеждат като 16 битови при очаквани стандартни 32 битови.

- **AS (address size)** обръщат се адресите на операндите от 32 битови към 16 битови адреси.

- **SO (segment overwrite)** предназначен е за промяна на сегментите (CS< SS< DS< ES< FS< GS). Първоначално инструкцията е за фиксирано използване на сегментите и инструкциите се четат от кодовия сегмент (стандартно инструкциите се четат от кодовия сегмент). Има си стеков сегмент, а операндите са в дата сегмент, който може да се покрие от ES, FS и GS. SO сменя DS, ES, FS, GS.

Т.е. имаме

CS code

SS stack

DS data

ES extra

FS extra

GS extra

Първите 3 сегмента са непресичащи се. Последните 4 се пресичат физически. Чрез segment overwrite може да се променя само сегмента на данните и той да се замени с FS, ES или GS.

Инструкцията има следния вид и възможни размери на отделните части

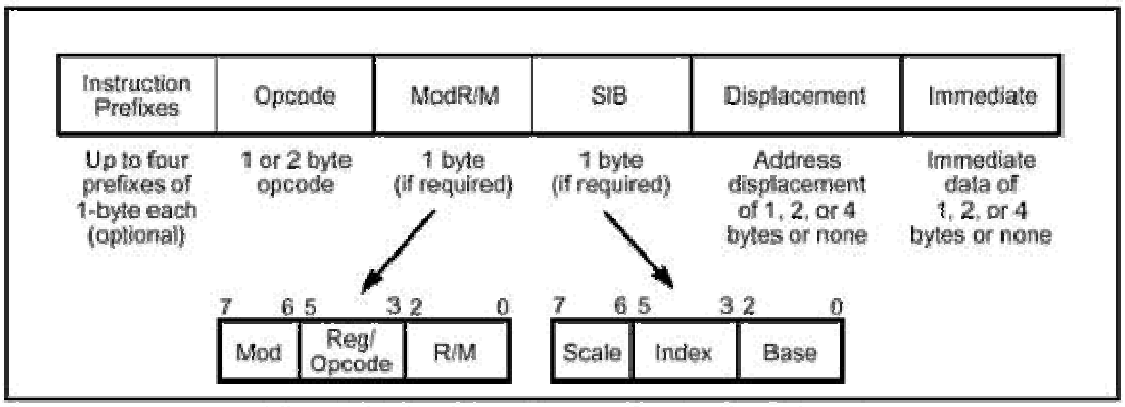
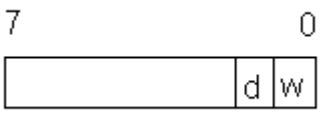


Figure 2-1. Intel Architecture Instruction Format

12. **КОП – код на операцията.** Това е уникален код еднозначно различим от префиксите. Процесора има 256 комбинации за префикси и операции, но инструкциите са повече от 256 , затова КОП може да е 1 или 2 байта, като има и някои 6 битови кодове



w означава дължина на операндите, например ако процесорът очаква 32-битов операнд и в него постъпи инструкция с поле w = 0 , то той се настройва за работа с 16-битов операнд.

означава direction – посока по отношение на прехвърлянето на операндите. При 0 те се прехвърлят от регистъра към паметта, при 1 от паметта към регистрите. Кодът на операцията сам по себе си не носи стойността или адреса на операндите, а само каква е операцията и размера им. Техните адреси се определят в следващите полета на инструкцията.

13. Полето mod r/m е с големина 1 байт и има фиксиран формат състоящ се от 3 части mod, reg и r/m както е указано на схемата. **Първият операнд във инструкцията задължително е в регистър.**

    12) полето mod указва режима, тоест къде е втория операнд. Ако mod =11b, то и двата операнда са в регистрите(като стойност) и техните адреси техните адреси се изчисляват по таблица 1, ако то е различно от 11b, то втория операнд е някъде в паметта.
    - полето reg определя номера на регистър в който е първия операнд (спрямо таблица 1)
    - полето r/m в случая mod = 11 определя номер на регистър, в който е втория операнд, а в другите случаи определя по какъв начин този операнд ще се търси в паметта.(таблица 2)
14. полето sib се използва при базова индексация със скалар на втория операнд, като то се разделя на три части scale, index, base, като чрез него адреса се определя спрямо таблица 3.
15. полето disp указва отместване спрямо базовия регистър.
16. в полето immediate се подава непосредствена стойност на операнд, ако тя присъства в инструкцията

В първия случаи на R-R адресация при mod=11 операндите се намират в следните регистри спрямо таблицата

reg (3b)	r/m(3b)	няма w		има w				
		16	32	16		32		
				w=0	w=1	w=0	w=1	
000	000	AX	EAX	AL	AX	AL	EAX	
001	001	CX	ECX	CL	CX	CL	ECX	
010	010	DX	EDX	DL	DX	DL	EDX	
011	011	BX	EBX	BL	BX	BL	EBX	
100	100	SP	ESP	AH	SP	AH	ESP	
101	101	BP	EBP	CH	BP	CH	EBP	
110	110	SI	ESI	DH	SI	DH	ESI	

111	111	DI	EDI	BH	DI	BH	EDI	

Таблица1

Във всички останали случаи първия операнд е в регистрите и неговата стойност се взима по същия начин, но втория операнд е в паметта, и полето mod r/m показва къде точно е адреса му.

Адрес на втория операнд:

r/m (3b)	mod=00	mod=01	mod=10
000	EAX	EAX + d8	EAX + d32
001	ECX	ECX + d8	ECX + d32
010	EDX	EDX + d8	EDX + d32
011	EBX	EBX + d8	EBX + d32
100			
101	d32	SS[EBP+d8]	SS[EBP+d32]
110	ESI	ESI + d8	ESI + d32
111	EDI	EDI + d8	EDI + d32

Таблица 2

В втората колона sib=0 и disp=0, с изключение на случая 101, където d32 не е регистър а отместване(според нас d8 е когато няма alignment, а d32 – когато има). В случая това означава,че имаме явен адрес. Във всички случаи данните са в data segment, освен при 101, когато са в stack segment. Кодът 100 е запазен, ако той присъства то полето sib става един байт(има го) и адреса се указва чрез базов адрес, индекс и отместване. Тук

s – скаларен множител, който е 1,2,4 или 8 в зависимост от таблицата

i – номер на индексен регистър

b- номер на базов регистър

Адреса се сформира по следния начин

Аоп=Abase + ss*ind

Индексния регистър(регистърът, сочен от полето съдържа число, което означава размера на елементите в масива. Ако не е фиксиран този множител се определя по таблица 4. Адреса в различните случаи се определя по следния начин:

Адрес на втория операнд:

Base (3b)	mod=00	mod=01	mod=10
000	EAX+SS*ind	EAX+SS*ind+d8	EAX+SS*ind+d32
001	ECX+SS*ind	ECX+SS*ind+d8	ECX+SS*ind+d32
010	EDX+SS*ind	EDX+SS*ind+d8	EDX+SS*ind+d32
011	EBX+ss*ind	EBX+ss*ind+d8	EBX+ss*ind+d32
100	SS:[ESP+ss*ind]	SS:[ESP+ss*ind+d8]	SS:[ESP+ss*ind+d32]
101	d32+ss*ind	SS:[EBP+ss*ind+d8]	SS:[EBP+ss*ind+d32]
110	ESI+SS*ind	ESI+SS*ind+d8	ESI+SS*ind+d32
111	EDI+SS*ind	EDI+SS*ind+d8	EDI+SS*ind+d32

таблица 3

index	registry
000	EAX
001	ECX
010	EDX
011	EBX
100	
101	EBP
110	ESI
110	EDI

SS(трябва да е s)	множител
00B	*1
01B	*2
10B	*4
11B	*8

Таблица 4

При инструкциите в Pentium няма адресация тип M-M, само R-M и R-R. Втория операнд може да е в паметта и да се достъпва с косвена адресация, явна адресация, базова адресация с отместване, неявно зададен непосредствен операнд, който се запомня в полето immed.

10. Модели на адресация при процесори Pentium

(темата я няма в лекциите, взета е от Wikipedia, предполагаме, че това се има предвид)

При процесорите Pentium се срещат три основни модела на адресация

Intel има два режима на работа – *Real Mode* и *Protected Mode*.

CR_0 – в най-десен бит: 1 – Protected Mode;

0 – Real Mode.

като Pentium и Itanium CPU работят стандартно при Protected mode, а предните модели работят в Real mode. Pentium CPU позволява да се симулира Real mode. При Protected mode нововъведенията са с цел поддържане на многозадачност, стабилност на системата, защита на паметта, както и хардуерна поддръжка на виртуалната памет.

При Real mode се прави така наречената плоска адресация (flat memory model). Адресите се получават по следния начин: съдържанието на сегментния регистър се измества с 4 позиции наляво и се събира с регистъра на отместване, като това, което се получава, е абсолютният адрес на клетката. Сегментните регистри са винаги 16 бита. В Real mode те са младшата част (първите 16 десни бита). По този начин получаваме 20-битов адрес, тоест максималната адресура в Real mode е 2^{20}B или 1MB. Този 1MB е за управляващи таблици, код на потребителската програма.

Управление на паметта и транслация от логически (ефективен) към физически адрес все още би могла да бъде имплементирана върху модел със плоска памет, чрез него може да се постигне функционалността на операционна система. Но главното предимство на този модел е че цялото адресно пространство е линейно от адрес 0 до адрес MaxBytes-1.

Управление на паметта е възможно, но не се изисква от архитектурата

- В прост контролер, или в *single tasking* програма, където организация на паметта не е нито необходима, нито подходяща, плоския модел е най-удачен защото предоставя най-простия интерфейс от гледна точка на програмиста, със директен достъп до всяко място в паметта и минимална сложност на дизайна.

- В компютърна система със общо предназначение, която изисква многозадачност, алокация на ресурси и зашита, плоския модел трябва да бъде съпътстван със някаква схема за организация на паметта, която обикновенно се имплементира чрез комбинация от специализиран хардуер и софтуер вграден в операционната система. Плоския модел на ниво физическа адресация дава възможност за голяма гъвкавост при прилагането на такъв тип управление на паметта.

Този режим е предвиден с цел програмите, направени за Intel 286, 386 (16бита), да могат да работят директно без тяхното прекомпилиране.

Предимствата му са

- Прост интерфейс за програмисти, чист дизайн
- Предоставя най-голяма гъвкавост
- Максимална скорост на изпълнение

Недостатъка му му е че не е подходящ за многозадачни операционни системи, освен ако не е подобрен със допълнителен хардуер и софтуер за организация на паметта. Това е най-честия случай при модерните CISC процесори – сложна технология на организация на паметта и сигурност върху прост плосък модел на паметта.

Това е при нормален режим на работа Real mode. В защитния режим на работа (Protected Mode) паметта се организира по по-сложен начин.

Сегментацията на паметта е най-използвания метод за да се постигне memory protection. Другия често срещан начин е страницирането. В компютърна система използваща сегментация, операнда на инструкцията, който сочи към място в паметта включва стойност, която идентифицира сегмент и отместване във този сегмент. Сегмента има разрешени нива на достъп и големина асоциирана с него. Ако процесът, който тече има правото да достъпи паметта, която се опитва да достъпи, и отместването е в рамките на сегмета, адресирането е разрешено, иначе се вдига хардуерно изключение.

В допълнение към нивата на достъп и границата, сегмента също носи и информация относно къде е той в паметта. Той също така има флаг който указва дали сегмента го има или го няма в главната памет. Ако го няма, ще се хвърли изключение и операционната система трябва да прочете сегмента от вторичен носител.

Информацията относно локацията на сегмента може да сочи към първата клетка в сегмента ако е в режим на сегментна преадресация, а може и да сочи страница за сегмента. В първия случай, отместването ще бъде прибавено към адреса на първата клетака и това ще даде адреса на търсения операнд, във втория случай, отместването се привежда до адрес в паметта използвайки каталог на страниците. На процесора се указва дали работи в режим на страниране във флаг в регистъра CR0

Характерно за страницирането е че:

- Подходящо е за мултитаскинг, дизайн на операционни системи, защита на ресурсите и алокация
- Подходящо е за имплементация на виртуална памет
- Малко по -бавна скорост при изпълнение
- По-сложно за програмиране
- Непроменяеми граници на страниците, не винаги е ефективно относно памет

а при сегментирането

- Близко до страницираната памет, но сегментирането се имплементира със имплицитното събиране на два регистъра сегмент:изместване
- Променливи граници на сегментите, по-ефикасно и гъвкаво от модела със страницириране
- Доста сложно и нетривиално от гледна точка на програмиста
- По-сложно за компилатори
- Сегментите могат да се припокриват
- Много комбинации сегмент:отместване сочат един и същи физически адрес
- По-голям шанс за грешки при програмиране
- Имплементирано в оригиналните Intel 8086, 8088, 80186, 80286 и поддържано от 80386 и всички следващи x86 останал и до днешните Pentium and Core 2 процесори.

11.Сегментна адресация в PENTIUM

В машините PENTIUM са вградени два механизма за организация на виртуалната памет – сегментация и страниране (**segmentation** и **paging**);

В така наречения защитен режим (**protected je**) могат да се използват и двата механизма при адресация.

При сегментацията адресът се получава от 16-битов сегментен регистър и 32-битово отместване;

сегментните регистри в PENTIUM са следните:

17. **CS** – кодов сегментен регистър – показва началото на сегмента на кода на програмата;
18. **SS** – стеков сегментен регистър – показва началото на сегмента на стека;
19. **DS**, **ES**, **FS**, **GS** – сегментни регистри за данни - показват началото на четири сегментна за данни;

регистърът **EIP** е 32-битов и в него се записва отместването относно началото на кодовия сегмент.

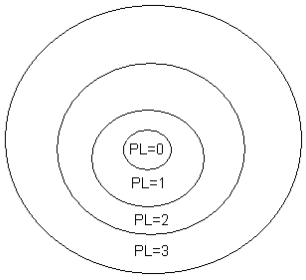
регистърът **ESP** е 32-битов и в него се записва адресът на върха на стека, зададен като отместване относно началото на стековия сегмент;

всеки от останалите 32-битови общи регистри може да се използва за задаване на отместване при адресация на данни в някой от сегментите за данни.

в защитен режим сегментните регистри се наричат **селектори**, поради специалната роля която изпълняват;

Шестнайсете бита на всеки селектор се интерпретират по следния начин:

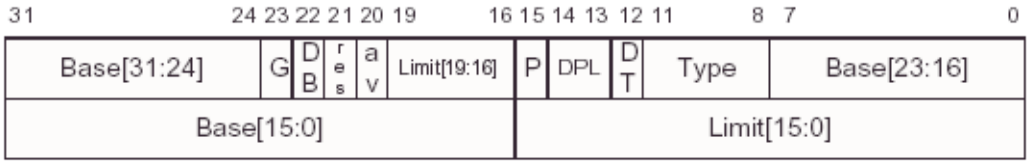
20. индексът показва отместване в **глобална (GDT) или локална () дескрипторна таблица**;
21. ако бита TI е 0, селекторът е за поле в глобалната дескрипторна таблица, ако е 1, селекторът е за поле в локалната дескрипторна таблица;
22. битовете RPL (Requested Privilege Level) са битове за нивото на привилегии на текущата програма. Нивата на привилегии са 4 като най високото е 0. Това е нивото на привилегия на операционната система.



RPL

Дескрипторните таблици съдържат **сегментни дескриптори**, като всеки сегментен дескриптор описва един сегмент. сегментите дескриптори са по 8 байта, разпределени по следния начин:

(на долния ред са младшите 4 байта, на горния ред са старшите 4 байта)



23. полето Base задава адреса на началото на сегмента. То е разделено на 3 части
24. полето Limit задава размерът на сегмента; то е 20 бита, така че максималният размер на сегмент е 1000000 единици;

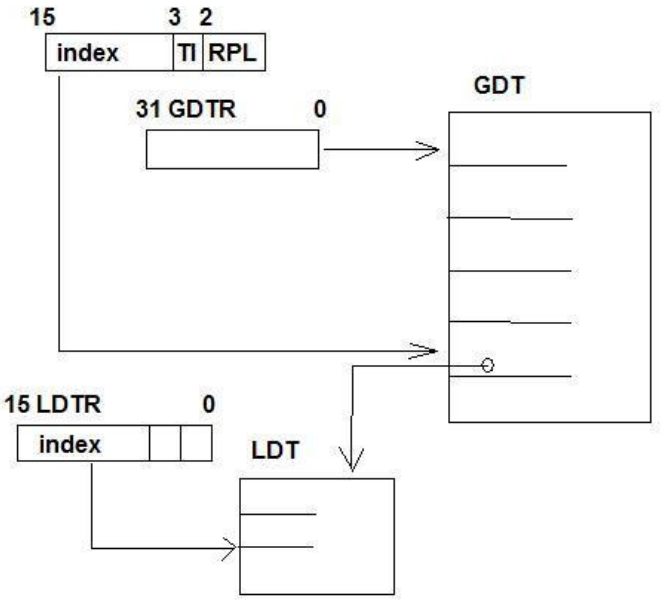
25. ако битът G за гранулярност е 0, единиците в сегмента са байтове, така че един сегмент е до 1 MB, ако битът G е 1, единиците в сегмента са страници по 4KB, така че при G = 1 един сегмент може да достигне 4GB;
26. битът DB задава размерът на операндите и адресите за този сегмент – 16 бита, ако DB = 0 или 32 бита, ако DB = 1;
27. битът P е бит за наличност (present) – P = 1, ако сегментът е наличен в оперативната памет, P = 0 в противен случай;
28. битовете DPL са битове за нивото на привилегии на дескриптора;
29. битът DT определя типа на дескриптора – DT = 0 за системен сегмент, DT = 1 за сегмент за приложна програма;

при адресиране се използват два типа дескрипторни таблици:

30. глобална дескрипторна таблица GDT, която е единствена и се използва най-вече за дескриптори на системни сегменти; адресът на началото на GDT се записва в регистър **GDTR**, който се инициализира от ядрото на операционната система при нейното стартиране;
31. локална дескрипторна таблица LDT, която не е единствена и се използва най-вече за дескриптори на приложни сегменти; в даден момент може да се използва точно една локална дескрипторна таблица – нейният начален адрес е поместен в дескриптор на GDT, индексът на който се намира в регистър **LDTR**;

коя дескрипторна таблица ще се използва – глобалната или текущата локална зависи от бита TI на съответния сегментен селектор – ако TI = 0 се използва GDT, ако TI = 1 се използва LDT;

тъй като размерът на индекса в селекторите е 13 бита, то една дескрипторна таблица може да съдържа най-много 8192 дескриптора;



получаване на адреса при сегментацията става по следния начин:

(зададени са селектор и отместване)

32. по бита TI се определя в коя дескрипторна таблица да се търси сегментния дескриптор;
    1. ако TI = 0, таблицата е GDT – по регистъра GDTR и индекса в селектора се определя сегментният дескриптор в GDT;
    2. ако TI = 1, таблицата е LDT – по регистъра LDTR и регистъра GDTR се определя дескриптор в GDT, от който се извлича адресът на началото на LDT и след това по индекса в селектора се определя сегментният дескриптор в LDT;
33. след като е определен сегментният дескриптор, се прави проверка дали сегментът е достъпен за програмата, дали се намира в оперативната памет, дали отместването е в

рамките на сегмента; ако това е изпълнено, взима се адресът Base на началото на сегмента и към него се прибавя отместването;

за по-голяма ефективност на тази адресация, към всеки селектор се залепя 64-битов невидим регистър, който съдържа дескрипторът на сегмента, който последно е бил адресиран; по този начин, ако непосредствено след това отново се адресира този сегмент, информацията се изважда директно от невидимия регистър, а не от съответната дескрипторна таблица;

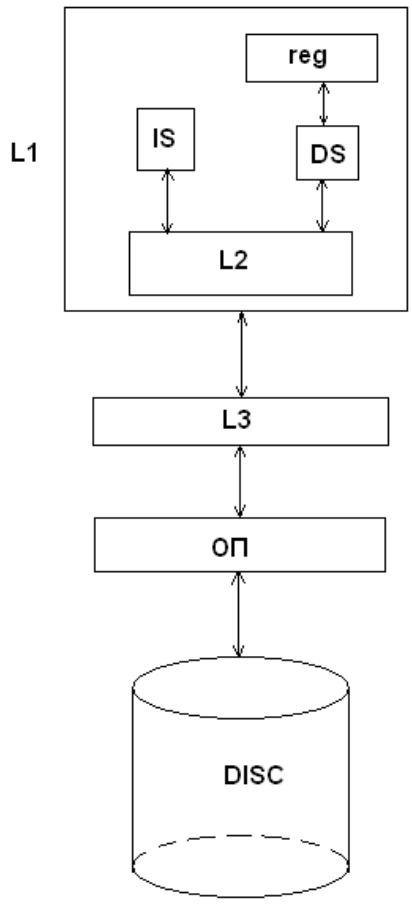
полученият адрес след сегментацията се нарича **линеен адрес** и той е част от **линейното адресно пространство**;

линейното адресно пространство е 4GB; оттук са възможни два случая:

34. линейното адресно пространство директно се изобразява върху физическото адресно пространство;
35. линейното адресно пространство е виртуално – извършва се странична преадресация; страниците имат размер 4KB;

кой от двата случая е налице зависи от бит, разположен в контролния регистър **CR0** – този бит указва дали се използва страниране или не;

12. Организация на кеш-памет. SRAM памет. Йерархия на кеш-паметта



Структура и организация на паметта: адресно пространство, като адресите (на данните и инструкциите) се формират от процесора. Паметта е била хомогенна линейна последователност от адреси. С развитието на технологиите се откриват недостатъци - забавяне поради време за достъп (време за достъп - от задаването на адреса до попадането в клетката, която се задава с този адрес). Това време се намалява непрекъснато, но намалява и такта на процесора. Следователно отношението на двете времена се запазва и то е в рамките на около 50 (1 наносекунда за такт на процесора и 50 наносекунди за време за достъп)

За премахване на този проблем се преминава към архитектурни подходи - изграждане на йерархия в паметта. Така вече има бърза и нормална памет (cache и оперативна памет).

ОП – основната памет. Тя трябва да съхранява цялото оригинално линейно пространство.

кеш – съхранява само малка част дублирани адреси от адресното пространство.

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)

L1 – най-малка като обем и най-бърза

L2 - малко по-голяма и по–бавна

L3 – най-голяма и най-бавна

И трите са много по-бързи за достъп от ОП и трите съдържат само дубликати. Ако един адрес е представен в L1, то той е представен по същия начин в L2 и L3 (съответно в ОП), но ако е в L3, например, то не е задължително той да е представен в L1 и L2.

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)(при Intel – 8 KB за данни и 8 KB за инструкции). Когато L2 е вътре в процесора (заедно с L1), се повишава скоростта на работа. Идеята е скоростта на работа с данните между регистрите и L1 да е почти същата като тази на процесора (с % по-малка). Преди освен с бързината е имало проблем и с обема на паметта – не е стигала. Например в IBM 360 (адресната решетка е била 24 бита, тоест адресируеми са били 16MB. На практика обаче размерът на реалната памет е бил е1MB. Затова е започнала да се използва виртуална памет, като паметта 1MB е в ОП, а останалите се записват на диск. Там са всички байтове и при 32 разрядна решетка там има 4GB. Реалната оперативна памет днес се прави така, че да достига 4 GB. Преди виртуалното пространство е било около 10 пъти повече от ОП. Сега се различават до един-два пъти. В ОП реално има дубликати на нещата от диска .

Нека X е структура в йерархията. Тогава означенията са:

X block – X на нивата L1, L2, L3 и в ОП – минималното количество памет което може да се използва , обикновено за кешовете е 16Bytes

X hit – когато имаме адресация и адресираният блок се намира в X, имаме попадение

X miss – блокът, който е адресиран, го няма в X

X hit time – времето, за което имаме достъп до X при hit

X miss ratio – отношението на непопаденията – опитите за достъп до X при които не се попада на адреса, тоест X MISS RATIO = сполучливи/несполучливи.

X miss penalty - блокът го няма в X и той трябва да се докара от най-близкото място и да се откара после в процесора. (това е времето за справяне със ситуацията, когато адресът го няма в X). Например, ако го няма в L1, го търси в L2, ако го няма там, го търси в L3, ако не – в ОП и после го слага нагоре по йерархията в процесора (L3, L2, L1, процесора).

Tavg = Thit + %miss*Tmiss

Tavg – средно време за достъп

%miss – процент на пропускания

Tmiss – време за донасяне от друго място

Thit – време за взимане когато е намерен

Ако Тхит =1 , Tmiss = 20 , %miss = 5% => средното време ще е равно на 2.

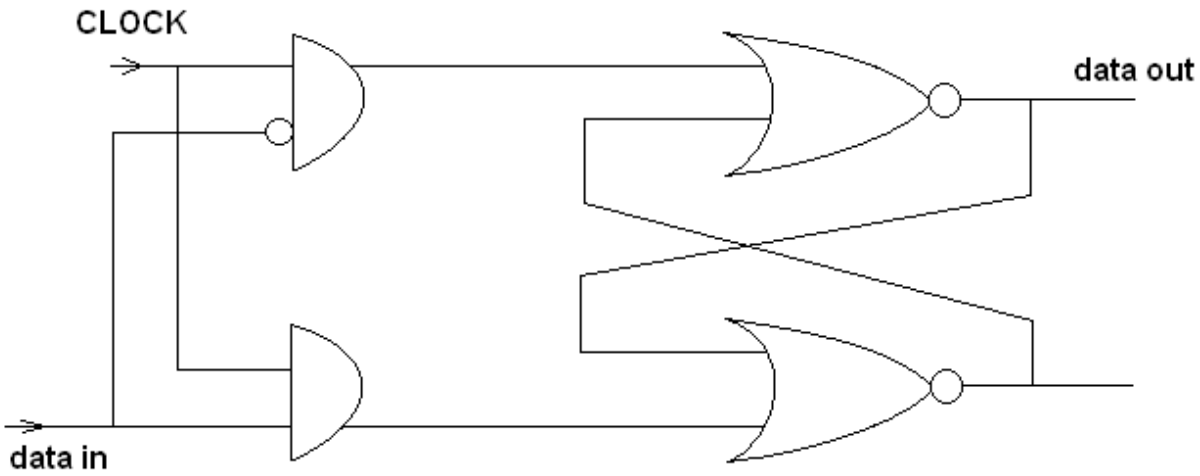
Ефективността на кеша зависи от процента на miss-а. Кеширането се оказва ефективно, благодарение на особеностите на Нойманoвата архитектура и програмите които се пишат. Инструкциите които се изпълняват от процесора са подредени линейно или на групи в линейни

участъци, освен това скоковете при преходите, имат стремеж да не са много далечни, а в сравнително близки участъци. Така се получава времева локалност – напоследък използвани адреси трябват отново в близко време. (Те са по-често използвани в кратък период от време). Пространствена локалност – съседните адреси на използвания в момента адрес е много вероятно да бъдат скоро използвани. Тоест ако ни е нужен един адрес, то заедно с този адрес докарваме в кеша и съседите му, тъй като най-вероятно те ще потрябват. Тоест в L1 се докарват не само 4 байта за адрес, а цял пакет от 16 байта. Теглят се повече адреси и освен това, когато се сложи някакъв блок в кеша, той най –вероятно често ще се използва => програмите са локални по отношение на времевото и пространственото използване на адреси. Адресите които са активни (използват се в момента) се движат на групи, това води до сравнително добри характеристики на miss ratio. Регистровата памет е най-бърза.

Reg	< 2 KB	1 ns	150 GB/s
L1	< 64 KB	4 ns	50 GB/s
L2	< 8 MB	10 ns	25 GB/s
L3	< 64 MB	20 ns	10 GB/s
ОП	< 4 GB	50 ns	4 GB/s
Disc	>1GB	10 000 000 ns	10 MB/s

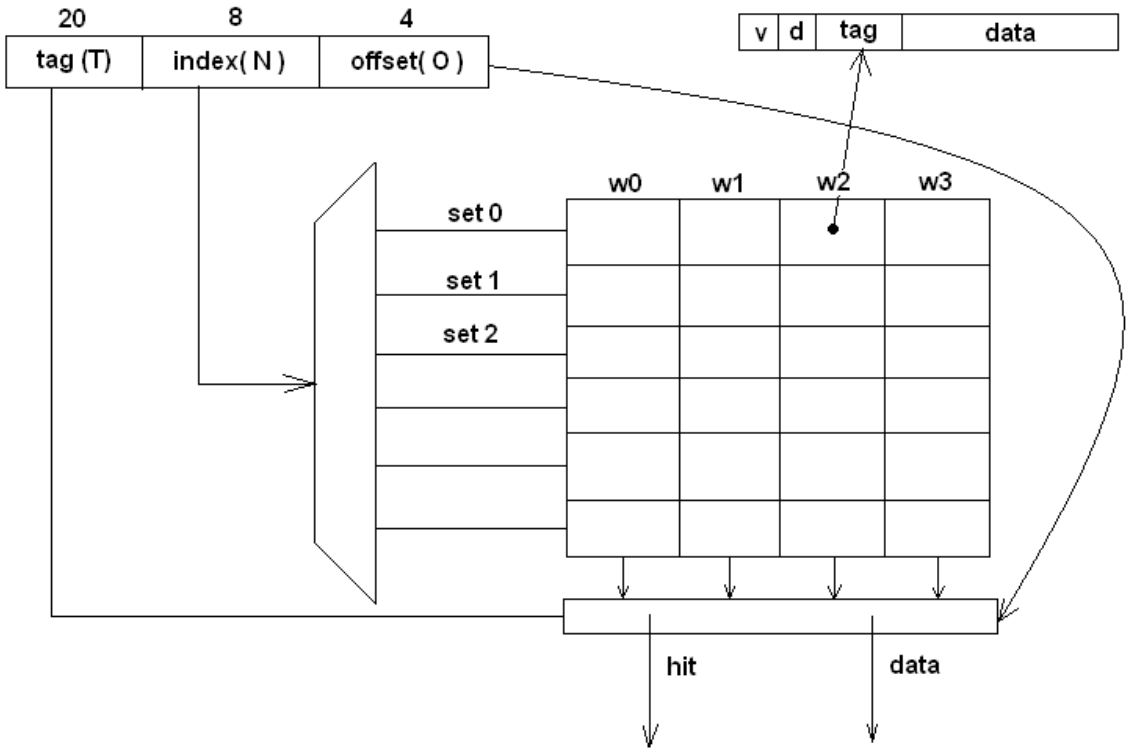
Целта на кеша е да е бърза => не може да се управлява от програма или ОС, трябва да е хардуерно управлявана. Кеша не се командва от ОС.

Технологично има 2 типа памет – статична и динамична (SRAM и DRAM). Биполярни транзистори (по-бързи), се използват за SRAM , отделят повече топлина и са необходими 6 транзистора за 1 бит информация.



Информацията в тригера се записва само по време на единичната стойност на clock-а (клокиран тригер) и се пази неопределено дълго време, не се нуждае от освежаване. Cache паметите се правят на базата на такива тригери, които обаче са големи (като обем), затова са с по-малка плътност. По тази причина cache паметите са с малък капацитет.

ABC = Associativity - Block-Capacity



Структура на cache – сх.2 – съставена от множества и във всяко множество има еднакво количество frame-ове(n на брой множества). Данните във frame-а са един блок и frame-а съдържа tag – идентификатор на блока (име) и 2 бита за състоянието на block-а -> valid(v) и dirty(d).

valid = 1=>frame съдържа нещо, ако е 0 – няма нищо. Какво точно се съдържа се определя от tag и data. В началото, когато се зарежда нова стойност във frame-а d е 0. След първото писане на каквото и да е в блока на frame-а d става 1. Когато се подаде адрес към кеш паметта, той се разглежда като съставен от 3 части – offset – отместване спрямо началото на блока, определя размера на блок (колко байта е блокът в тази кеш памет). Ако блокът е 16 Bytes, то offset-ът има размер 4 bits. index – свързване с номер на множеството. При конструирането на cache-а първо се определя колко ще са множествата (винаги степен на двойката - например 256 8 бита поле index). Остават 20 b за tag (tag присъства във frame-а). В зависимост от приетата асоциативност се определя бройката (плътността) вътре в множеството. При 4 frame-а във всяко множество има четирипътна асоциативност. При подаване на адрес автоматично се определя index , активира се 1 от 256 множества. Фреймовете се подават на асоциативното устройство и в него се търси едновременно match-ване в tag-а от адреса. Ако някой tag съвпадне с този от адреса hit става 1 и се подават на изхода всички фреймове (още не се знае кой е номерът на фрейма). При cache паметта не трябва да има последователно проверяване за match на tag-а, защото

това забавя. Едновременно се пускат всички tag-ове за проверка. Ако имаме match излиза hit = 1 и самите данни от блока на frame-а, но не знаем кой frame ни е свършил работа. Тази схема не дава номера на фрейма вътре в множеството. Това върши работа при четене, но при писане ще се изисква да претърсим всички фреймове в множеството, последователно да ги обходим и да работим с този, който ни трябва. Чрез асоциативната схема бързо вадим, каквото ни трябва, но нямаме позициониране.

Имаме 2 крайни случая при реализация на кеш памет: директно изобразяване (direct-mapped) – тук нямаме асоциативност; всяко множество се състои само от 1 frame. Тогава по index директно се отива на множество и се проверява дали tag-ът е същият. Друг граничен случай е set-associative. При него полето index = 0 като дължина, т.е. кеш паметта има само 1 множество и в него са всички фреймове (в случая 1024). tag-ът нараства и става 28 бита, докато при direct-mapped полето за tag намалява по размер. L1 cache в Pentium е двупътно асоциативен, 8KB капацитет, 16 Bytes block => по 2 frame-а в множество, 256 множества, index – 8 bits, offset – 4 bits, tag – 20 bits. В L2 cache – 64 KB капацитет, 32 B block, двупътна асоциативност (2 frames в множество) => 1024 множества => 5b offset, 10b index, 17 bits tag (или може 256 множества при 4 frames в множество – 4-пътна асоциативност).

Има 4 аспекта, по които се определя как работи един кеш:

- Къде може да бъде поставен 1 блок (пласиране на блок) – при напълно асоциативен кеш се слага в един от малкото фреймове в множество (в кой да е фрейм). При direct-mapped в единствения frame на множество
- Идентификацията на блока – първо се намира множеството, на което е блокът, в паралелно търсене се търси съвпадение на tag-а и се взима по offset-а конкретната стойност на блока
- Replacement – замяна на блокове във frame-овете. Frame-овете се пълнят бързо и когато вече няма място, а трябва да се постави нов => replacement. Първо трябва да се избере в кой фрейм на дадено множество ще слагаме блока. Това става като се използва ефективен алгоритъм по времева локалност (Least Recently Used = LRU). Изхвърля се най-рядко използваният напоследък блок и се слага на негово място новият => всяка операция на четене, запис, трябва да се брои, като тези броячи се нулират периодично. По тях се определя кой да се изхвърли. Това е най-добрият алгоритъм, но тук говорим за hardware-на реализация, това изисква много хардуерни ресурси => ще е тежък. Затова се използва и random –изхвърля се случаен (това не е хубав начин, но е прост и не утежнява допълнително работата). Затова в cache-а се прави подобрение на random, което е not most recently used (NMRU) – следи се най-използваният напоследък блок и се гарантира, че няма да се изхвърли той, а върху останалите се генерира случайно число, което определя кой да се махне.
- Стратегията на запис (write strategy) – когато се получи резултат, той да се запише до последното ниво на паметта (в L1,L2,L3,ОП,ДИСК) – нарича се write-through – това намалява производителността на кеш-а т.к. се отнема повече време са се копират данните навсякъде. Но се поддържа консистентност на адресите. Другата стратегия е

write-back – процесорът пише само в едно ниво и край. Така може да се получи, че един tag в L1 не отговаря на L2,L3,… т.е. това не е консистентност на данните, но е по-бързо.

При replacement – ако d = 0 => блокът е идентичен на оригинала, т.е. не е писано и можем направо да запишем върху него. Ако d = 1, значи той е променян и трябва да го запишем първо на долното/долните нива и после да пишем върху него. Ако се работи често с една данна, тя се променя в кеша (например L1) и когато вече не е нужна се записва надолу по йерархията (когато тръгнем да махаме блока) – това се прави и write-back. По-бързо е отколкото да се сменя 10 пъти навсякъде, но се губи консистентност.

Локация – какво се прави, когато искаме да запишем, а блокът го няма в L1? Опитваме се да запишем в несъществуващ в кеша блок => има 2 стратегии – write-allocate (върви в комбинация с write-back) – ако го няма в блока, го докарваме в кеша и правим запис по адреса, който трябва да се запише; блокът остава в кеша; no-write-allocate – блокът го няма в кеша и не се докарва – направо се отива в паметта и се пише там (разумно е да се ползва с write-through). Нормално кеша работи с write-back и write-allocate.

ABC:

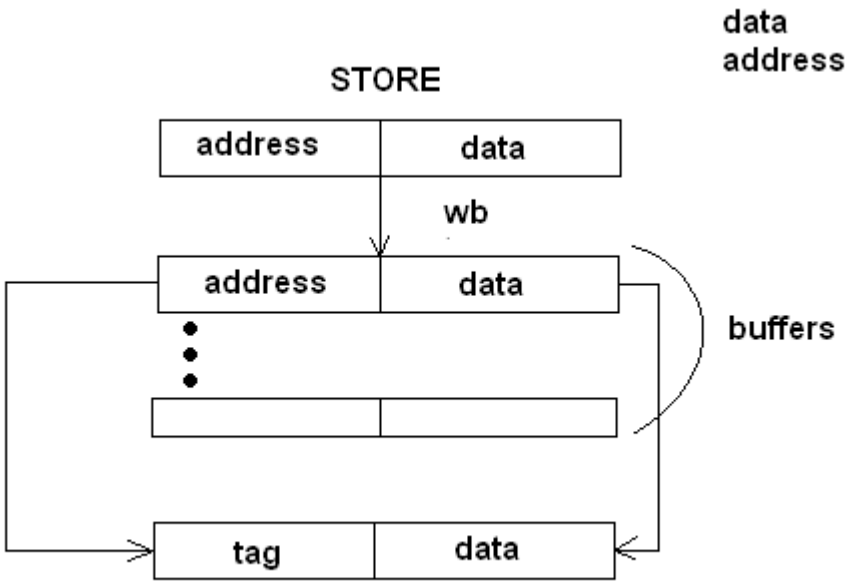
A – Associativity (Асоциативност) – 1,2,4,5,8,16 – по колко frame-а в множество; по-голяма асоциативност – по-нисък miss rate, но кешът поскъпва.

B – Block (Блок) – 16,32,64 при кеша се работи с целия блок, независимо каква част от него ни трябва, затова не е нито много голям, нито много малък.

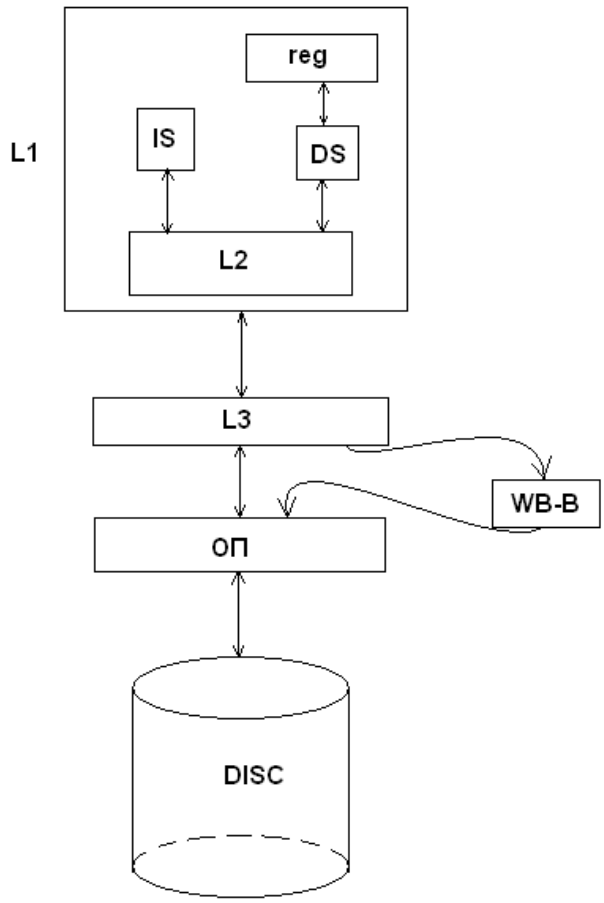
C – Capacity (Капацитет) – ако е много малък – голям miss rate, ако е много голям – бавен => затова са на нива

POLICY	HIT/MISS	to
back + alloc	Both	Cache
back + noalloc	Hit	Cache
back + noalloc	Miss	Mem
thru + alloc	Both	Both
thru + noalloc	Hit	Both
thru + noalloc	Miss	Mem

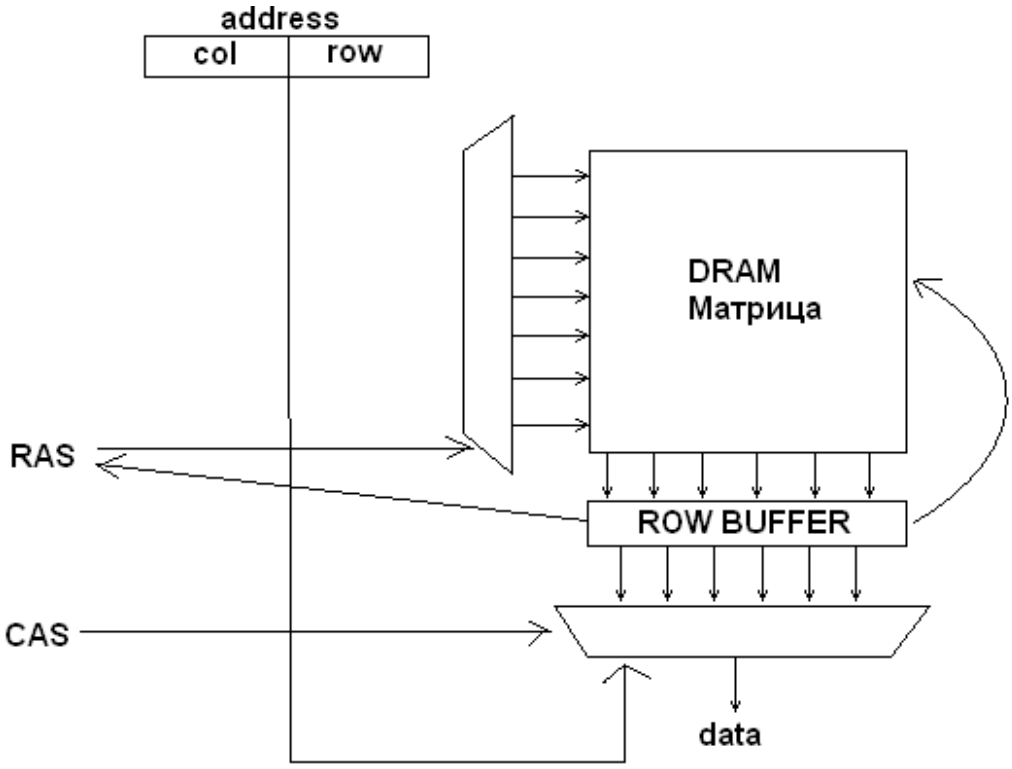
13. Кеш памет. Начини за повишаване на производителността.

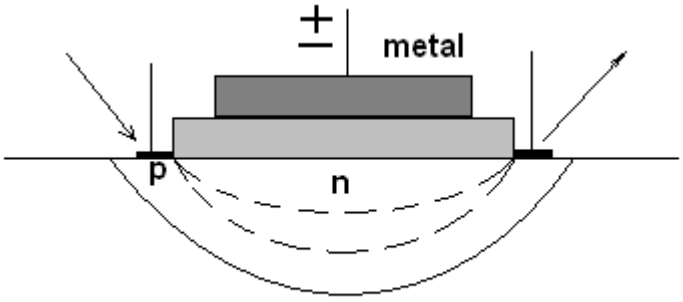


На входа на кеша има Write Buffer. Записът в кеша е няколко пъти по-бавен от четене. Възможно е още преди буферът да се изпразни в кеша, процесорът да е готов с нови данни за запис. В този случай ще има изчакване или объркване. За да се избегне това се работи с повее от 1 buffer, но не повече от 4 буфера. Защо не повече от 4?. Може инструкцията да даде нещо за запис и след малко време да иска този адрес и на него във frame-овете още да не е обновена информацията (и да трябва да се обхождат и буферите). Средно е изчислено, че всяка 4-та инструкция увеличава шанса да искаме да четем нещо, което сме записали до 3 инструкции преди това в буфера. При Replacement, когато искаме да махнем блок от frame с dirty = 1 => блокът трябва да бъде записан в паметта. Слага се write-back buffer (WB-B): първо блокът се вкарва в WB-B, чете се новият блок и след това старият блок се праща в паметта. Така новият блок ще е готов по-бързо в Cache. Такъв буфер WB-B има не само на L3, а на всяко ниво на кеша. Нужен е само 1 такъв буфер на всяко ниво. Използването на кеш много увеличава производителността – особено при двуядрените процесори, които постоянно бълват адреси.



14.Оперативна памет. DRAM – принцип и особености. Interleaving





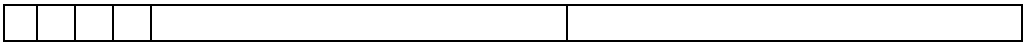
RAM = Random Access Memory – основната памет, която се адресира. Основната памет е голяма и по размер се доближава до виртуалното адресно пространство. Cache паметта трябва да е много бърза, затова се прави от статична рам памет и за всеки бит са нужни по 6 биполярни транзистора. Това отнема много място и отделя голямо количество топлина. За основната памет се използва DRAM (Dynamic RAM), базирана на друг принцип (MOS = Metal Oxide Semiconductor). Силицият е полупроводник, а SO2 е изолаторна пластина и върху нея е поставена метална плочка. Между металната плочка и силиция се получава кондензатор. В зависимост от посоката, в която е зареден n-островчето слиза надолу или се качва нагоре. В зависимост от това дали каналът е отворен може или не може да се стигне от единия gate до другия. Така се пази 1 бит (1 или 0). Технически проблем е, че състоянието се пази от заряда на кондензатора, който е много малък, поради което се саморазрежда и критичното време за разреждането му е 2ms, т.е. помни само 2 ms. Затова на всеки 2 милисекунди кондензаторът трябва да се презарежда – така нареченият refresh на динамичната памет (DRAM) . Преминаването на ток между двата gate-а е деструктивно – разваля съдържанието, затова при

четене съдържанието трябва да се възстанови. Въпреки това този вид памет се използва, защото е много икономично минимален ток => няма греене и се постига огромна плътност (bit/mm2), което компенсира недостатъка с refresh-a. Поради наличието на капацитет на кондензатора четенето и писането е по-бавно от това в cache паметта, която е съставена от транзистори, пък били те и 6.

Intel първи разработват тези полупроводникови RAM памети. Заводите са много скъпи – един такъв струва няколко милиарда долара. Самите елементи се произвеждат в бели стаи. Плътността на полупроводниковите елементи е огромна – няколко милиона на cm2.

DRAM чиповете се основават на квадратна матрица. Адресът е разделен на 2 части. Първа влиза младшата част.

CS



CS – Chip Select – посочва кой от чиповете съхранява даден адрес. Има от 4 до 8 чипа в една конструкция. Подава се част от адреса и само 1 чип я взима. в DRAM матрицата битовете не се групират в байтове, а в редове. Частта от адреса, която отговаря за реда се подава на дешифратор, селектира се една изходна линия (RAS – Raw Address Strobe). По този Strobe дешифраторът активира матрицата – пуска се четящ ток по всички елементи на реда, прочита се целия ред и се записва в буфера на реда. Пуска се сигнал, че буферът е записан и Управляващото устройство в чипа пуска сигнал CAS = Column Address Strobe, който дешифрира частта от адреса, която отговаря за колоната и се достига конкретен адрес (пак се минава през дешифратора). Тъй като четенето е деструктивно, след края на операцията информацията в row buffer-a се записва отново на съответния ред. Refresh-a върви постоянно – на всеки 2 ms прочита даден ред и го записва отново. Тъй като при този вид памет адресът е разделен на 2 части, като първо се чете редът, а после колоната, тя е асинхронна. Чака се да се върне отговор от RAS, за да се изпълни CAS. Може да се изчисли за колко време RAS ще върне отговор и да се направи синхронно – с фиксирано разстояние между RAS и CAS (SDRAM = Synchronous DRAM , DDR = Double SDRAM). Към 2000 година: 64MB на матрица, 50 ns – Access Time, 100 ns per cycle. DDR е два пъти по-бърза, защото работи по-нагъсто, но е по-чувствителна към излизане от синхронизация. Обикновено от паметта се четат 4 последователни байта (от row buffer-ът излизат 4 байта, които започват от адрес кратен на 4). За първият цикъл (това са цикли на паметта – не на процесора!!!), който е 50ns се подава първата част на адреса, за вторият цикъл – втората, през третия цикъл данните излизат навън и в 4-тия цикъл се възстановява извлеченият ред. Всичко това е 200ns. 4 последователни думи ще се прочетат за 16 такта (думите са през 200 ns).

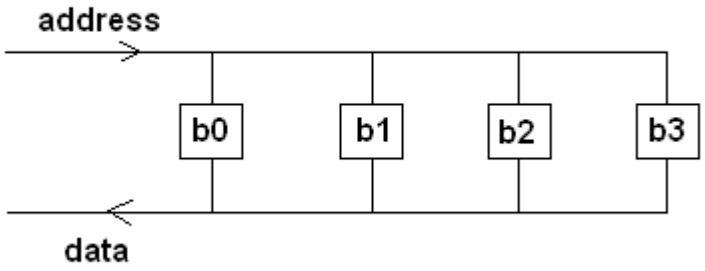
Simple Interleaving:

Cycle	Address	Memory
1	12	A
2		A
3		T
4		B
5	13	A
6		A

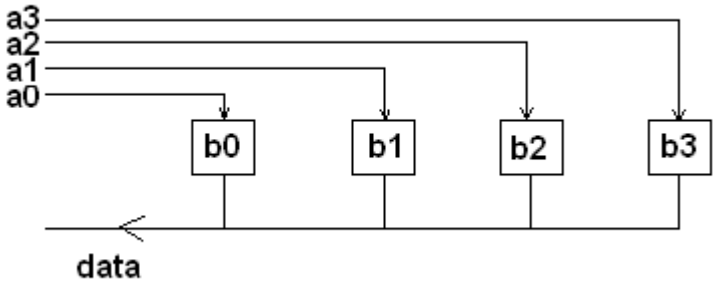
7		T
8		B
9	14	A
10		A
11		T
12		B
13	15	A
14		A
15		T
16		B

На таблицата е показано извличането на 4 думи от паметта. Очевидно този метод не е достатъчно ефективен. За да се увеличи скоростта се създават така наречените банки памет – подреждат се няколко чипа памет (банки) и се използва interleaving – припокриване на работата на 4 чипа. Съществуват няколко вида Interleaving:

- Simple Interleaving (прост) – нареждаме банките на една обща шина на адресите. Като се подаде адрес A – на първата банка влиза A, на втората A+1, на третата – A + 2 и на 4-тата A + 3. По този начин от един адрес започваме да работим по 4 думи. Нулевият адрес се подава на b0, първият на b1, вторият на b2 и третият на b3. По този начин тактовете buffer и transfer могат да се сменят и по схемата отдолу – на 3-ти такт излиза съдържанието на b0, на 4-ти такт – на b1,на 5-ти - b2 и на 6-ти - b3 и те се пускат по шината за output данните. Така четем за 6 такта а не за 16, което е сериозна оптимизация. Освен това нямаме увеличение на адресните шини линии (шините). Ако отгоре имаме L3 Cache simple interleaving-а е много удобен, защото наведнъж обработва 16 байта.



- Complex Interleaving – при този вид имаме 1 шина за данни и 4 отделни шини за адрес. На 4-те банки се подават 4 произволни адреса, които постъпват такт след такт и излизат на output-а на 3-ти, 4-ти,5 -ти и 6-ти такт съответно b0,b1,b2,b3. Тук свършва за 7 такта, защото няма пренареждане на тактове – смяна на T и B.



Complex Interleaving:

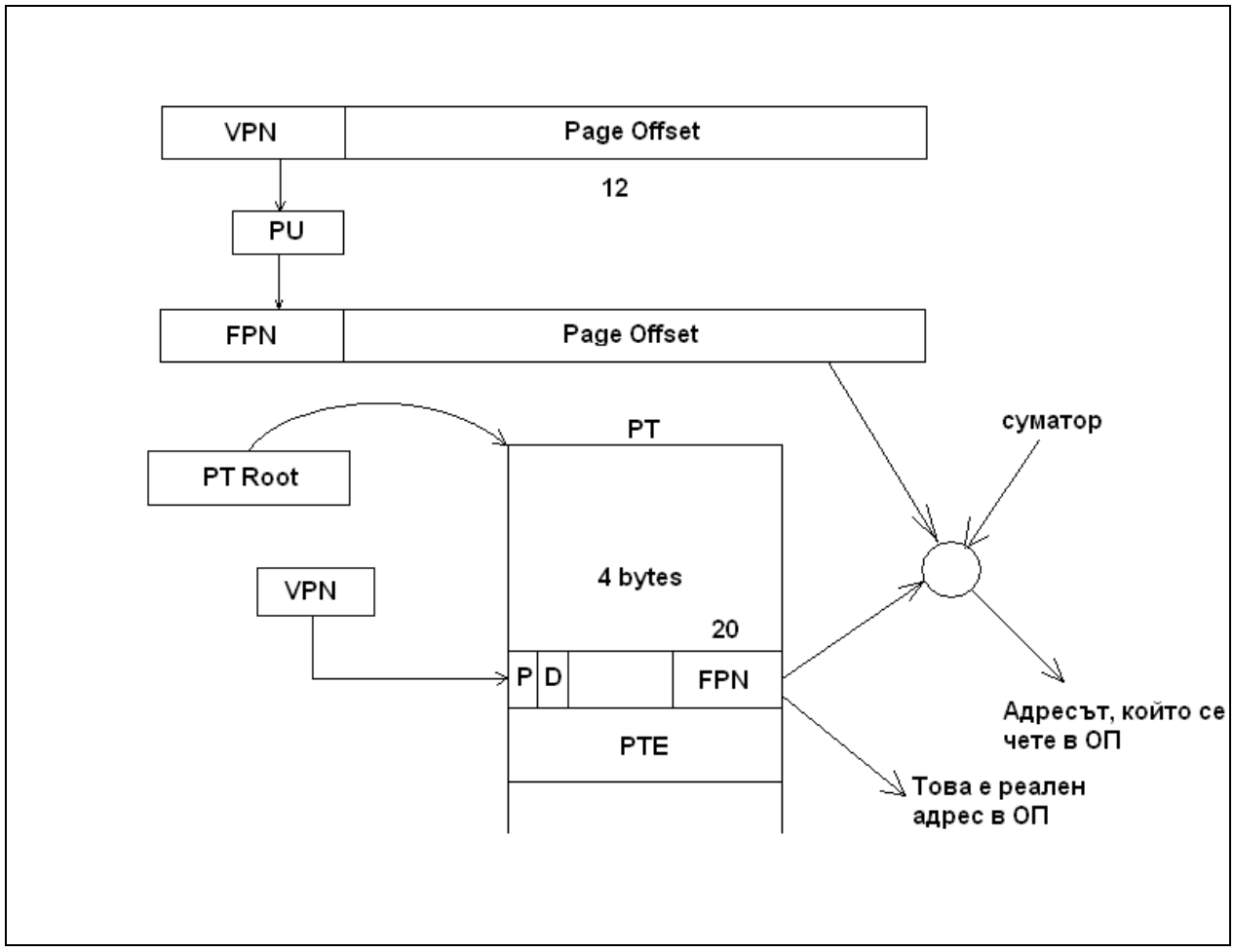
		b0	b1	b2	b3
1	12	A			
2	18	A	A		
3	24	T	A	A	
4	32	B	T	A	A
5			B	T	A
6				B	T
7					B

В тази таблица адресите 12, 18, 24 и 32 са абсолютно случайни и не зависят един от друг. В таблицата на простия interleaving адресът 12 е също случайно избран, но адресите след него задължително са следващите 3 адреса. При наличие на L3 cache над паметта е по-удобно да се ползва simple interleaving.

При синхронната памет е възможно следното усложнение – вместо 1 да се сложат от 4 до 16 row buffera. Пуска се адрес – вади се в първия буфер, пуска се втори адрес – вади се във втория буфер и така нататък. Когато се вади втори адрес се пуска четене за колона на първи буфер и т.н. По този начин адресите се подават като опашка към буферите и постепенно се изтеглят към изхода. Паметта с повече буфери е разработена от Intel и се нарича RAMBUS. За всеки адрес в опашката се проверява дали няма да му трябва ред от матрицата, който го няма там (вече е използван и е в някой друг буфер). С тази доста скъпа памет се достига 5GB/s скорост на обмен.

15. Виртуална памет. Странична организация на паметта. Реализация и механизми. Стратегии при замяна на страниците

Виртуалната памет е множество отделни блокове от последователни адреси (страници). 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. Но в наши дни реалната (основната) памет може да е почти колкото виртуалната. Така реалното адресно пространство може да е колкото реалното. Въпреки това разбиването на страници позволява да имаме схема на транслация на адреса.



Адресът се състои от 2 части – Virtual Page Number и Page Offset – отместване в границите на страницата (ако страницата е 4 KB offset-а е 12 bit). Чрез Paging Unit-а VPN-а се преобразува в Page/Physical Frame Format (номер на реален frame – както в cache frame-ове)(и обратно). Това преобразуване се поддържа от архитектурата и чрез него преминаваме от виртуален в реален номер на страница и съответно от виртуален в реален адрес. В наши дни 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 – например всеки процес да работи от адрес 0 с 32-битово поле и при смяна на процес да се презареждат нови адреси (да се сменя съдържанието на ОП). Ако процесите работят в отделни процесни пространства, то всеки процес ще се shift-ва спрямо началото (това не е добре). 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.

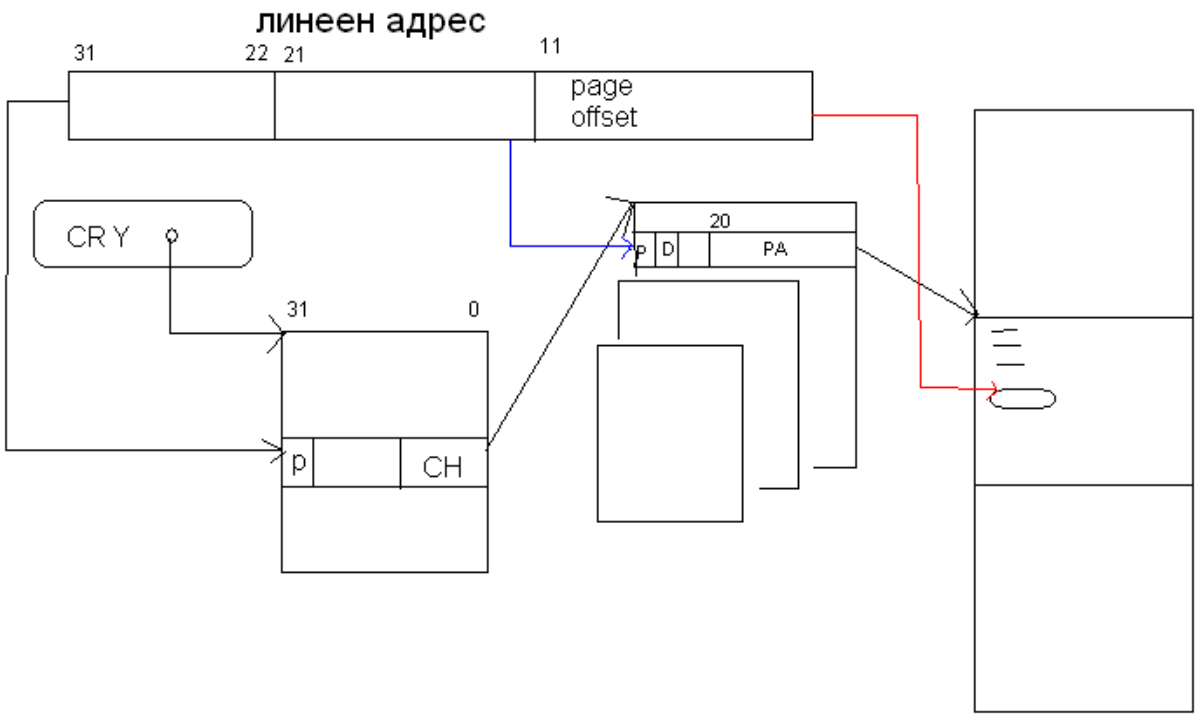
Page Table-а се състои от редове – всеки ред от таблицата съответства на номер на виртуална страница. Класическият размер е 32 бита за ред(при 32-битова машина), което прави 4MB

таблица - таблицата е 1 000 000 реда, което са 1M виртуални страници и всеки ред е по 4 bytes (20 bit за VPN). Тези 4 MB трябва да се неподвижни. Всеки ред (entry) се състои от:

- Present bit – показва дали е представена физическа страница, която съответства на виртуалната, ако този бит е 1 – има, ако е 0 – няма
- В случай, че има – трябва да има адрес на фрейма, адресът е малък, защото е кратен на 4K => адресът на frame-a е 20 bit (всеки адрес на frame завършва на 12 нули . Ако P е 0 процесорът не се занимава и предава управлението на OS, която трябва да разположи тази страница в реалната памет. Тя търси свободно място в паметта и ако намери – слага тази страница на съответното свободно място в ОП и записва 20-битовия номер на реалния фрейм в FPN и слага P=1 за „намерена страница”, връща се от прекъсването, процесорът отново започва да работи и търси страницата, като прави нова адресация. Разликата е, че този път той открива страницата.
- В свободното място между P I 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).
- Когато в ОП няма място, операционната система взема решение коя страница да се изхвърли. Това се определя по Least Recently Used Algorithm (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 е 1 – записва на диска, иначе – не.

Ако адресите са 64 bit, а не 32 bit, тогава остават 52 bits за номера на page-a и table-ът става огромен => този механизъм е неприложим за 64 bit (приложим е максимално за 32 бита).

16. Странична преадресация в Pentium



Pentium е пример за двуетапна адресна транслация. Ако в CR0 регистъра битът paging е set- нат, значи освен сегментация се използва и странициране (segmentation and paging). Линейният адрес се дели на 3 части: CN (Catalog Number), PN (Page Number), Page Offset. В паметта имаме записан 4KB каталог(каталог на каталозите), в който са записани адресите на останалите 1024 каталога в паметта. Той се състои от 1024 реда съдържащи линейни адреси. Десетте бита в адреса, означени с CN определят с кой ред (entry) на каталога на каталозите е свързан търсеният адрес. Адресът на началото на каталогът на каталозите се записва в регистъра CR3. Описаните каталози в него се намират на произволно място в паметта и всеки от тях е по 4KB, т.е. описват се 2 на степен 20 страници. Всяка страница е по 4 KB, което прави точно 2 на степен 32 (колкото е цялото адресно пространство в 32-битова машина).

В режим на сегментация и странициране получаваме 32-битов линеен адрес. Първите 10 бита от него (CN = Catalog Number – номерът на реда в каталога на каталозите) се умножават по ширината на полето на каталога (4 bytes) и се добавят към началния адрес на каталога (записан в CR3). Ако старшият бит на entry-то (P) е 1 (следователно present), значи че съществува каталог в паметта, който се сочи от това entry. В такъв случай се взимат младшите 20 бита от entry-то, добавят им се 12 нули отдясно (началният адрес на страница е кратен на 4096) и се прибавя стойността на полето PN, умножена по широчината на entry-то (4 bytes). В това entry (в младшите му 20 бита) е записан адресът на търсената от нас страница, ако тя съществува (ако P = 1). Последните 20 бита на entry-то съдържат линейния адрес на страницата, а физическият адрес на нейното начало се получава, като отново към тези 20 бита прибавим 12 нули отдясно и се прибави 12-битовият offset. Това е реалният адрес в паметта.

Този метод използва 4KB повече пространство от обикновеното странициране (4KB + 4 MB).

Предимството на виртуалната адресация пред сегментацията във връзка с процесите (отделните големи програми) е, че смяна на процесите означава само смяна на entry-та в PT

Root, т.е. ако се сменя адресното пространство всичко стои постоянно в паметта и всички програми работят във виртуални адресни пространства. Предимството на двуетапната транслация на Pentium пред обикновеното странициране е, че смяната на всеки процес води до смяна на entry в главния каталог (който е само 4 KB). Работи се с активни динамични блокове по 4 KB (малките каталози). Недостатъкът е двойното адресиране. Това адресиране се прави от DLB – блок във cache-а. Тези адреси не се смятат от АЛУ-то а от специални малки суматори.

17. Прекъсвания

Развитието на компютърните архитектури е създало проблеми при една строго последователна програма - контрол на входа и изхода и външни асинхронни събития или ситуации, които налагат една програма да бъде активна по време на друга програма. Оттам възникнала и програмата за обслужване на прекъсванията. Тази програма идва и заменя програмата след осъществено прекъсване и започва да работи с процесора. Важно е да се възстанови прекъснатата програма от същото място все едно нищо не е станато.

Архитектурата трябва да е направена така, че основната програма винаги да може да се прекъсне, да се извика подходяща програма за прекъсване и прекъснятата програма после да продължи все едно нищо не се е случило, тоест последователния алгоритъм се изпълнява строго последователно и прекъсването е напълно прозрачно.

В началото програмирането е само процедурно. То съответства на архитектурата. Но тъй като се оказва, че програмирането е най-скъпия процес се разработват нови подходи като обектно-ориентираното програмиране. То е външен слой – всяка обектна програма се преобразува в процедурна. Прекъсванията не нарушават последователноста. Процеса на изчисление е последователен. В процесорите всеки процес се развива в определен конекст – временно състояние на всички програмно достъпни регистри на процесора. Когато един процес се прекъсне е много важно да се съхрани контекста му в момента на прекъсването и след това той да се възстанови. Системата за прекъсвания трябва да се грижи за съхранение и възстановяване на контекста.

Относно контекста има различни обеми

- минимален обем на контекста – съдържа регистъра EFLAGS и регистъра IP – те трябва задължително да се запазят и възстановят
- всички общи регистри – до някаква степен е полузадължително тяхното възстановяване.

Друг важен аспект на прекъсванията е източника на прекъсването – кой източник го предизвиква. Има следните 5 източника

- прекъсване по машинна грешка – то е предизвикано от схемитеза контрол на централния процесор (хардуера) на компютъра. При правенето на определена архитектура извън основните функционали елементи се вкарват допълнителни хардуерни елементи – схеми за контрол, които следят дали основните елементи работят правилно.

- схема за контрол на работата на АЛУ – АЛУ се прави двойно, при събиране двете АЛУ-та смятат право и обратно събиране и се проверява дали сумата по модул 2 е 0.
- Всички линии за данни имат контролен бит, върху 8 линии деветата е контролна по четност. Ако всички единици в основните линии са четен брой то контролния бит е единица.
- Контролът по нечетност е на обратния принцип.

Има схеми които следят дали не става грешка при всичките процеси в компютъра и ако такава грешка изникне се получава прекъсване поради машинна грашка. Основната програма моментално се прекъсва. Това е най-привилигированото прекъсване. При overclock може схемите на контрол да не хванат машинната грешка. При персоналния компютър няма много схеми за контрол. Има външни схеми за контрол. Те пускат заявка за прекъсване по линията NMI – non maskable intrerrupt - на външни на централния процесор устройства

- входно-изходно прекъсване – когато сме задали нещо и то е завършило задачата си се обажда със входно-изходно прекъсване че е приключило задачата. Източника на това прекъсване е входно-изходния контролер.
- програмно прекъсване – това е когато се изпълни програма в централния процесор и в него настъпи особена ситуация (получава се деление на нула) – се пуска програмно прекъсване. Процесорът не знае и какво да прави и когато работи с операнди на изравнена граница и получи операнд на нечетен адрес. И в този случай се извиква програмно прекъсване.
  Има маскируеми програмни прекъсвания – класически пример е препълването. То може да се маскира. Програмата ще продължи на своя отговорност нататък. Ако не е маскирано при препълване програмата ще се прекъсне.

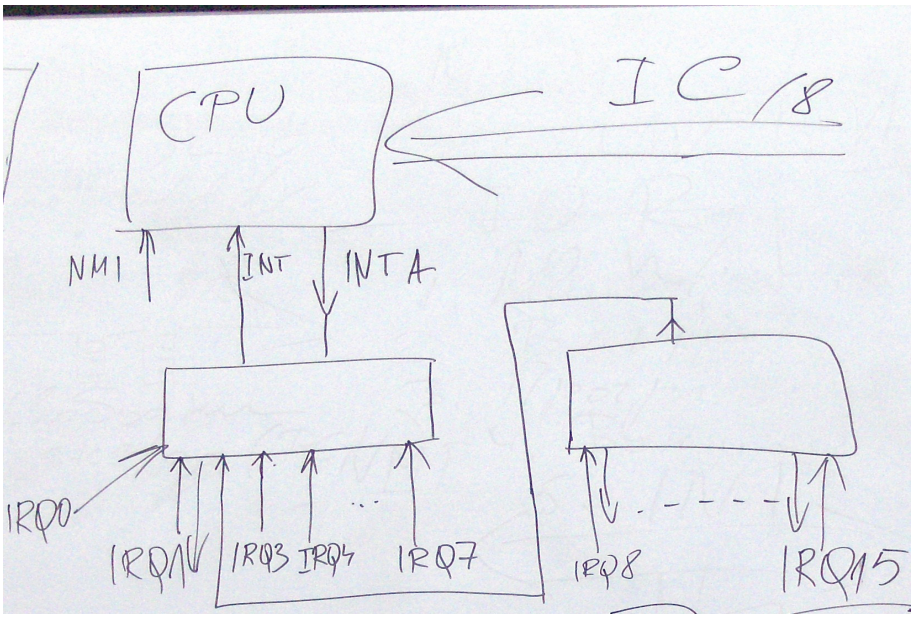
- Външно прекъсване - то не е свързвано с работата на процесора и се постига чрез бутонче reset. То е свързано с тотално изчистване на програмата. Машините VAX , след такова прекъсване се обръщат към конзолата и после основната работа продължава
- INT – искане на прекъсване с интрукция INT. Самата инструкция генерира прекъсване. Тя е последна по приоритет.

Това подреждане съответсва на приоритета на видовете прекъсвания. Когато се случи прекъсване идва програма, която го обслужва. Може да се прекъсне прекъсване от такова с по-висок приоритет. Могат да се вложат няколко прекъсвания. Прекъсването по машинна грешка блокира всяка схема за прекъсване включително и от друга машинна грешка.

Сигналите за машинните прекъсвания са с различни източници, но се събират на едно и също място. Входно – изходните прекъсвания са от различни външни устройсва. Поради естеството на източниците не е нормално всички входно-изходни прекъсвания да постъпват. Някои устройсва правят прекъсване и докато то се обслужи нищо не може да се прави. Не би трябвало друго входно-изходно прекъсване да го прекъсне. Трябва да има система за контрол. Винаги може да се прекъсне от машинна грешка следователно

трябва да има система за маскиране и приоритет. В архитектурата Intel има бит IE – interrupt enable – в регистъра EFLAGS. Ако входно – изходното прекъсване постъпва по линията INT – interrupt и IE бита е 1 то централния процесор възприема да прави входно-изходно прекъсване. Ако IE =0 то прекъсването е маскирано и не се допукса, тъй като сигналът не знае каква е маската. Процесорът, ако максата му позволява, връща INT A – interrupt acknowledge. Централният процесор има една линия за вход и една за изход за всяко външно прекъсване. Немаскируемите прекъсвания влизат по линията NMI – non maskable interrupt.

Слага се контролер на входно – изходните прекъсвания. Прави се конкретна схема, която се вкарва в chipset-a.



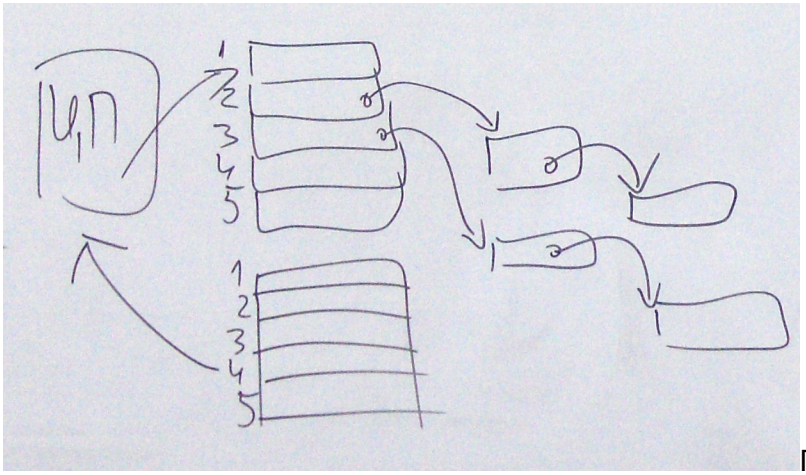
Логиката е следната. Входно – изходните прекъсвания се номерират от IRQ0 – IRQ7 и IRQ8 – IRQ15. Тоест има 16 линии на interrupt request като втората осмица е вкарана каскадно. Контролерите на прекъсванията на компютъра са направени така, че по номера на линията върви приоритет, малкия номер е по-приоритета. Поради каскадните свойства приоритета върви така – 0 1 8 9 10 11 ... 15 2 3 ....7. Това е реалния приоритет. Нулевото прекъсване е за refresh на паметта, 1 е прекъсване от клавиатурата. Дисковите прекъсвания са на втория чип.

Контролера помни откъде идват заявките и потвърждава на устройството откъдето идват и това устройство се идентифицира пред процесора. Устройството си дава 8 битов код на прекъсване IC –interrupt code. С него устройството идентифицира вектора, по който процесора да търси програмата на прекъсване. Когато централния процесор получи заявка за входно-изходно прекъсване той пуска INT A и получава адреса на един от 256-те вектора, които съдържат адрес на програма.

Процесора работи със регистрите IP и FLAGS. От вектора се зарежда нов IP и нов флагов регистър. По тази схема при подаване на различен код на прекъсване става разклоняване на входно изходните прекъсвания. На немаскируемите прекъсвания процесора автоматично отиват на вектор 2 откъдето зарежда новата програма.

Какво прави централния процесор като приеме прекъсване. В него влиза заявка, той я приема или не и се включва стандартен механизъм за обслужване на прекъсвания. Има два начина за обслужване на прекъсванията:

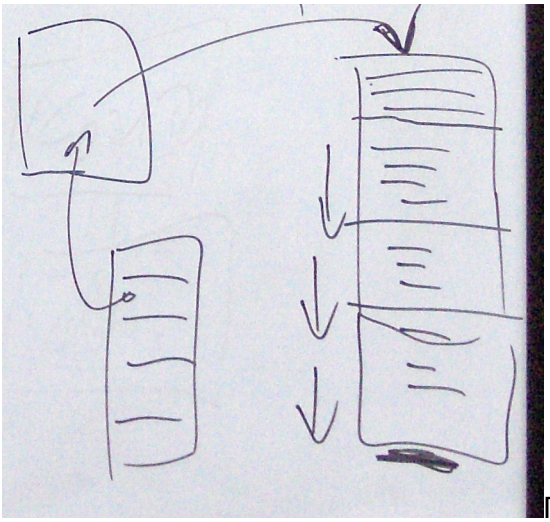
1) със фиксиран адрес в паметта
2) със стек



Прекъсване със фиксиран адрес в паметта

1) В процесора постъпват прекъсванията. Те се идентифицират потипово (1ви до 5ти тип) в зависимост от типа процесора запазва техния контекст в старо състояние. Минималния контекст включва регистрите на програмния брояч и флаговете. Настъпва прекъсване номер 5. Той запаметява старото състояние под номер 4. И от фиксирани адреси зарежда нов програмен брояч и нов флагов регистър за съответния тип прекъсване (при стековата организация контекста се запазва на върха на стека). Новото състояние се чете от вектор на прекъсване. Определя се номер на вектора експлицитно или имплицитно в централния процесор.

   Връщането от прекъсване се осъществява от специална инструкция IRET. Тази инструкция отива, взима стария контекст и заменя текущия контекст. Новия адрес се чете от фиксиран или определен вектор.

 Прекъсване със стек

2) схемата със стек е по-удобра при влагане на прекъсвания. Ако при прекъсване от втори тип се допуска вложено прекъсване от същия тип значи трябва да се прави списък със стари контексти, иначе новия контекст ще изтрие стария. Ако няма списъчна структура могат да се влагат прекъсвания само от различни типове, тогава няма засичане и не се нуждаем от списък. Схемата със стек е удобна и възприета. След края на прекъсващата програма IRET връща контекстите от върха на стека.

При стека може също да се объркa нещо. Всяко извикване на push трябва да има съответен POP. Във VAX има 5 различни стека за различни нужди. Стекове за операционната система, за приложни програми и така нататък. За да е чисто е добре да има отделен стек на прекъсванията за да не се окаже че там няма контекст а круши. В SPARC контекста много често се прави разширен. При Intel контекста е минимален. RISC компютрите пазят всички общи регистри. Там регистрите са много и прекъсването става тежко за обслужване. Част от регистрите винаги се запазва, друга част не се местят. Но обслужването зависи от броя на регистрите. Има конвенция програмата която обслужва прекъсването да освобождава регистрите преди извикването на IRET трябва да ги възстанови за да може контекста да бъде чист. Но това зависи от програмиста. Всички програми, които обслужват входно-изходните прекъсвания се наричат драйвери.

Векторите на прекъсванията се задават фиксирано и обслужват паметта, където се запазват адресите на програмите за обслужване на прекъсване. Изпълнението на инструкцията INT с номер на вектор е преход към вектора, в който се взима нов програмен брояч. За разлика от jump се зарежда нов флагов регистър с нови маски. Големия недостатък на Intel , е че липсва разделение между състояния на операционна та система и на приложните програми.

Критичните инструкции могат да се изпълняват само в състоянние supervisor. Обикновени програми не могат да изпълняват привилигировани инструкции и състояние на Supervisor не може да се направи само със смяна на флага. Трябва да се направи прекъсване. Така трудно може да влезе вирус. Вирусът е приложна програма, която прави неща, които не може. Ако операционната система е строго разделена от приложните програми не може да се зарази.

18.Осъществяване на входно-изходен обмен. Входно-изходни процесори

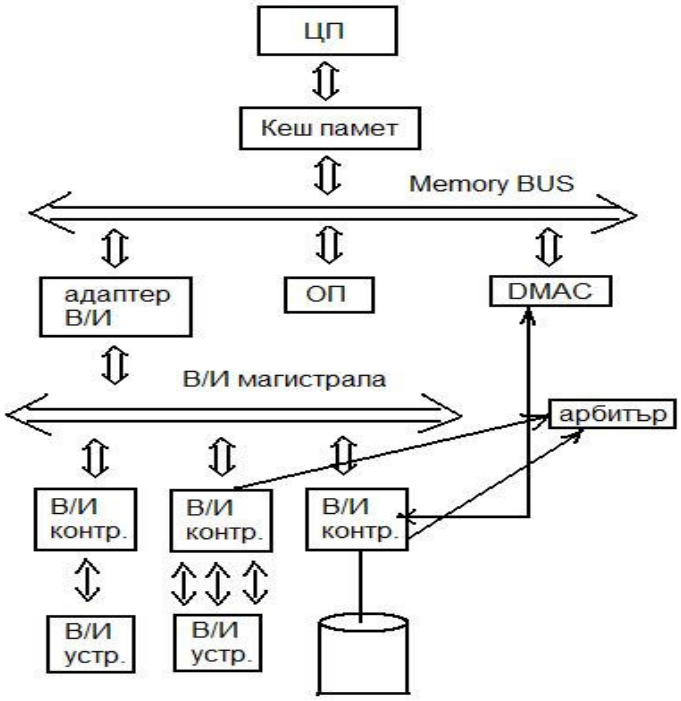
Простата машина на фон Нойман включва само централен проц есор и оперативна памет. Централния процесор работи само в оперативната памет. Чете данни, изпълнява инструкции, записва резултати, образува цикли, прехвърля се на различни адреси и така нататък. В даден момент съдържанието на оперативната памет ще се обработи. Затова трябва да има начин от работни адреси на ОП да се изкарва и вкарва информация. От тук идва и идеята за входно-изходна система.

Входно-изходната система е свързана с това как се прави вход и изход към ОП и ЦП. В нея са включени и така наречените входно-изходни устройства – това са устройства за въвеждане, извеждане и съхранение на памет. Те биват три типа

- входни – въвеждащи(клавиатура,скерер ...)
- изходни – извеждащи(монитор, принтер,..)
- междинни – запомнящи (магнитни дискове, оптични дискове, флаш-памет)

В структурно отношение има два начина системата ЦП ⇔ кеш ⇔ ОП да се разшири и да включи входно- изходната система:

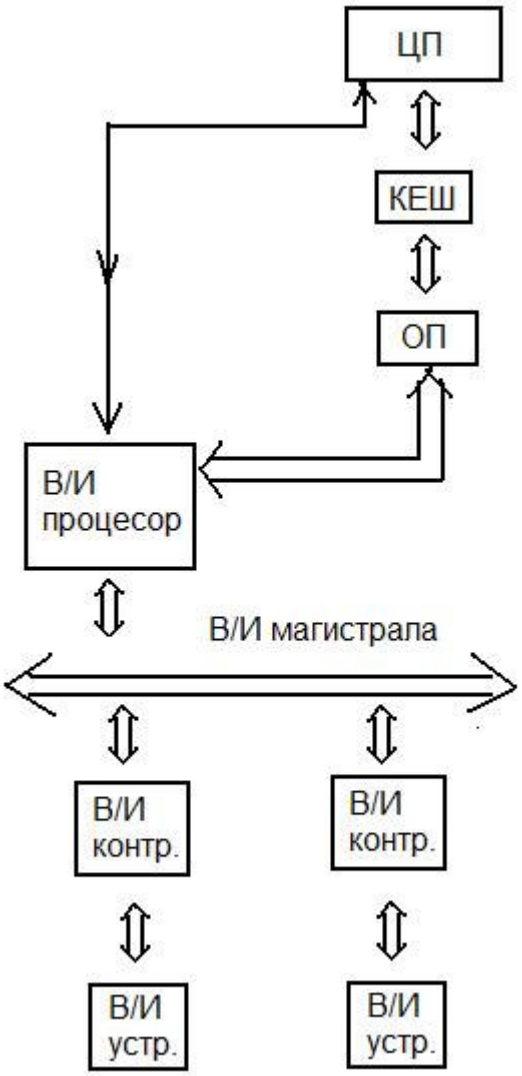
1) Прави се магистрала на входа между кеша и оперативната памет (memory bus)



Достъпът до ОП може да бъде или чрез специализирано устройство наречено DMAC – direct memory access controller, управляващ достъпа, или чрез адаптер на входно-изходния обмен, който прехвърля данни от memory bus към специална входно-изходна магистрала. От тук достъпът до ОП се получава само през общата шина на паметта . И по двата пътя става обмен. Към входно-изходната магистрала се включват външни устройства, които са специализирани или универсални, затова между входно-изходните устройства и входно-изходната магистрала се слага входно-изходен контролер. Има групови входно-изходни устройства, които се управляват с един общ контролер. Има други единични устройства, които си вървят със собствен контролер (например клавиатура).

При груповите устройства контролерите са отделно и този входно-изходен интерфейс се стандартизира. Това е постановката, където има една шина за достъп до паметта.

2) Другата структурна схема е когато паметта има почти независими (конкурентни) шини



– една шина за централния процесор и друга за входно-изходния процесор, който изпълнява входно-изходни програми. Тяхна задача е да осъществяват входа и изхода. Този процесор излиза на своя собствена магистрала, към която са прикрепени входно-изходните контролери. Достъпът до паметта се командва конкурентно от два процесора, като когато единия ползва част от нея, достъпът до тази част се заключва за другия процесор. Заключването става на страници. Ако единиият процесор работи с една страница, тя е заключена спрямо другия. Обменът с различни страници става паралелно. В паметта има устройство, което осигурява заключването на страница, спрямо едната или другата си шина. Може да има определени приоритети като обикновено централният процесор е с по-висок приоритет. Заключването е подхардуерно. Проблемът с управлението на кеша не е тривиален. Тази схема е много по-скъпа. Характерна е за машините CRAY, суперкомпютри с много сложна структура. Персоналните компютри работят по първата схема.

Магистралите имат 3 компонента – адресни линии, управление и линии за данни. Основно има 4 пункта, по които се разглеждат магистралите:

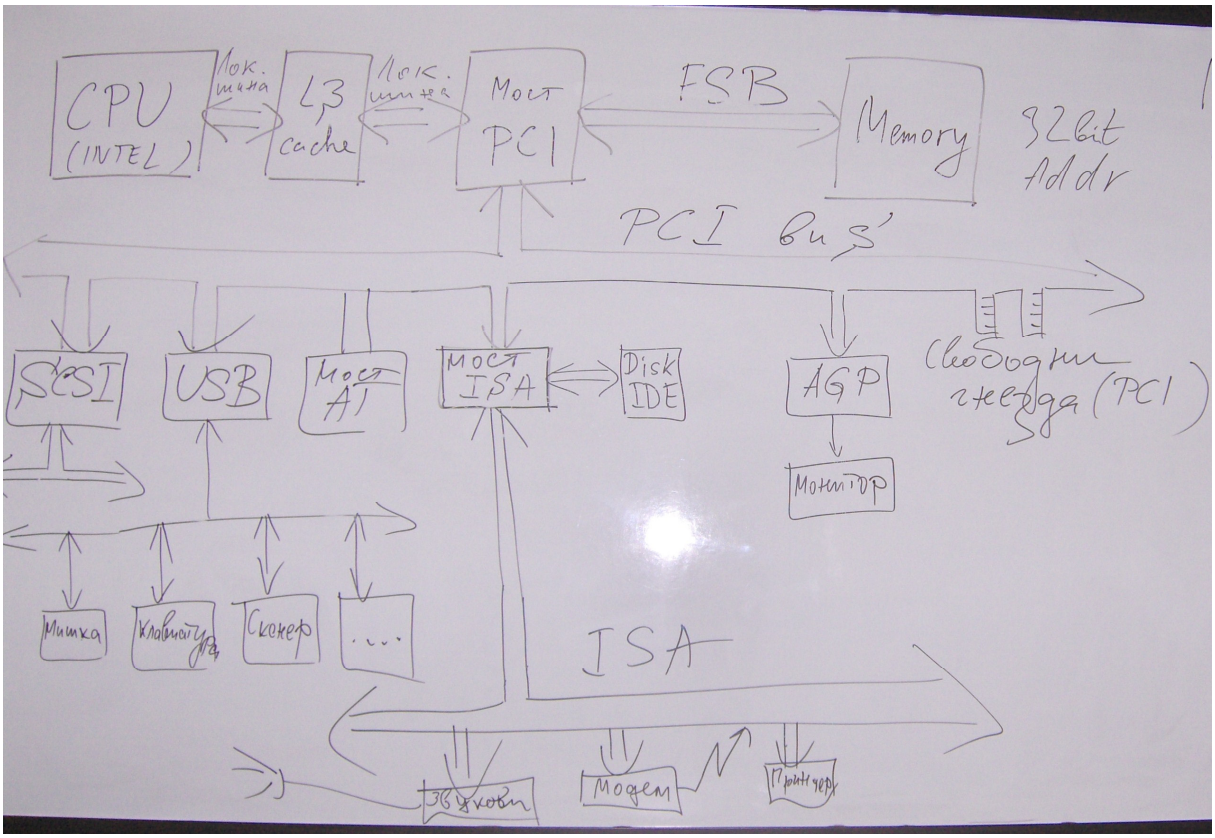
- clocking – тактуване на обмена на данни по шината. Може обмена да е и асинхронен, когато няма clocking. При него няма определени времена за една транзакция. Прави се обемна между активното и пасивното устройство, и когато той приключи, пасивното пуска сигнал, че е готово. Асинхронните магистрали са по-сигурни, но са по-бавни. При тях всичко се квитира(?). При синхронните магистрали транзакцията трябва да стане в рамките на такта. Тактовете са едни и същи винаги с определена честота. При предаването на данните няма изчакване за потвърждение. Такта е независим. Всичко се тактува по отделно. Определящо е продължението на такта – колкото по-къс е такта, толкова по-бърза е магистралата, но и възможността за грешка е по-голяма.
- switching (превключване) – определя кога магистралите се превключват. Осъществява се връзка от тип точка точка. Двама прехвърлят данни. Най-често оперативната памет, която изпълнява пасивни функции, е единия субект, а другия е някой от включените компоненти.Въпросът е за колко време магистралата се ползва от 2 субекта. Вариантите за осъществяване на транзакцията са:
    - атомарно - само за обмен на единица данни, колкото са линиите. При смяна на такта, други 2 субекта си обменят данни. На всеки такт субектите са различни.
    - друг вариант езапочват транзакцията и държат магистрала, докато не я приключат – извършват цяла транзакция. Обменят един цял блок от данни (примерно 16 байта). Тази транзакция е съставена от няколко атомарни транзакции.
    - split transaction – припокриване на транзакцията. Две двойки устройства започват паралелен обмен и магистралите се превключват, става припокриване по отношение на транзакцията (характерно за машини VAX). При персоналния компютър се работи на принципа на цяла транзакция. В двойката субекти участва паметта. Това е само по отношение на преминаването на данни. Не може да има мултиплициране по отношение на писане в една и съща област от паметта. При входа и изхода, най-големият проблем е консистентността на паметта.

- арбитраж – определя се кои устройства ще ползват магистралата. Това става по два начина:
    - със специално устройство наречено арбитър. Всички компоненти имат отделни линии за заявки към арбитъра. Използването на магистралата става в два цикъла
        - захващане – входно-изходното устройство подава заявка. Арбитърът дава разрешение на този, който ще заеме магистралата. Арбитърът определя приоритета.
        - транзакция

ТЕМАТА НЕ Е ДОВЪРШЕНА

19. Входно-изходни системи и магистрали

Така изглежда шината при стандартен персонален компютър



При тази схема L2 cache паметта се намира в централния процесор (CPU), а L3 cache-а е извън него. Обменът на данни между CPU и L3 cache-а се осъществява с локална шина. Такава се използва и за връзка с голямата оперативна памет. PCI bus е магистрала. Ролята на наличния

PCI мост е да отделя двете шини: първата бърза за връзка с ОП, втората по-бавна за осъществяване на входно-изходни операции. В даден момент се комуникира само по едната от двете шини, т.е. на лице е арбитраж. Арбитърът определя кое устройство ще ползва магистралата в определен момент (схема за приоритет). Арбитражът се осъществява по следния начин: по време на „раздадена“ магистрала се пуска заявка към арбитъра(това е така наречения цикъл на захващане); след това се определя заявката с най-висок приоритет; след това се заема освободената магистрала от най-приоритетната заявка (устройство) (цикъл на заемане). Технологичното развитие е, че ОП се отделя от другите устройства с по-приоритетна шина. Към PCI bus са свързани и други магистрали(SCISI, USB).

- SCSI се използва за по-бързи устройства, а USB съответно за по-бавни такива.

- USB интерфейса се използва за по-бавни последователни устройства.
- По-рано е имало само **ISA** – industrial standart architecture. Нямало е е PCI шина. ISA се е използвала и за паметта. Тя е много по-бавна. При днешните архитектури през мост се влиза в ISA шината за работа с по-стари устройства. На дънната пладка има някаква вградена ISA шина без свободни куплункги, както е за PCI външните адаптери. ISA – моста излиза на специализирана IDE шина за единични дискове.
- Друга шина към PCI магистралата е **AGP** – единична шина, към която зе закачамонитора. Когато щината е за едно устройство и няма арбитраж.
- има свободни гнезда.

Относно управлението на входа и изхода е важно по какъв начин се комуникира с процесора. Тази система е за комуникация с паметта.

- Ако процесора се обърне със специални инструкции през магистралата към входно-изходното устройство, той записва данните в регистър и го използва. Това е най-примитивния начин. Има предимството, че като процесора комуникира с външното устройство имаме консистентност на данните в кеш паметта. Данните ще минат през нивата на кеш. Като се прави запис чрез write back или write through записва резултатите в паметта.
- Набелязва се адресно пространство, което наричаме входно-изходно адресно пространство. При персоналния компютър се определя обема от адреси от 0 до 64К за входно-изходно адресно пространство. Централния процесор, когато работи със входно-изходното адресно пространство , пуска идентифициращ управляващ сигнал. Процесора вади адреса в паметта и го съпровода със MEMR или MEMW управляващи сигнали. Това са две отделни управляващи линии извън 32те линии на паметта. Ако кеша е в режим write-back то той казва MEMW. Същите адреси се съпровождат със IOR и IOW. Тоест 32 битовият адрес е съпроводен от един от четирите сигнала. Това указва дали комуникацията е със ОП или със входно-изходната памет и дали ще пише или ще чете. Разделянето на адресното пространство на входно изходно и на паметно се съпровожда със управляващ сигнал. Така се адресират външните устройства. Те трябва да се нагласят за кои адреси отговарят. От адресите се взимат десните 16 бита при входно изходен управляващ сигнал. Порта, нагласен за тези адресни пространства прави четене или запис. На магистралата трябва да има еднозначност на реакцията на входно-изходното устройство. Преди това е ставало със ръчно поставяни jumper-и а сега става автоматично.

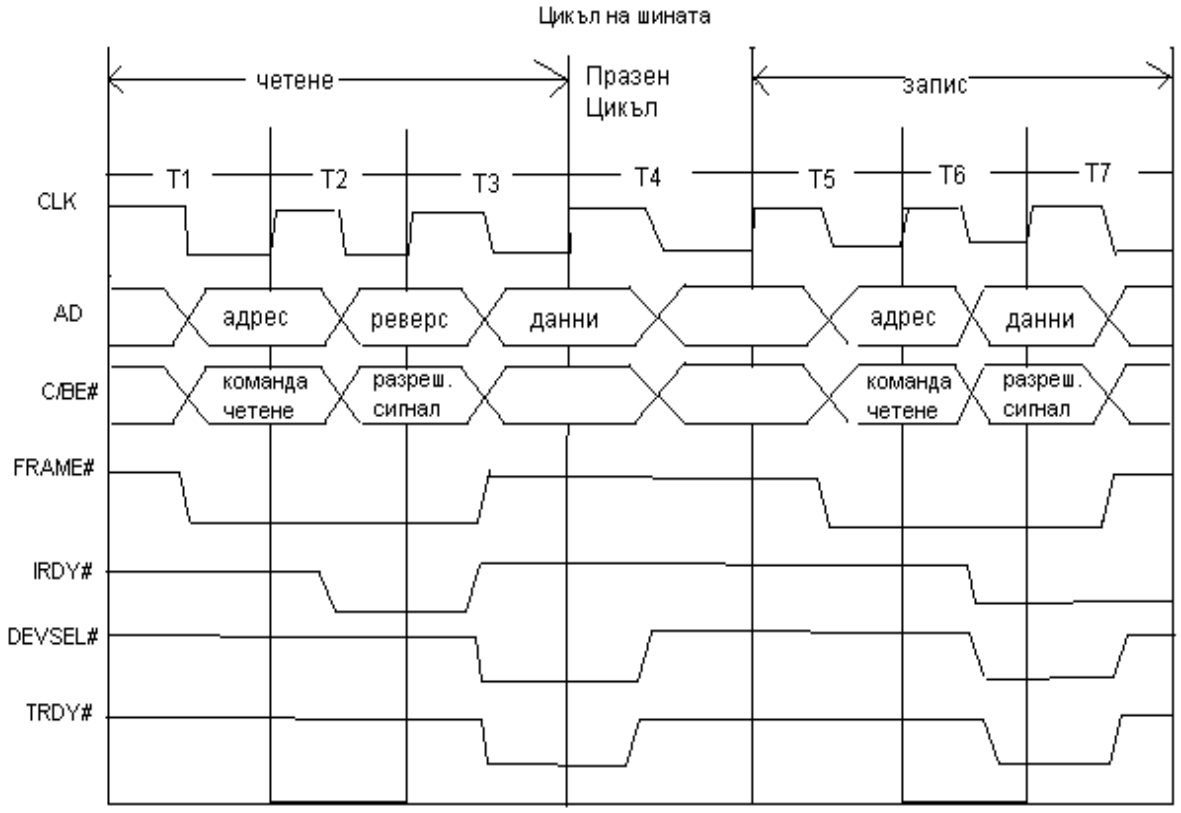
- при по- сложните компютри този начин е неприложим. При тях централния процесор не се занимава с входа и изхода. Има само инструкции за входно-изходния процесор, който извършва целия вход и изход. При входно-изходното адресно пространство се поставя въпроса как централния процесор явно започва входно-изходна операция. Програмата знае разпределението на входно-изходните адреси и юе прави вход и изход направо през клавиатурата. Клавиатурата си има контролер, който се включва към магистралата. Контролера има три порта (като регистри) и те имат три адреса (60,61,62). Единия порт е за данни, втория за управление, третия за статус. Процесора ако иска да чете от клавиатурата адресира порта за съсстояние и чете състоянието. Той там търси битовете,които указват, че са постъпили нови данни. Когато има такива той прочита от порта за да нни и записва в порта за управление че е прочел данните и клавиатурата вкарва следващия код. По-сложно е когато входно-изходните устройства трябва да извършат подготвителни дейности (например при магнитния диск). При него трябва да се каже отиди на този сектор и го прочети. Когато намери този сектор, той ще го прочете от носителя и че отиде в оперативната памет. Но централния процесор трябва да разбере кога диска е гото, тоест кога главите намират сектора, прочитат го и данните преминават в ОП. По какъв начин програмата в ЦО разбира кога е завършила входно-изходната операция:
    - програмата стартира операцията като в порта за управление записва започни да търсиш сектора. Диска търси сектора и той се записва в порта за състояние. Процесорната програма проверява периодично байта за състояние (polling) Програмата пуска непрекъснато четения от порта за състояние. Това е бавен подход.
    - Процесора задава на входно изходното устройство работа и продължава да изпълнява други инструкции. Контролера казва на процесора, че си е свършил работата. Тогава трябва да има система на прекъсвания, която позволява по асинхронна външна заявка да се прекъсне работата на процесора, за да се вкючи друга програма. Прексването извиква допълнителна програма, като централния процесор в това време прави нещо друго. ЦП е активен при започване, при завършване контролера прави прекъсване(?).
    - Спезциални начини, при които централния процесор не работи с входа и изхода. В DMAC контролера се зарежда блока от адреси в паметта, указва се кое устройство ще чете/пише и DMAC формира адреса към паметта и работи директно с нея.

  Проблема с DMAC контролера и със входно изходния процесор е свързан с кеша. Губи се консистентност ако не са отделени адресите. Ако в кеша са попаднали такава група адреси, които не участват във входно-изходни операции. Централния процесор, четейки от адресите в кеша взима грешното съдържание откъдето следва загуба на консистентност на паметта. При входа и изхода се се отделят блоковете за вход/изход. Процесора или работи с write-through и по време на четенето процесора не работи с тези адреси. Втория проблем е свързан с виртуалната памет. Виртуалната памет е такава че оригиналното съдържание е върху диска и се прехвърля на страници. Значи входно-изходния процесор трябва да има досъп до таблиците или страниците в които се чете/записва трябва да се блокират.

  Виртуалната памет се поддържа от оперативната памет. При нея адреса се прехвърля от ефективен в линеен във фактически или физически адрес. Чете се в каталога дали

търсената страница е в паметта. Ако е на диска я докарва и си работи с нея. DMAC контролера трябва да мине или през каталога и да работи директно във физическата памет ако страницата не е блокирана, или да докара страницата във физическата памет и да я блокира докато завърши вход/изхода.

PCI шина - PCI local bus е компютърна шина за включване на хардуерни устройства в компютъра. Тя е разработена от Intel. Първоначално е 32 битова, но в момента има и 64 битови. Името PCI означава Peripheral Component Interconnect. PCI магистралата е често срещана в днешните персонални компютри, като замества по-старите шинни интерфейси като ISA и VESA като стандартна шина за разширение. Прехвърлянето на данните става синхронно, тактува се със CLK 33/66 MHz с коефициент на запълване 1/1



Стандартните линии на PCI шината са:

- 1 линия за **CLK** по която се тактува прехвърлянето на данните – PCI е синхронна шинна архитектура, като всички трансфери на данни се изпълняват спрямо системно тактуване(CLK). Първоначалната PCI спецификация позволявала максимална тактова честота от 33 MHz, разрешавайки на една шинна транзакция да се извърши на всеки 30 наносекунди. По късно версията 2.1 на PCI спецификацията разширява възможностите

до поддръжка на опериране при 66 MHz, но по-голямата част от днешните персонални компютри все още имплементират PCI шина със максимална скорост от 33 MHZ.

- PCI имплеметира 32 битова мултиплексна шина за адрес и данни **AD[31:0]**. Архитектурата позволява начин за поддръжка на 64 битова шина за данни чрез по дълъг слот за връзка, но повечето от днешните компютри поддържат само 32 битов трансфер на данни. При скорост 33 MHz, 32 битов слот поддържа максимална скорост на трансфер от 132 MB/секунда, а 64 битов - 264МБ/секунда
- Линия **PAR** за отчитане на четност – отчита четност по линиите на сигналите AD и C/BE. Този сигнал се изчислява на същия интервал от време, както и AD, но е забавен със един цикъл за да позволи повече време да се изчисли четност.
- **C/BE#[3:0]** - Командите за шината(C) и byte enables(BE) са мултиплексирани върху тези линии. През фазата на адресиране тези сигнали носят командата за шината, която дефинира типа на трансфера, който трябва да се изпълни. Списъкът долу показва значението на валидните кодове на команди. През фазата на трансфер на данни тези сигнали носят byte enable информация, тоест кой байт се взима, C/BE[3] е byte enable за старшия байт, а C/BE[0] – за младшия. Сигнала C/BE се подава само от активното устрйсво. Видове команди:

  0000 Interrupt Acknowledge

  0001 Special Cycle

  0010 I/O Read

  0011 I/O Write

  0100 Reserved

  0101 Reserved

  0110 Memory Read

  0111 Memory Write

  1000 Reserved

  1001 Reserved

  1010 Configuration Read

  1011 Configuration Write

  1100 Memory Read Multiple

  1101 Dual Address Cycle

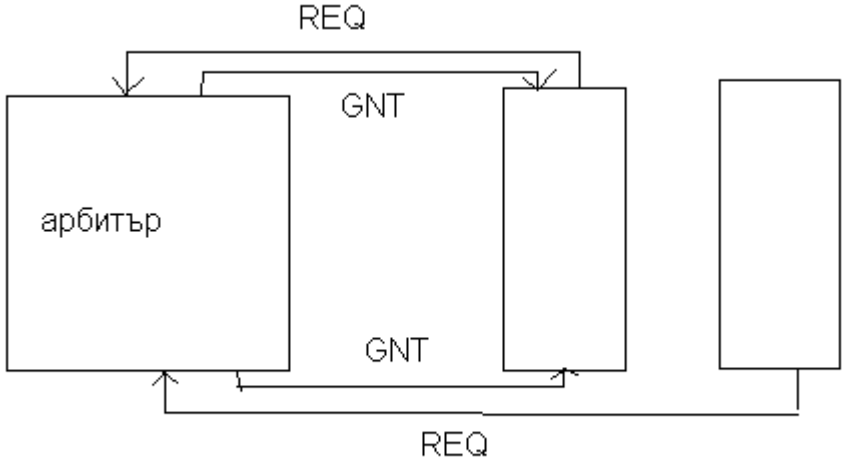
  1110 Memory Read Line

  1111 Memory Write and Invalidate 1 линия за frame, която указва да се гледат сигнала на AD и C(те са стабилни)

- Магистралата има арбитър в основния мост и арбитъра определя две устройства – задаващо (главно) и получаващо. Фрейм на цикъла се подава ниско от подаващото устройство през първия такт. Адресната фаза на транзакцията става през първия такт след прехвърляне на сигнала **FRAME#** от високо към ниско. Ако подаващото устройство има намерение да изпълни транзакция само с една фаза с данни, тогава то ще върне FRAME# отново високо след само един цикъл. Ако трябва да се изпълнят множество фази със трансферн на данни, подаващото ще държи сигнала ниско през всичките освен последната фаза.

- сигнала **IRDY#** – ready - означава, че задаващото устройсво се намира в операция четене и е готово да адресира подчиненото устройство чрез команда запис или четене. При зададена команда четене със сигнала IRDY задаващото показва на подчиненото кога е в състояние да вземе тези байтове. Това е сигнал който се определя от задаващотото устройство.

- **IDSEL#** - Initialization Device Select се използва за идентификация на чипа по време на PCI конфигурирана транзакция на четене/писане. IDSEL е уникално за всяко устройство, което има собствен уникален 8 битов код. Това позволява на механизма за конфигурация индивидуално да адресира всяко PCI устройство в системата. PCI устройство се избира за транзакция само ако IDSEL сигнала е високо, старшите два бита в AD са 00 индикирайки цикъл на конфигурация от тип 0, и командата на линията C/BE сигнализира през фазата на адресиране или конфигурационно четене, или конфигурационно писане.
- **DEVSEL#** – device select – това е сигнал , който се връща към задаващото устройство и означава разпознаване на подчиненото устройство
- **TRDY#** – transfer ready - това е сигнал, който се задава от подчиненото устройство когато то е готово да изпълни операцията (трансфера)
- Сигнала **STOP** означава че подчиненото устройство изисква спиране
- **PERR** и **SERR** са грешки по четност
- **REQ** и **GNT** са отделни линии за request и granted

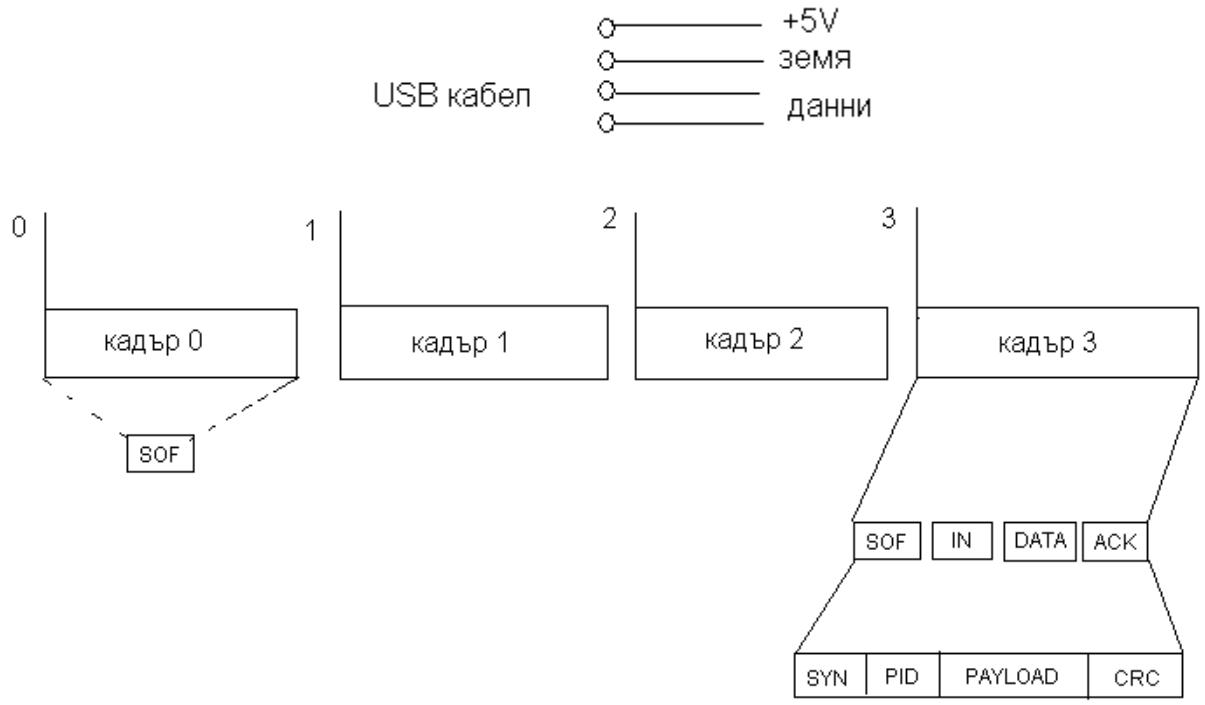


Това са сигнали предназначени за работа с арбитъра. Той определя кое ще бъде задаващото устройство. Само едно устройство може да получи GNT от арбитъра и то

вече е новото задаващо устройство. Една тразакция има 7 такта – 3 такта четене, един празен такт и през 5, 6 и 7ми такт се осъществява запис.

При четене е определено задаващо устройство. При падане на тактовия сигнал изкарва адреса на линиите за адрез. Изкарва команда за четене по линията за команда и се подава сигнал за готовност по FRAME. Тъй като адреса излиза от задаващото устройство имаме команда четене. Подчиненото устройство трябва да сложи на тези линии данните които са на съответния адрес. Адреса се подава от задаващото устройство. Подчиненото устройство трябва да сложи данните на следващия такт, тоест имаме reverse. Линиите се обръщат и трябва да има един такт за да премине управлението към подчиненото устройство. След този такт подчиненото пуска данни на линиите. Всичко се командва по падащия фронт. Вдига се сигнала TRDY което значи, че подчиненото устройство е изпълнило транзакцията. След това имаме един празен цикъл. Задава се адрес към подчиненото устройство, но понеже командата е запис няма reverse. Пуска адреси и зпаисва данни. Подчиненото устройство трябва да пусне TRDY. Проблемът може да дойде от подчиненото устройство когато не върне TRDY на време. Шината PCI е синхронна шина защото всичко става в рамките на такта.

USB



Universal Serial Bus (USB) замества различни интерфейси за свързване на бавни периферни устройства (RS-232, клавиатура, ...). USB е универсална шина, направена така, че да отговаря на всички напрупани с времето изисквания, създавали проблеми. Идеята е била всеки неграмотен потребител да може да включи периферно устройство към интерфейса, като за това не са нужни никакви допълнителни настройки, никакви ръчни настройки и със сигурност без нужда от отваряне на кутията на компютъра. Освен това има само един тип кабел за включване на всички устройства с цел да не може да се объркa начинът на включване. Друга важна особеност е, че периферните устройства получават захранване от USB шината. Към един компютър могат да се свържат максимум 127 устройства, като има възможност за поддържане на бавни периферни устройства в реално време. Освен това устройствата могат да се включат и по време на работа на компютъра. USB Flash памет може да се включи също винаги, но семантиката на изпълнение е различна в зависимост от това кога се включва. Ако файловата система остане незатворена, може да се увреди. Не може да се допусне reload след включване на периферно устройство. Примери за бавни периферни устройсва са: клавиатура, мишка, фотоапарат, цифров телефон, ... Максималната скорост на обмен при технологията USB 1.1 е 1,5MB/sec. USB шината е дърво скорен – централен хъб (концентратор, коренен концентратор). От него излизат 4 извода, като на всеки извод може да се закачи по един външен хъб, от който излизат 4 извода... USB устройствата могат да се връзват в произволен ред едно след друго (наредбата няма значение). Технологично кабелът е съставен от 4 проводника. Захранването е 5V, 0V е земя, останалите 2 проводника са за данни. Данните се предават импулсно – 0 – с импулса, много нули – със серия регулярни импулси. Централният хъб е водещ, не е магистрала, а шина, която се управлява строго от компютъра. Всяка милисекунда централният хъб формира кадър, който се пуска по дървото. Колкото и да е голям кадърът, той трябва да е по-къс от 1 ms. До следващата ms кадърът е разпространен. Концентраторът е длъжен всяка милисекунда да имитира кадър, дори и да е празен. SOF = Start Of Frame. В кадъра има маркери:

1. in/out – дали се получава от устройството или се изпраща към устройството (in – от устр. към хъба, out – обратно). Освен това определя 7-битов адрес на устройството, който се раздава на устройствата в момента, в който се включат. Адресът, естествено, е уникален. Пакетът на данните е отделен от маткера и след маркера хъбът може да чака пакети от външното устройство (ако е in).
2. ACK – указва какво е станало с данните - указва, че пакетът е дошъл неповреден
3. NACK (Negative ACK) – указва, че нещо се е случило с данните
4. STALL – при in устройството може да пусне stall, което означава, че не е готово да приема и иска изпращачът да почака

Пакетът данни съдържа:

1. SYN – синхронизиращо поле с дължина 8 бита
2. PID (Packet Identifier) – 8 бита
3. PAYLOAD – 64 bytes максимум
4. CRC – Cyclic Redundancy Code – указва дали правилно е прехвърлена една информация. Най-простият начин е с контролен бит, който върху група разреди брой четност и допълва до четен или нечетен брой. Този метод е много бърз за хардуерна реализация, но има и недостатък – много слаба сигурност (например ако има четен брой грешки). Разработват се по-сложни начини за кодиране. Идеята е начинът на изчисление на кода да е хардуерно подходящ – формулата да е полиномен код. Измислят се стандартизирани кодове. Наричат се циклични, защото изчисленията се натрупват. CRC е 16 бита, има

специална формула и алгоритъм – устройството, което формира пакета слага в края CRC и получателят пресмята по същата формула кода и сравнява двата – ако са еднакви – няма проблем. Ако не съвпаднат, значи има грешка и се праща NACK.

CRC – цикличен код с излишък. ECC – Error Correcting Code – код, който поправя грешки – тези кодове най-често са много дълги, с голям излишък и доста сложни. Например 16 байта CRC са достатъчни за 1000 байта информация, а ECC е приблизително 10 пъти по-дълъг от CRC. При комуникации се използва CRC – ако пакет пропадне се resend-ва. Съществуват много пътища от Централния хъб към периферните устройства, но има само 1 път от всяко периферно устрйство до Централния хъб.

Типове кадри:

1. Кадри за управление – Централният хъб се координира с устройствата
2. Масиви данни – за предаване от устройство към устройство на данни
3. Изохронни – устройство с реално време (цифрови телефини). При такива устройства е определено равномерно подаване на кадри. Няма паузи. Разликата между тези кадри и обикновените масиви е, че при обикновените в случай на NAC кадърът се повтаря, а при изохронните – не се повтаря.
4. Прекъсване – няма линия от устройството към хъба. Клавиатурата, например, работи с кадри на прекъсване. На всеки 50 ms се изпраща кадър, който съдържа всички кодове, които са натрупани.

Периферните устройства са следните типове: входни, изходни, входно-изходни и външна памет.

Устройсвто	Тип	Скорост KB/s
Мишка	I	0.01
CRT	O	60 000
LAN	1/0	6 000
Лента	памет	2 000
Диск	памет	10 000

Както виждате от таблицата, мишката е много бавно периферно устройство – приблизително 10KB/s.

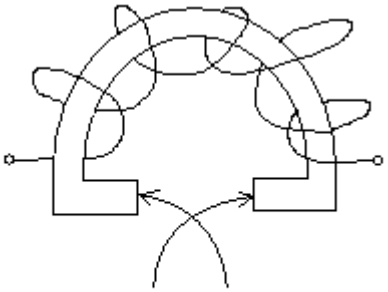
20.Външни запомнящи устройства на магнитен диск.

Физическа организация на данните и операции изпълнявани от дисковете

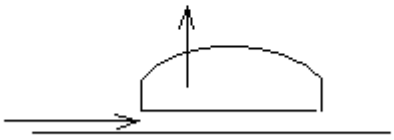
Hard disk-ът е магнитен диск. Както знаем от предишни лекции, той е задължителна част от йерархията на паметта(разширение на Оперативната памет). Твърдият диск е съставен от няколко твърди метални плочи с формата на диск. Плочите са изработени от много твърд метал, като страните на плочите са покрити с магнитно покритие с феромагнитни свойства. Ето графиката на хистересизния цикъл на намагнитване:

Както се вижда на рисунката, има 2 устойчиви състояния, между които се преминава по различен начин в зависимост от посоката. Най-малката различима област от покритието на диска ще наричаме феромагнитен домейн с устойчив цикъл на намагнетизиране. Преминаването от едно устойчиво състояние в друго предизвиква промяна на магнитния поток на домейна. Промяна в магнитния ток индуцира малък електрически сигнал. Особеното е, че като се наелектризира домейна, не се забравя посоката. Един намагнетизиран домейн е много стабилен – възможно е да запази заряда си повече от 50 години. Домейните могат да се намагнетизират в 2 посоки: S – N и N – S(от север на юг и обратното). Когато домейнът се движи, може да създава 2 различни посоки на магнитния поток.

Магнитните силови линии са от север на юг. Домейните са разположени много близо и ако мине една намотка, ще се предизвика индуциране на електричен ток и в намотката ще протече слаб сигнал в едната или в другата посока. Това е случаят с четяща намотка. Намотката представлява потковообразна сърцевина, около която има намотан тънък проводник.



В четящата глава няма сигнал, в нея се индуцира такъв. В записващата глава има сигнал, за да се намагнетизират домейните в подходящата посока. Колкото по-малки са домейните, толкова по-добре. Плочите са свързани с шпиндел, който ги държи в постоянно взаимно положение една спрямо друга. На една плоча има до 2 работни повърхности. Навремето външните страни на двете крайни плочи не са били използвани. Всяка плоча има отделна четяща глава. При твърдите плочи главите са летящи.



Профилът на главата има специален радиус, за да може да лети. Намагнетизираната повърхност на плочите увлича газовите молекули наоколи и при въртенето в близост до плочата се образува вятър, който образува подемна сила. От горната страна има пружини, така че да не се издигне главата прекалено нависоко. По този начин се поддържа константна позиция на главата спрямо плочата – 1-2 микрона. Важно е, че главата не се трие в плочата, за разлика от магнитните ленти, при които главата опира в лентата, поради което тя се движи по-бавно, за да не се отдели голямо количество топлина от триенето. Плочата се върти с голяма скорост. Скоростта се измерва в обороти в минута(об/мин). Възможни скорости са: 2400, 3600, 5400, 7200, 10 000, 15 000. Всички дискове се въртят едновременно, а всички глави са в карета и са на една ос (през всички глави може да се прекара права). Главите се движат заедно по радиуса на плочите. Има механизъм за аварийно изтегляне на главите, след което щпинделът се върти по инерция известно време. Ако главите не се отделят моментално и започнат да трият по повърхността ще я увредят. Въздухът се оказва твърде груб газ, затова вече дисковете се затварят в херметични капсули, в които има инертен газ – смесица от аргон, неон, ... При въртенето главите застават във фикцирани позиции – има определен брой такива позиции (фикцират се с точност до микрон). Колкото по-малък е радиусът, толкова по-малко е разстоянието между отделните писти. Всяка фиксирана позиция определя една писта. Пистите са мислени концентрични окръжности. Каретата се движи дискретно навън-навътре по радиуса. Всички писти вертикално образуват цилиндър. Шпиндела при всички харддискове се движи с

една и съща скорост на въртене, която е постоянна и когато се достигне – тогава влиза каретата с главите. Ъгловата скорост на диска е константа, но линейната скорост на пистите е различна. Във външните писти плътността на битовете е по-малка, а битовете близо до центъра са по-близо. Преди външните и вътрешните писти са били с един капацитет. Сега външните писти имат повече сектори от вътрешните. Броят сектори на писта вече не е константно число. Всички сектори, обаче, са с един и същ размер(например 512 bytes), а пистите имат тазличен брой сектори.Външно се маркира къде е началото на пистата. Даден радиус се определя за начало на всяка писта(някъде на шпиндела). Чрез най-долната повърхност се прави синхронизация. Всички писти започват от един и същи радиус. Дисковете са така направени, че могат да обработват само започвайки от началото на пистата. Затова като излети главата чака началото на пистата и оттам брои определен брой сектори докато стигне до желания сектор. Капацитет на диска е количеството байтове, които можем да запишем. Фирмите производители на твърди дискове често пресилват при оценката на капацитета на един диск като например смятат 1KB = 1000 bytes, което, както вече знаем не е така(поне в рамките на курса Компютърни архитектури) или обявяват общия брой байтове записани на диска, част от които са служебни и на тях не може да се записва полезна потребителска информация. Във всеки сектори има служебна част – накрая на сектора има Error Correcting Code. При всеки запис контролерът записва кода в края на сектора и при всяко четене този код се чете и сравнява. При твърдите дискове е неизбежно използването на Error Correcting Code, защото ако информацията се повреди няма друг начин, по който може да се възстанови.При бавните дискове междусекторното разстояние е по-малко от това при по-бързите. Има проблем – когато искаме да прочетем няколко последователни сектова, се оказва, че дискът се върти прекалено бързо за да се прочетат всичките. Затова се използва лъжливо подреждане на секторите върху пистата, което се определя от фактори на кратност. Например ако първи сектор в последователността си е на мястото, втори е след още 4 сектора, а 3-ти – още 4 сектора след 2-ри. По този начин времето между 2 последователни по номер сектора е достатъчно голямо да сработи цялата електроника и да успеем да прочетем последователни сектори. Така се учеличава ефективността на една писта. Дискът не обработва информацията по сектори, а прочита цялата писта, слага я в кеш памет и чете от там. Няма физическа разлика между секторите, има само логическа. Секторната огранизация позволява поединично записване на секторите, а при тази организация се пише и чете цяла писта. Така времето за достъп става следното: t_{seek} – време за установяване каретата на въответния цилиндър. Каретата не се движи равномерно(ако трябва да мине от най-външна до най-вътрешна писта – в средата на изминатия път се движи с много по-голяма скорост, отколкото в средата на пъта при преминаването от една писта към нейна съседна писта. $t_{seek} = t_{avg}$. t_{avg} – едно средно време(например 9 ms, ако преминаването е от най-крайна до най-крайна писта е приблизително 20ms, а преминаването между 2 съседни – 1 ms). Винаги работи само една глава. Останалите са мъртви. Когато каретата застане на позиция се включва една глава. Файлът се обработва най-добре, ако е разположен в един цилиндър – тогава t_{seek} е 0 между 2 съседни писти, защото се превключва само между главите. $t_{rotation}$ – avg 0,5RPS(RPM/60) – това е времето, което в нужно, за да се докара плочата в положение, в което главата съвпадне с началото на пистата. Най-тежкият случай за t_{seek} е когато главата е малко след началото на пистата и трябва да се направи почти цял оборот, за да се достигне началото, а най-лекият случай е, когато главата е точно в началото. При средно време 0,5RPS и при 7200 об/мин ⇔ 120об/сек => 60об/сек – което е приблизително равно на 15 ms, която е напълно сравнимо с времето за пизициониране. Времето за трансфер е: $t_{transfer}$ = bytes / rate. Rate = sector x track x RPS, където sector е размер на сектора в байтове, track е брой сектори на писта, а RPS е обороти в секунда. $T_{controller}$ – секторът престоява в контролера за да се изчисли ECC, след което

се буферира сектора и по шините се предава към паметта. Контролерът е хардуерно реализиран, не се контролира от софтуера. Обработката на сектора в контролера е много бърза. Ако контролерът работи с опашково дълбоко буфериране има много висене по опашки (FIFO, не е стеково!).

Тактът на процесора е приблизително 1 ns, 1 инструкция е приблизително 20 такта, а времето за достъп до ОП е приблизително 50 ns, което е много по-бързо от например 18,5 ms за диска. Разликата е милиони пъти. Ако се обработва огромен файл има голямо значение как е разположен – ако е разположен вертикално в един цилиндър е страхотно, защото времето за позициониране е 0, тоест няма seek time, $t_{rotation}$ също е много малък – ще дойде веднага началото на пистата.

Компютрите работят добре на стайна температура при включване… Капацитътет на диска зависи от броя на секторите, които са на пистата, или казано по друг начин – плътността на битовете върху пистата. По такъв начин със смяна на потоците от 2 различно заредени домейна се отчитат битовете.

Frequence modulation – най стар метод – 2 магнитчета едно срещу друго бележат началото на бита. Вътре в бита – ако няма промяна на потока – 0 , иначе – 1. Колкото бита има – толкова промени на потока, определящи началото. Например:

Това е числото 747, записано в BCD. Сигналът изглежда така: (картинка ☺)

За всеки бит се използват 4 домейна – неефективен метод, ниска плътност.

Друг метод е MFM – Modified FM – синхронизиращ импулс няма!. Разчита се на това, че дискът се върти строго равномерно и тактовете могат да се генерират в самия контролер (определят от къде започва битът). Пуска се тактов генератор отбелязващ началото на всеки бит – имаме импулс само в средата на такта. Контролера синхронизира вътрешно и знае къде да очаква импулс. (картинка ☺)

Имаме двойно увеличение на скоростта. Очевидно ако има начин да увеличим броя на нулите между единиците този метод ще стане още по-бърз, защото 0 означава, че няма промяна в потока. Шугарт, измисля начин на кодиране за увеличаване на плътността.

Run Length Limited RLL 2.7 (2.7 означава, че между 2 единици може да има от 2 до седем нули – използва се при IDE, SCSI дискове)

Advanced RLL 3.9(от 3 до 9 нули). При всички възможни поредици на битове има само 7 възможности за преобразуване – това е пълно множество.

000	000100
10	0100
010	100100

0010	00100100
11	1000
011	001000
0011	00001000

За горния пример – намираме групировката 011 и първите 3 бита се прекодират с 6, след това намираме 101, взимаме поредицата 10 и я прекодираме в 0100... Така накрая се получава:

001000010001000000001000. При 2.7 имаме 50% по-висок капацитет спрямо MFM, на пътека вместо 17 имаме 26 сектора, при RLL 3.9 – на една пътека има 34 сектора. MFM се използва за твърди дискове, но FM – само за флопита. При RLL имаме 800KB transfer speed вместо 300.

Стандартно във всеки сектор може да се запише 512 байта полезна информация.

Друг начин за увеличаването на плътността е намаляването на ширината на пистите, но това е изядено откъм възможности и вече няма накъде да се развива. По отношение на seek-time-а оптимизация би било да има толкова глави, колкото са пистите, но това е доста скъпо. Има реализации с паралелни глави, което обаче усложнява контролера – нужни са няколко контролера.

Най-важната оптимизация при дисковете се оказва cache-ът, защото дискът обработва заедно с контролера цялата писта наведнъж. В по-сложните системи на по-високо ниво има scheduling – предварително се преглеждат заявките към диска. Ако например се чете така: 1 – 50 – 1 цилиндър, scheduling-ът ще изпълни третата операция преди втората. Дисковите операции се буферират и след това се подреждат по специален начин, така че при движение навътре да се оберат максимално много, след което при движението навън да стане същото, като този процес се повтаря много пъти.

Patterson измисля технологията RAID - масив от нескъпи дискове, в които има излишък. В тези масиви се използва терминът stripping – разпръсване на записите по дискове. Това се оказва много ефективен подход при достъп. При файл всеки запис е 1 сектор и файлът се обработва последователно, ако се постави на 1 диск. Ако, обаче, е разположен на 4 диска, се чете едно нещо от първия, паралелно с него се чете груго от втория и прочитането става по-бързо. Разхвърлянето на данните по дисковете за паралелен достъп разпределя натоварването между дисковете. Освен това излишъкът води до увеличаване на надежността. Има различни RAID варианти:

RAID 1 – пълен mirror (D == C)

Ако водещият диск в даден момент не може да прочете нещо (магнитна или механична повреда), тогава автоматично се работи с втория диск. RAID Controller-ът е по-интелигентен от голям cache буфер, има собствено аварийно захранване, което когато усети, че даден диск е паднал, започва работа с другия. Ако има 3-ти – Hot Spare. Той автоматично отразява втория диск върху третия – т.н. Hot-pluggable – дискът може да се извади без да се спира работата на компютъра. Полезната използваемост е 1/3. За да се увеличи полезната използваемост се въвежда RAID 5,

който е за поне 3 диска. Първият запис на диск 0 + първият запис на диск 1 образуват контролна сума, която е първи запис на диск 2. Този контролен сектор е така изчислен, че който и от треите диска да отпадне, неговата информация се изчислява на базата на останалите. Контролните сектори се въртят на следващ запис(контролният сектор е на диск 0, а останалите са на 1 и 2). Класически е с 4 диска – 75% полезно използване – от 4 диска 3 съдържат полезна информация(може да има и hot spare).

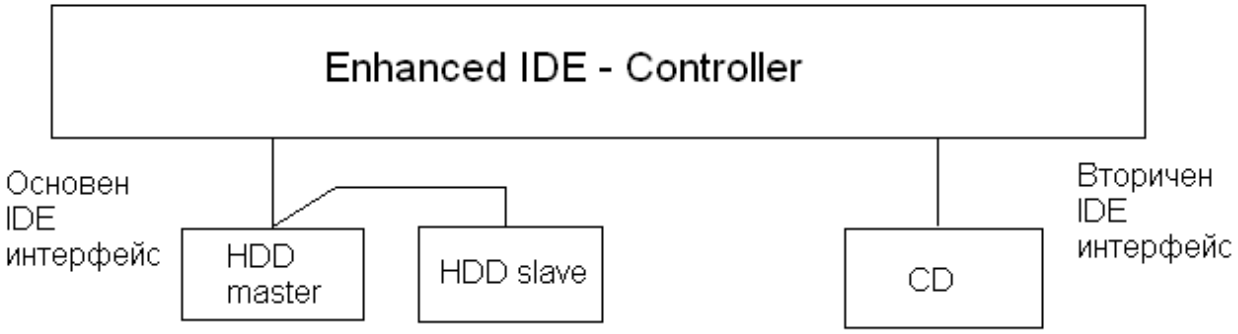
От 80-те години се търсят начини за създаване на малки дискове, които да могат да се побират в настолните компютри. Използва се секторна организация (512 bytes на сектор). Всеки сектор има preamble, номер на сектор и в края има ECC.

Howard Sugar – създава различни технологии и принципи, на базата на които се стиго до малки дискове с огромни капацитети.

В ROM-а е записан BIOS-ът (Basic Input Output System). Още с появата на първите компютри е определен стандарт за периферните устройства – оставени са 4 бита за брой глави, 10 бита за брой цилинтри и 6 бита за брой сектори. Използва се така нареченото CHS адресиране (Cyllinder, Head , Sector). Но поради ограниченията, поставени в ранните стандарти, в даден момент се оказва, че този начин на адресиране е неизползваем (например броят на цилиндрите е > 2000). Затова се въвежда LBA – Logical Block Addressing. Това е логическо адресиране – от началото на диска последователно (от 0 до $2^{24} – 1$)(24 бита размер на полето за номер на сектора). От номера на сектора се изчислява физическият адрес. Контролерът извършва пресмятането на логически адрес към физически.

	BIOS	IDE	LBA	CHS
сектор	512	512	512	512
брой сектори	63	255	63	63
брой цилиндри	1024	65536	1024	1024
брой глави	255	16	255	16
капацитет	7.8GB	127GB	7.8GB	304MB

При най-простите дискове – ISA – Industrial Standard Architecture – на шината се слага контролер и към нея се връзва диск. След това се увеличава полето в BIOS-а, след което се въвеждат IDE – Integrated Disk Electronics.



Тръгнали са от ISA шината, но развиват вътрешни правила по отношение на шината и самите дискове – дисковете са с контролер. По късно се появяват EIDE – EnhancedIDE. Контролерът е мост м/у PCI шината и IDE интерфейса. При Enhanced – от една дънна платка излизат 2 интерфейса – основен и вторичен. На всеки може да се върже по една двойка дискове. Дисковете се връзват верижно, а и логически имат неравностойни позиции – master и slave. Определянето на Master/Slave се прави или с jumper или cable select. Slave терминира сигнала, а Master го препраща към Slave. Логически устройствата са едно зад друго – всичко минава през master-а и ако не е за него се изпраща към Slave-а. Работи се само с единия диск – като се започне работа с него не се работи с другия докато работата е приключи. IDE – разработва се система от команди ATA – Advanced Technological Attachment – система от еднобайтови команди.

NOP-00 20h

Read Sector 21h

Writhe Sector 30h

Които приемат параметри

SC Sector Count

SN Sector Number

CN Cylinder Number

DH Drive Head

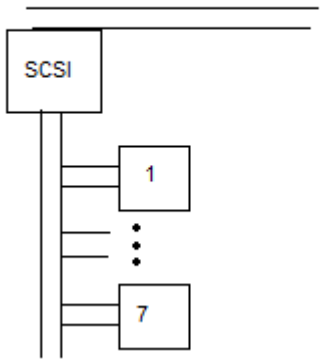
Има команди, с които се дават тези параметри и дискът изпълнява (параметрите се зареждат в регистри). SATA = Serial ATA.

Процесор – шина – контролер – диск – това е пътят на информацията. На какъв принцип се извършва I/O контролът в диска? – PIO – Polling I/O. Процесорът управлява вход-изхода. С увеличаване скоростта на процедора и шините PIO-то придобива различни mode-ове(0 – 4) – достигащи до 16.6 MB/s. До скоро се е препоръчвал PIO, защото няма асинхронни събития, които да усложнят OS и е просто (има по-малко грешки). Direct Memoty Access – специален контролер, на който се казва да извърши обмена, като му се дават данните и процесорът не прави нищо – практически се изключва, затова няма разлика дали се работи с PIO или с DMA.

Преди е нямало смисъл от DMA, но сега се въща като DMA – по-бързо предаване от буфера на дисковия контролер до оперативната памет. Дискът може да работи в PIO и DMA режим.

PIO	0	1	2	3	4
MB/S	3.33	5.22	8.33	11.11	16.6
Ultra DMA		0	1	2	
MB/S		16.6	25	33.33	

IDE – евтините дискове – практически компютърът работи с 1 диск. Sugar е създал система Small Computer System Interface (SCSI). Има SCSI adapter и мост към PCI шината:

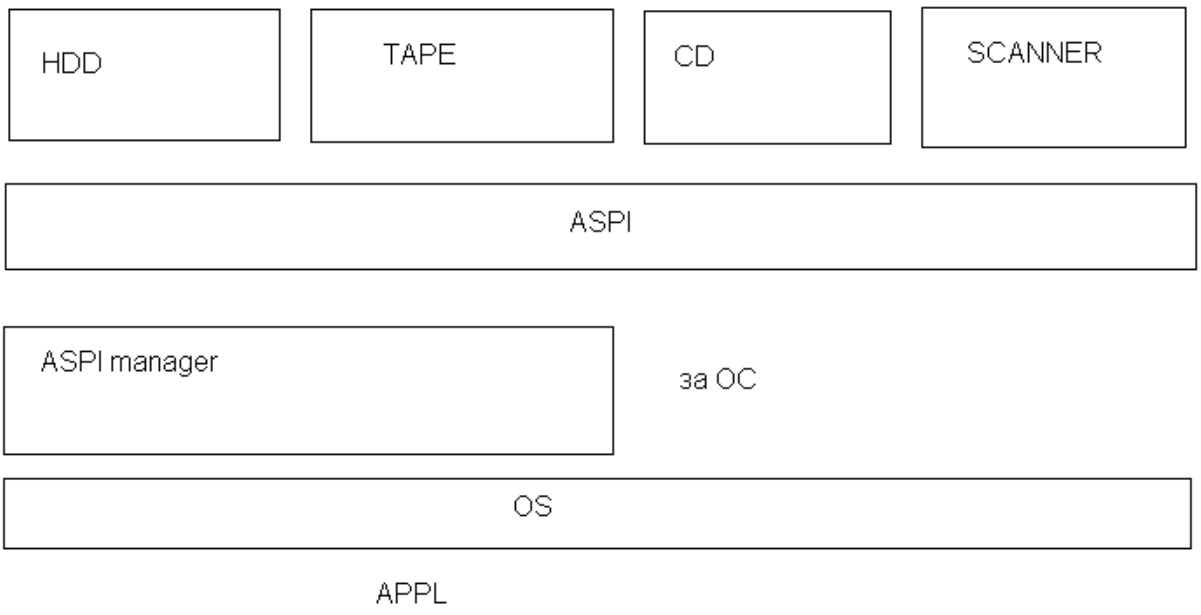


Разделя се входът и изходът на транзакции (първо се пуска на 1, после на 2, ... работи се относително паралелно с устройствата. Тази магистрала се оказва доста хубава. Чисто физически не е различна от другите, но електрониката е друга и скоростите на обмен са други, като цяло е по-бърза и по-надеждна. Стандартно – 8 паралелни линии за данни (5 MB/s). Кабел от тип A – 50 изводен куплунг – четвъртит. След това става wide, 68 извода, закръглен. Wide ultra SCSI2 – 16 bit – 80MB/s – сваля се нивото на сигнала и се въвежда малко по-друг сигнал. Има проблеми със съвместимостта, SCSI – основно стандарт за сървъри, защото не е желателна употребата на SATA за сървъри. Има и 32-битов, но има проблем с кабела. Общата дължина на кабела е 6 метра стандартно, а тези с ниско ниво на сигнала са по 12 метра.

	SCSI standard	fast	ultra	кабел	изводи
8 bit	5MB/S	10 MB/S	20 MB/S	A	50
16 bit	16 MB/S	20 MB/S	40 MB/S	P	68
32 bit	20 MB/S	40 MB/S	80 MB/S	P+Q	110

Разделя се входът и изходът на транзакции (първо се пуска на 1, после на 2, ... работи се относително паралелно с устройствата. Тази магистрала се оказва доста хубава. Чисто

физически не е различна от другите, но електрониката е друга и скоростите на обмен са други, като цяло е по-бърза и по-надеждна. Стандартно – 8 паралелни линии за данни (5 MB/s). Кабел от тип A – 50 изводен куплунг – четвъртит. След това става wide, 68 извода, закръглен. Wide ultra SCSI2 – 16 bit – 80MB/s – сваля се нивото на сигнала и се въвежда малко по-друг сигнал. Има проблеми със съвместимостта, SCSI – основно стандарт за сървъри, защото не е желателна употребата на SATA за сървъри. Има и 32-битош, но има проблем с кабела. Общата дължина на кабела е 6 метра стандартно, а тези с ниско ниво на сигнала са по 12 метра. SCSI интерфейсът е много добре специфициран от IEEE. Опира се до това как ще се използват SCSI устройствата. Следната архитектура:



Общ програмен интерфейс ASPI = Advance SCSI Programming Interface, на фирмата ATAPI. Програма, която се вкарва заедно с OS-а, която разпознава контролерите е OS независима. За OS-а има специфичен manager. ATAPI – няколко интерфейса за свързване на ленти, ...

ASPI интерфейс за облекчениена създателите на OS.

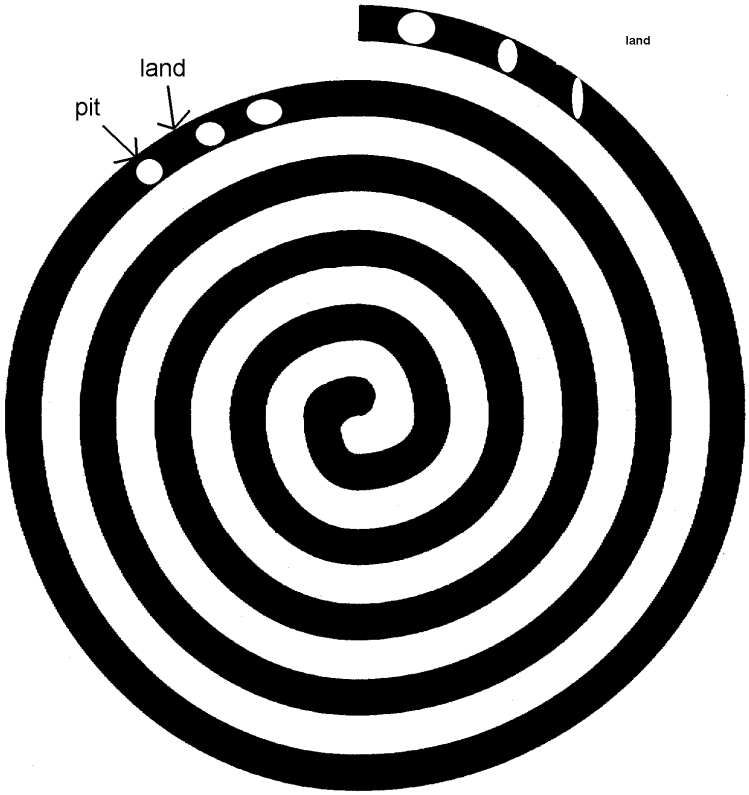
21. Външни запомнящи устройства на CD – диск и DVD – диск.

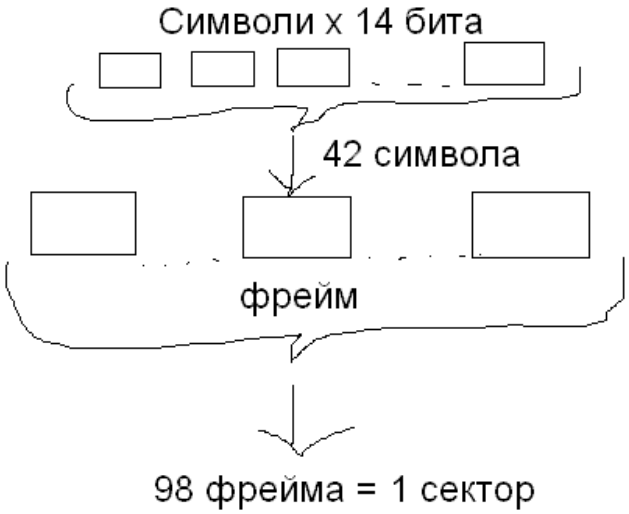
Физическа организация на данните

Disk – магнитен диск, disc – оптичен диск. CD – преди 30 години – Phillips, Sharp, Sony са целели да заменят грамофнната плоча, като измислят нова технология. Плочите са просто устроени: изрязана е спирала и иглата на грамофона се движи по канавката, от което получава трептения, които се преобразуват в аналогови електрични импулси и се изпращат към говорителя, който генерира съответните звуци от музиката. Има обаче технически недостатък: механичните игли трият в плочата и я повреждат при многократно използване (над 100 пъти). Нужна е по-добра

технология, която обаче да се стандартизира, за да се постигне масовост на разпространението и да се свалят цените. Така се ражда CD-то.

Извличането на информацията от CD-то се извършва от лазер и фотодиод. Подложката е с фиксиран диаметър 120мм, дебелина 12 мм, дупка 15мм, направена е от поликарбонатна пластмаса. При производството в тази пластмаса се издълбават малки дупчици с мощен лазер, които са подредени като много дълга спирала, започваща от най-вътреяната част и стигаща до най-външната. Дупките са 0.8микрометра. Дължината на вълната на инфрачервения лазер е 0.78мм. Там се излива смола, която потъва в дупките и върху тази матрица се нанася тънък светлоотразителен слой. Спиралата, в/у която е информациятасе състои от равнити - land и pit-падини. Спиралообразните пътеки от равнини и падини остават завинаги. Когато се чете, четящата глава се движи отвътре навън.Спиралата е дълга 5,6 км. Върти се със 120 см/сек по главата. Тук линейнта скорост за разлика от магнитните дискове е контранта (22188 оборота). За да се поддържа скорост трябва устройството да си променя оборотите. Във вътрешнтата част скоростта е 530 об/мин, вън е 200об/мин. Основна скорост – едно CD свири 74мин. Информацията се взима като в главата има малък лазер с точно определен лъч – 0.76микрометра дължина на вълната. Лъчът се отразява, връща се, минава през леща и попада във фотодетектор, който усеща силата на светлината. Където удари падина (¼ от дълвината на вълната е дълбока – става половин дължина на вълната, получава се интерференция – отразента светлина е по-слаба – следователно има тъмни и светли петна). Аналоговата информация е във вид на битове. С тази скорост успява в реален режим да възстанови музиката при 1x (основна скорост). На времето не е мислено за никакви компютри – единствената цел е била да се продава музика. В последствие компютърната индустрия решава, че това става за комппютрите. Проблемът е, че вече има създаден начин за работа с тях и няма как да се промени стандарта, защото ще се вдигне цената. Постепенно комютрите се пригаждат по следния начин. Съвкупност от 14 бита образува 1 символ, от които половината са контролни. 42 символа правят един фрейм, 98 фрейма правят сектор на CD-то. Той има следната структура.





1x = 75 сектора/сек., 75 сектора/сек. * 74 мин = 650 MB. Дискът се получава с капацитет 650MB. 27% полезна използваемост, защото останалите битове са контролни. Това се оказва много бавно за PC-тата – прекалино ниски са оборотите. По тази причина производителите започват да качват оборотите(2x, 4x, ..., 32x, 48x). Времето за обработка на диска намалява – върти се по-бързо, лин. скоростта се увеличава приблизително 50 пъти, но дължината на спиралата е същата => същото количество памет.

Появяват се втори тип дискове, които са по-плътни, имат повече капацитет (800MB). Създават се CD-та за еднократен запис – CD-R = CD-Recordable, на които се пише един път, след това може да се чете произволен брой пъти. Постепенно се заменят от flash памети. Параметри на подложката: 120мм, височина 1.2мм, 15мм дупка и 0.8 микрометра дупки. Подложката е поликарбонатна, като най-отгоре има защитен лаков слой, върху който е залепен етикетът. Под него има тънък отражаващ слой, който е златен или сребърен, има 1 слой оцветител, по-дебел от отражателя(на цвят е цианин (зелен), калоцианин(жълт) – специални оцветители, които отдавна се използват във фотографията и имат полимерен характер. При записване лазерът излъчва много мощна инфрачервена светлина и той се движи по спиралата(виртуална спирала, описва се от главичката на лазера). Там където има мощен импулс се отделя топлина в оцветителя и той полимеризира и потъмнява, а където не удари лазерът, остава същия свят. По този начин става записването – отделя се топлина и се загрява подложката. Процесът е химически и топлинен. Когато се чете от диск, лазерът е с мощност само 0,5 mW – 20 пъти по-малко. Четенето става така: лазерът отива където трябва да се чете, пречупва се и отива във фотодетектор – според това малко или много светлина има се определя 1 или 0. Всичко е същото, освен начинът, по който се разпознава информацията – тъмни и светли участъци, а не падини и равнини. Четящото устройство е едно и също. Записващото е различно. Оцветителят както полимеризира, така и деполимеризира – с времето може да се преебе. Самият процес на полимеризация изисква време – колко по-бързо може да топли лазерът, толкова по-добре. X4 е най-добрата скорост на записване, но би могъл да се направи компромис със x8. При записване с големи скорости, лазерът има по-малко време да затопли оцветителя и съответно разтапянето става по-некачествено и би могло да деполимеризира за по-кратко време.

DVD

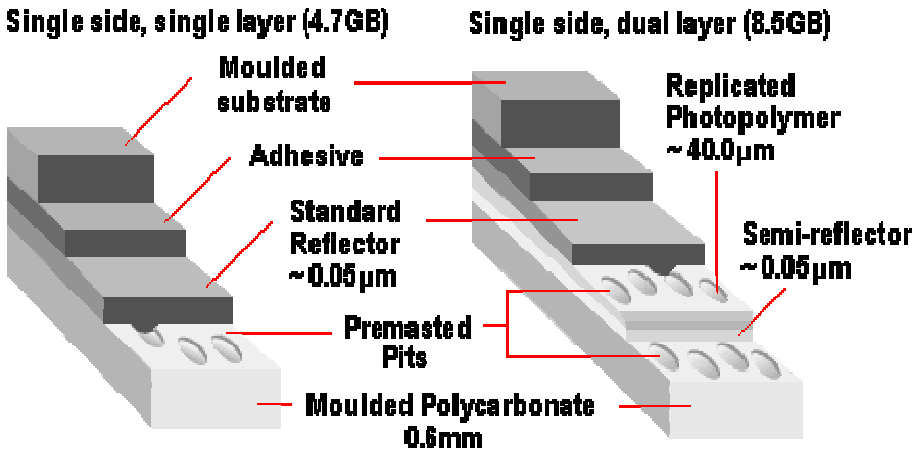
	CD	DVD
pit	0.8 microm	0.4 microm
Спирала стъпка	1.6 microm	0.74 microm
лазер	0.78 microm	0.65 microm
Капацитет	650 MB	4.7 GB

CD-R – при писане, записът се започва от началото и се пише докъдето стигне по спиралата. Теоретично може по-късно да се дозаписва. По-прост вариант е първо да се създаде image на hard-disk-а и този образ да се запише върху CD-то. Върху стандартен CD може да се записва само веднъж с матрицата и не може да се дозаписва. А CD-R-ът може да се пише в повече от 1 сесия. Проблем е надсекторната организация. Дискът е разделен на volumes, като всеки volume е записан в една сесия. Всеки том има главен каталог (VTOC = Volume Table of Contents), който описва съдържанието на volume-а. Принципно VTOC-а е в началото на диска, но това създава проблеми при многосесийното записване: при първа сесия се записва и при следваща трябва да се промени, което е невъзможно. Затова се променя стандарта на VTOC-а, т.е. прави се така, че да допуска продължение. Тоест в края на всяко парче от него има указател към следващото парче. За съжаление аудиосистемите не четат многосесиен запис.

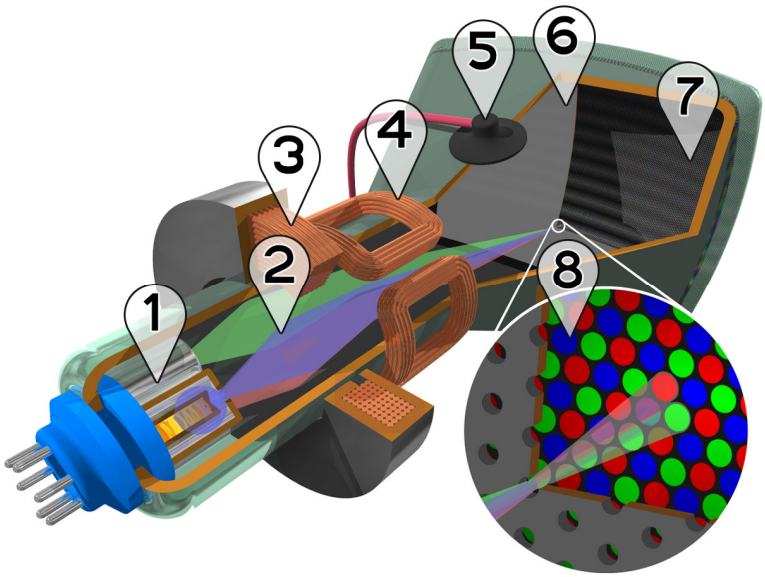
Има много напъни да се увеличи капацитетът на дисковете, затова се създава DVD. При него имаме по-тесни дупчици, стъпката на спиралата е по-малка, лазерът се сменя с по-късовълнов, но се използват 2 лазера. В резултат на оптимизациите капацитетът се качва до 4,7 гигабайта. Дори този капацитет изглежда недостатъчен, затова се слагат 4 спирали – по 2 от всяка страна. От всяка страна има вътрешен и външен диск. Под вътрешната има пълен отражател. Вътрешният и външният слой са отделени от полуотражаващ алуминиев слой. С едно и също въртене се четат всички 4 спирали. Естествено има 4 лазера.

Формати:

Едностранен еднослоен	4,7 GB
Едностранен двуслоен	8,5 GB
Двустранен еднослоен	9,4 GB
Двустранен двуслоен	17 GB



22.Принцип на работа на монитора и графичния контролер



1-катод

2-Лъч

3-Вертикални развивки

4-Хоризонтални развивки

5-Анод

6-Маска за разделяне на лъча на RGB

7-Луминатор

Мониторът е съществено устройство за комуникация с компютъра, типично изходящо устройсво. Първоначално са тръгнали от телевизора, но телевизорите са със значително по-малка разделителна сподобност. CRT – електронно-лъчева тръба (Cathode Ray Tube) – стъклена вакуумна колба, в дъното на която има котод, който излъчва поток електрони, който първо минава през магнитни диафрагми и се формира сноп, след това вертикални и хоризонтални плочи. В зависимост от потенциала между двете плочи се насочва лъчът (електростатично отклонение на лъча). На пътя на снопа има метална решетка с дупчици, която допълнително фокусира снопа и той удря задната част на едрана, която е покрита с луминофор. Той се състои от зърна с подходяща големина. Калибрирана големина, така че снопът удря точно едно зърно и при удара то свети. В тръбата е вакуум иначе би могло да се предизвика светене вътре. Има педиод на послесветене – не светва и не угасва моментално зърното. С тази система изображението е растерно – на хоризонталната и вертикалната плоча се подават 2 трионовидни напрежения. В резултат на това лъчът описва растера. Прав ход – равномерна хоризонтална линия (равномерно се движи снопът, след това бързо връщане, но 1 ред надолу, накрая имаме обратен ход. Има устройство, което спира обратния лъч. Изображението е двуцветно – светещи (зелени) и тъмни (черни) точки. За цвят – има по 3 зърна във вид на триъгълник, R, G, B. Художниците имат друга основна цветова система. Тези 3 зърна образуват 1 пиксел. За да свети

се бомбардира отделно от 3 снопа. В зависимост от интензивността на елетрона съответното зърно свети различно – по-силно или по-слабо. Всеки сноп си следи цвета – 1 сноп за зеленото, 1 за червеното, 1 за синьото. По подходящ начин са разположени, за да може 1 пиксел да се образува от триъгълник .

Основна единица на екраните – главен диагонал. Освен това – разрешаваща способност (800 : 600, 1280 : 1024, 1600 : 1200).

			17”		19”		21”	
X	B	Отн	X	B	X	B	X	B
800	600	1.33	0.43	0.43	0.48	0.48	0.53	0.53
1280	1024	1.25	0.27	0.25	0.30	0.28	0.33	0.31
1600	1200	1.33	0.22	0.22	0.24	0.24	0.27	0.27

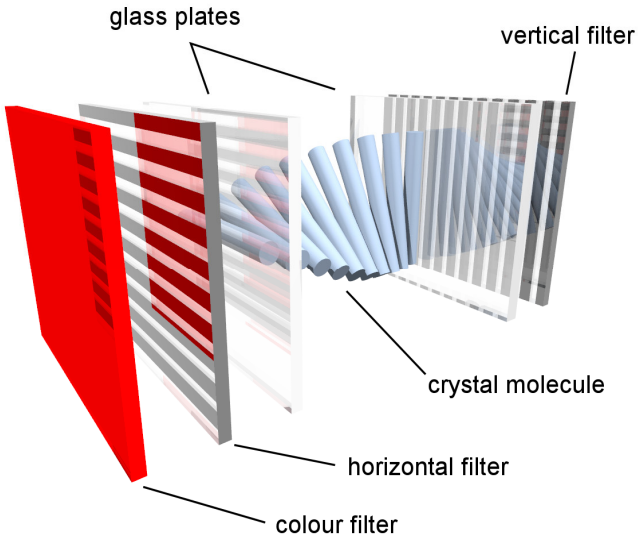
Зърното трябва да се помести в малък размер. Размерът на зърната е свързан с размера на екрана. За голяма резолюция трябват и големи екрани. Трите лъча се движат паралелно и всички зърна започват да светят, след което се връща в началото и започва всичко наново. Понеже има послесветене – старото изображение още не е угаснало. Образи с честота < 60Hz започват да трептят за окото. 75Hz – една разумна кадрова честота с прилично качество.

CRT мониторите имат редица недостатъци:

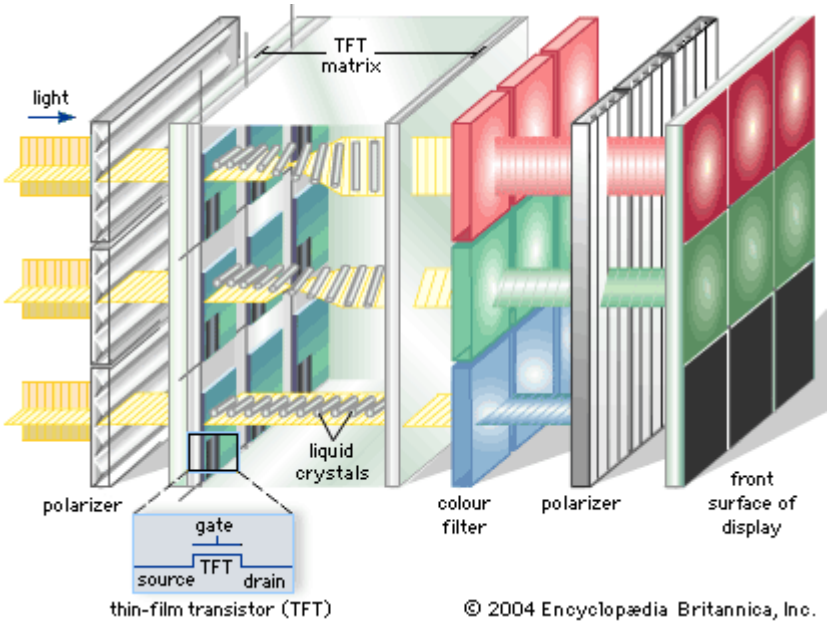
1. Електронно-лъчевата тръба е тежка и изисква доста място
2. Изразходват голямо количество енергия – за един 17” са нужни 150W
3. Специфичната им форма води до не толкова правилното им изобразяване в краищата на екрана
4. Работят с високо напрежение, поради което излъчват рентгенови лъчи
5. Излъчват високо и ниско честотните магнитни полета, които са доказано вредни за хората
6. технологията на сканиране, която те използват прави трептенето на образа е неизбежно, което води до напрежение в очите и умора
7. тяхната възприемчивост откъм електромагнитни полета ги прави уязвими във военни условия

Тези недостатъци дават повод да се търсят други решения и се преминава към LCD – технологията.

LCD – течен кристал. Течният кристал е оранически молекули, които се местят, движат се като в течност. Те са открити през 1888 от Райнцер – австрийски бутаник . 1960 година започват да се използват по предназначение. Могат да се местят, имат отношение към поляризираната светлина – молекулата може да спре или пропусне подходящо поляризирана светлина. Един екран на течни кристали се състои от източник на светлина, заден полароид, който е поляризиран в едно направление, преден полароид - на 90 градуса от от задния.



Изобразяване на субпиксел



Нищо не излиза отпред – двата полароида се ликвидират. Течният кристал е между пластините – те са съответно с хоризонтални и вертикални жлепове, които така ориентират молекулите на кристала, че отпред е тъмно. Когато се подаде напрежение, диполите молекули се влияят от полето и се местят в течността – изкарват се от статичното си равновесие, вече светлината минава с друг поляритет към предния полароид и той започва да пропуска повече или по-малко светлина – на екрана се получават по-слетли и по-тъмни точки. Електродите – преден и заден – като решетка – паралелни проводници на електродите – от едната страна 1280 , 1024 от другата – получава се матрица 1280x1024. Тя е доста евтина и вече не сеизползва. При TFT има активна матрица – там всеки електрод се командва отделно – има над 1 милион електрода и всеки си командва неговата позиция. Използва се TFT(Thin Film Transistor — тънкослоен транзистор) за управление на всяка точка от изображението Ако някой пиксел изгори – там се появява завинаги черна точка. При CRT – броят на точките зависи от зърната, а при TFT матрица е добре да се работи с толкова пиксела, колкото са електродите (има си препоръчителна разделителна способност, която е точно броят пиксели).

При CRT мониторите има видеоконтролер, който подава сигнал в синхрон с движението на лъча. Сигналът определя с каква интензивност да се светне съответният цвят. Лъчът се движи стандартно и безспирно. В точния момент се дават съответните нива на трите цвята – аналоговата стойност на напрежението. Информацията за цвета е цифрова. Постоянното преобразуване се прави от ЦАП (Цифрово-аналогов преобразувател) – подава се цифрова стойност и се генерира напрежение (DAC – Digital Analog Converter). Има палитра за цветовете – по вътрешна шина се подават цветовете до палитрата, която ги разделя на 3 канала. Има видео процесор, който е за обработки върху бидеоизображението. Има също и шинен интерфейс, който служи за паметта. Устройството за синхронизация знае кой пиксел да прожектира на екрана. Всеки кадър се пази във видеопаметта, като последователност от информация за пикселите. Въпрос: колко информация има за всеки пиксел? Има няколко варианта:

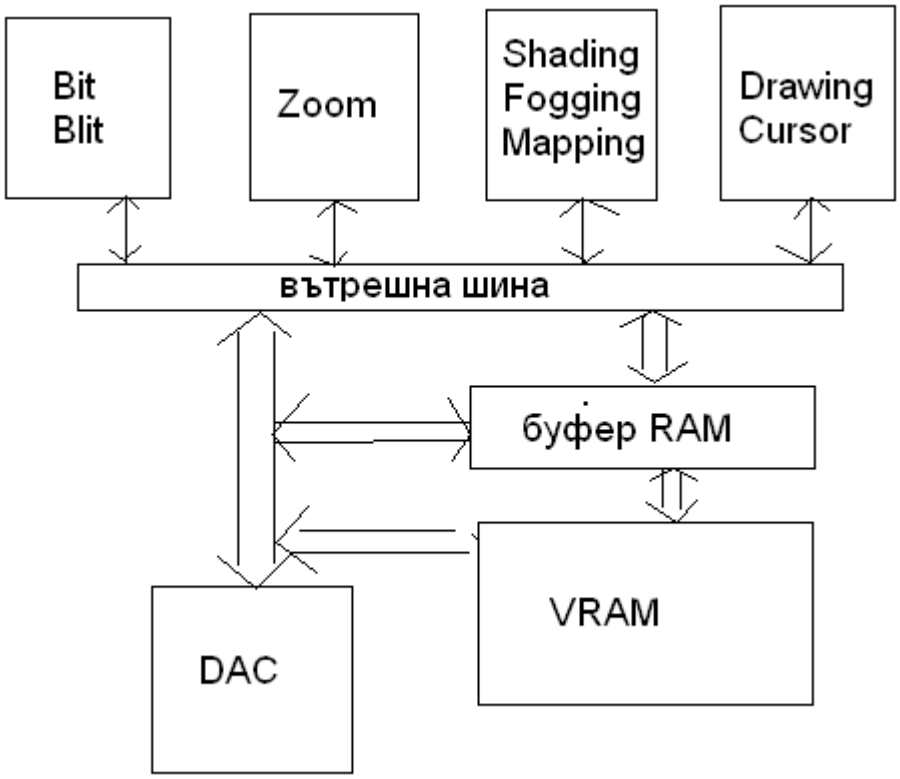
1. 3 байта – за всеки цвят по 8 бита. Всеки основен цвят има 256 нива на интензитет, което прави общо 16 милиона комбинации. Нарича се още True Color.
2. 2 байта – 5 бита за червен, 6 бита за зелен и 5 бита за син – нарича се още high color. Възможните цветове са 65535.
3. 1 байт – само 256 възможни цвята. При тази схема, когато 1 байт отиде в таблицата на палитрата, тя вади на изхода си 3 байта – по един за всеки основен цвят. Трудно се сменят палитрите в рамките на един кадър. Недостатък е, че палитрите са статични.

При резолюция 1280x1024, 75Hz,1 байт => 93,75MB/s, при 2 байта => 187.50MB/s, при 3 байта => 281.25 MB/s – много мощен поток. Този поток се осигурява от вътрешните шини. При 1 байт един кадър е 1.25MB, при 2 – 2.50, а при 3 – 3.75. Проблем се оказва темпото, с което данните трябва да постъпят във видеопаметта от оперативната памет. Връзката с видеопаметта: отделени са адреси от A000 до BFFF – 128 KB – за видео лупа. Програмата пише само в тези адреси – през тази лупа по-бързо се извеждат данните във видеопаметта (може и самият този участък да е видеопаметта). Кадърът първо се изчислява, след това се наглася в ОП, след това се записва във видеопаметта, след това постъпва в палитрата и накрая се рисува. Нужна е огромна скорост по шината, за да се захранва видеопаметта. Затова при новите компютри се въвежда и отделен видеопроцесор.

Тъй като изчисляването на всеки нов кадър в компютъра при бърза смяна на кадрите е невъзможно за прекарване със скорост в паметта на виде контролера. За работа със бързо движещи се изображения се разработват така наречените видео ускорители. Те се базират на това, че има стандартни, често срещани промени, които са общи за движението. Те се изнасят като отделни функции и се реализират хардуернно. Съответен хардуерен блок изпълнява указана функция върху указана част. Основото за графичните ускорители е че имат блокове, които изпълнява добре дефинирани указани функции с дадени параметри. Командването трябва да познава структурата на графичния ускорител – как се извикват и комбинират тези функции. Управлението не е автоматично. Трябва да се прави от програми които знаят какво правят. Не е достатчно да има графичен ускорител, а и софтуер, който да познава и използва неговата функционалнотст. Много от функциите са стандартни. Програмата за управление се нарича драйвер. Работи на ниско ниво през специален контролер и е тясно свързана с операционната система. Много е трудно да се почви такава програма извън архитектурата на операционната система.

Програмите, които управляват графичните ускорители са сложни, поради което тяхното появаване се бави. Първо на пазара излиза графичния ускорител и по-късно поддържащия софтвер. Тъй като това струва много скъпо създателите на операционни системи не могат да поддържат много разработчици на графични ускоритили. Microsoft има специална система за работа с графичните ускорители наречена DirectX., която се използва от производителите. Това спъва развитието на графични възможности при други операционни системи. Linux се развива по-бавно в това отношение и затова под Linux вървят много малко игри.

Има няколко основни групи от функции които се пресмятат в графичните ускорители:



При двуизмерна графика

- преместване на прозорци и блокове от пиксели – Bit Blit, графичния ускорител изчислява коя част от екрана закрива и открива в този случай.
- хардуерен куроср – изобразяването на курсора на мишката спрямо неговите координани не става със смяна на целия кадър. На графичния ускорител се дава новата координата на курсора. Той го слага на новот място и го маха от старото.
- При един кадър от над един милион точки в повечето случаи само някои от тях се променят. В изображението има чертаене на линии, кръгове и полигони – определени геометрични обекти. Те могат да се опишат с функциите drawing
- zoom е свиване или развъване на определена област, тоест размножаване на пиксели в определена околност. Това също подлежи на изчислени. Мащабирането е дискретно и на стъпки.

3D ускорителите имат фуккции за работа с триизмерни изображения. Там се добавят много нови и сложни функции. Основните са shading, fogging и mapping.

- Shading функциите придават на обекта сенки и полусенки и му дават триизмерен вид
- Fogging представлява замъгляване на контурите при отдалечаване.
- Mapping е запълване на повърхнина с обекти които се наричат текстури. Пова придава по реалистично движение. На функцията се подава текстурата и за коя част от фигурата става дума.

Формални параметри свързани с характеристиките на монитора

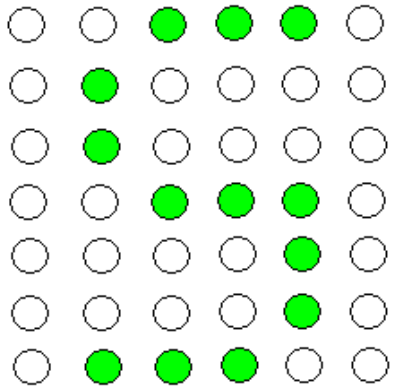
широчина = хоризонтален размер* размер точка = 1024*0.31 = 3.17 (не знам дали тук трябва да е 1280)

редова честота = брой редове*честота = 1024*75 = 76KHz

честотна лента = разделителна способност* редова честота = 1280*1024*75 = 98.3

тактова честота= редова честата * хоризонтална разделителна способност = 1280*1024*75

Изобразяване на символи на екрана. То е започнало при най- старите мониторир които изобразявали букви и цифри. Изображението не се е подавало като съвкупност от точки а като поредица от кодове на символи, разположени на редове (25*80=2000 символа на екран). Те се подавали като кадър от 2000B в ASCII код. Изобразяването ставало с помощта на вграден в монитора набор от фонтове. Фонтовете се представяли с таблица с толкова елементи колкото са кодовете на валидните символи и за всеки от символите има двумерна матрица от точки с определен размер

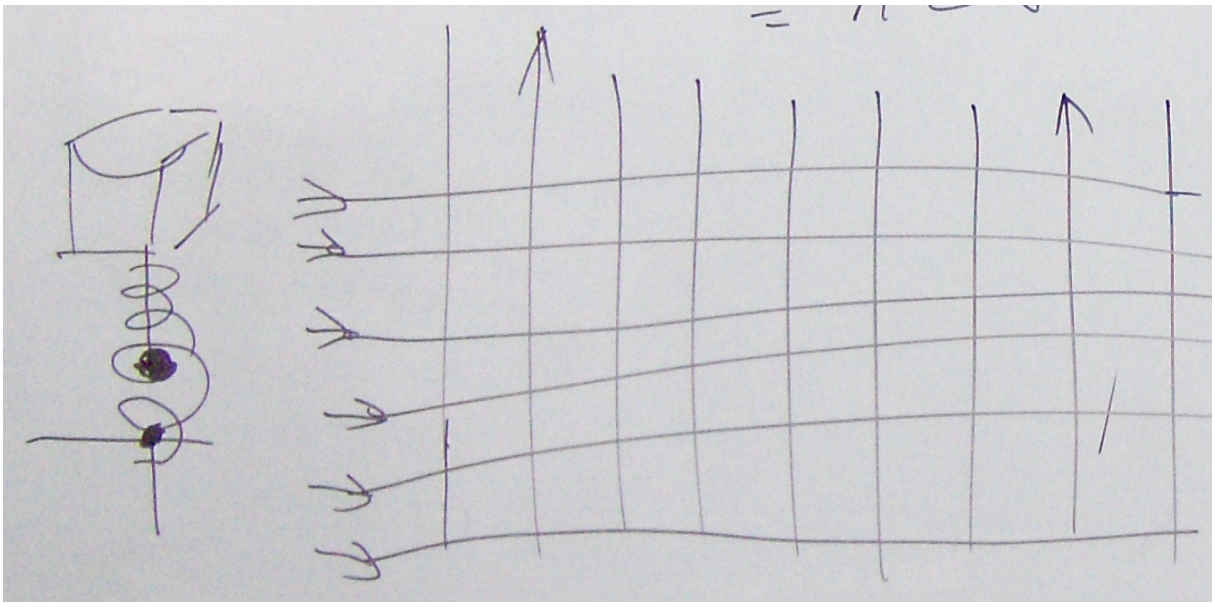


В най примитивния си вид тази таблица съдържа единици и нули, и точките, които се оцветяват са 1ци, а в които не светят са 0. Единиците са зелени,а нулите са черни. При цветното изображение 1 определя цжета на символа а 0 – цвета на фона. Тоест при цжетното изобразяване се дават още два цвата.

Фонтовите матрици са с различни размери, но помежду си символите са с еднакви размери. Този начин се изчертава като серия от точки. В крайна сметка се преобразуват кодове като последователност от точки. Обемът на данните, които се трансферират е много малък. Фонтовите матрици се изчисляват не само от графичния ускорител. В операцинните системи масово се използват за формиране на изображение, което се показвана екрана. С фонтовите матрици се праваят шрифтове. Те подлежат на мащабиране, но това се изчислява. Техниката с фонтовете е основен начин за изобразяване на символи. Множество фонтови матрици образуват шрифтове.

В узкорителя има VRAM DDR памет, която е едновходова. Такава е обикновената памет. В нея или се пише или се чете, което създава проблем при комуникацията със ЦАП(цифрово аналогов преобразувател). Добре е от друг изход да излизат данните и да отиват в ЦАП. Ако е проста паметта всичко трябва да мине през вътрешна шина.

23. Принцип на работа на клавиатурата



Клавиатурата е устройство което се използва за въвеждане на символи в компютъра. Класическата клавиатура се състои от бутони. Устройството на един клавиш е следното. На една пружина са части от един контакт. Пружината държи контакта нормално отворен(механичен нормално отворен контакт). Когато се натисне клавиша контакта се затваря. Клавишите са подредени в наборно поле – правоъгълна матрица. От един ред клавиши долните контакти са свързани с редови проводник, а горните контакти са свързани с проводник по колоните. Това е

класическото наборно поле. Периодично на всеки ред се пуска електрически импулс и се следи дали той ще се почви на някои от стълбовете. Ако се появи се определя позицията на затворения контакт. Поради периодичното сканиране, което не е просто, се вкарва малко микропроцесорно управление от малък микропроцесор с RAM памет, в която се пази състоянието в наборното поле. Ако при две последователни сканирания на един клавиш се установи че контакта му си е сменил състоянието то този клавиш се отчита като съответно натиснат или отпуснат. Следи се и докога ще стои натиснат. След две последователни сканирания се указва че контакта е например отворен и се задава ново състояние – отпуснат. Нормалното състояние на един клавиш е отпуснат. Сканиранията се правят много бързо в сравнение с движението и затова се отчита движение при две последователни сканирания за да се избегне грешка.

Позиционния код определя координата на клавиша. Има система за разпознаване на кодовете. Позиционния код е 7 бита, пораду тива наборното поле не може да е по голямо от 128 клавиша. Стандартното еоколо 101-102 клавиша. Има система за определяне на координатите. Микропроцесорното управление държи в паметта състоянието по позиции и когато се установи изменение се генерира код 7 бита казващ позицията и 1 старши бит указващ състоянието – дали е натиснат или отпуснат. Този 8 битов код е позиционнен. Няма нищо общо с ASCII кода и кода на таблиците. Той постъпва в опашков буфер. кодовете се подреждат така че първия влязъл е първия излязъл. Опашката има размер около 20-40 кода. Когато е препълнен буфера се издава звук за грешка.

Стандартната връзка на клавиатурата с контролера е със 4 проводни кабела – един за захранване +5V, земя 0V, проводник за данни и синхро импулс. Предаването става последователно бит след ит и на нива. Няма особена кодировка на информацията но четенето зависи от синхросигнала.

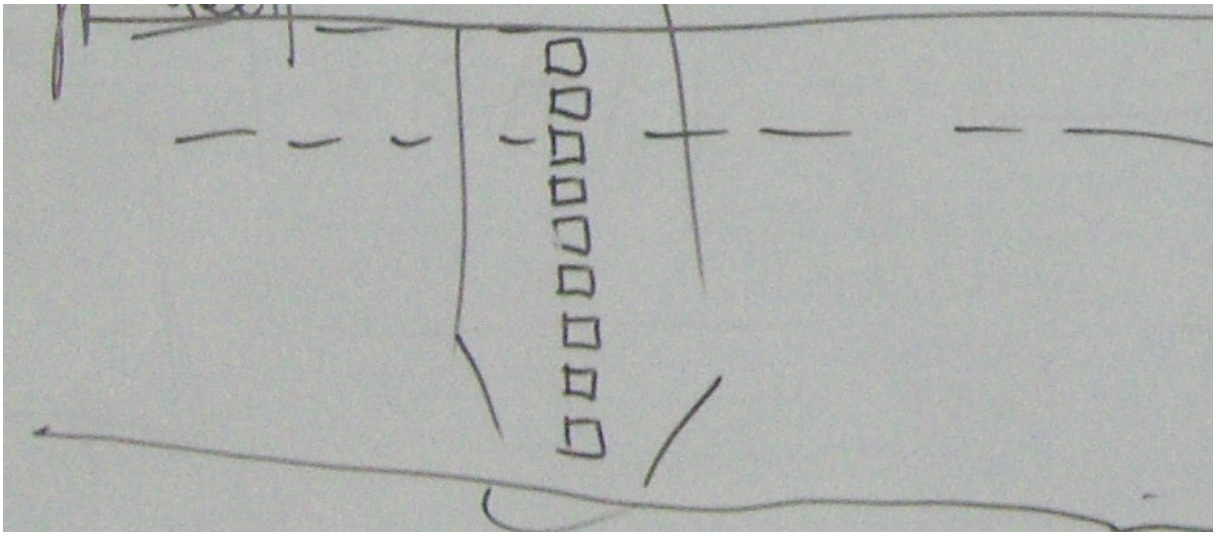
Връзката между контролера и клавиатурата е двупосочна. Подават се команди например reset на клавиатурата. Контролера на клавиатурата се състои от три регистъра по 1B – контролен регистър, регистър за състояние и регистър за данни.

Тези регистри са на определен фиксиран порт за клавиатурата – входно-изходен адрес(61,62,63).

Когато контролера е в нормално състояние четене от клавиатурата то кода на натиснатия клавиш влиза в регистъра за данни. Когато го напълни контролера предизвиква прекъсване по линията IRQ1, която е високоприоритетна, и ако не е забранено маскирането става прекъсване, което включва програма за обслужване на клавиатурата – клавиатурен драйвер. Тази програма знае точно номерата на портовете, чете състоянието, вижда че е постъпил нов код, чете го и го прехвърля в опашката. След това клавиатурния драйвер записва команда в регистъра за управление, нулира регистъра за данни и записва в бита че контролера е готов за следващо четене. Вече в процесора клавиатурния драйве обработва постъпилия код в зависимост от състоянието. И от този код прави код, взаимстван от конкретната кодова таблица. Един контролен код има много различни съответсвия спрямо ASCII кода в зависимост от състоянието за горен – долен, валидна кодова таблица. Клавиатурния драйвер е програма която може да се настрои как да интерпретира позиционните кодове. Всички прехвърляния се свеждат до проверки и таблици в клавиатурния драйвер. Той върши основната работа по интерпретация на кода и управлява портовете.

24. Принтери – лазарни, струйни и матрични

Първоначалните принтери са механични. В компютърното печатане това е най-стария начин за известните преди принтери. Првите принтери са барабанни с дълъг барабан колкото един ред и по външната му повърхност са гравирани извъкнали символи, като един ред от барабана е пълен с едни и същи изпъкнали символи. Барабана се върти и застава в дискретна позиция на съответния символ. От началната позиция на барабана стои символа 'A'. В принтера е вкаран един ред от символа, по 160 символа на ред и принтера сканира този ред и вижда на кои позиции има А и докато е на първа позиция се активират електомагнитни чукчета и те изкачат. След това барабана се върти за В и така нататък преминава през всички символи докато напечата един ця ред. Печата се доста бързо, но смяна на шрифта би означавало смяна на барабана. Така да се каже шрифта е много дълбоко хардуерно заложен. Поради проблема със шрифтовете и защото тези принтери са скъпи и много големи и са неудачни за малки персонални компютри.

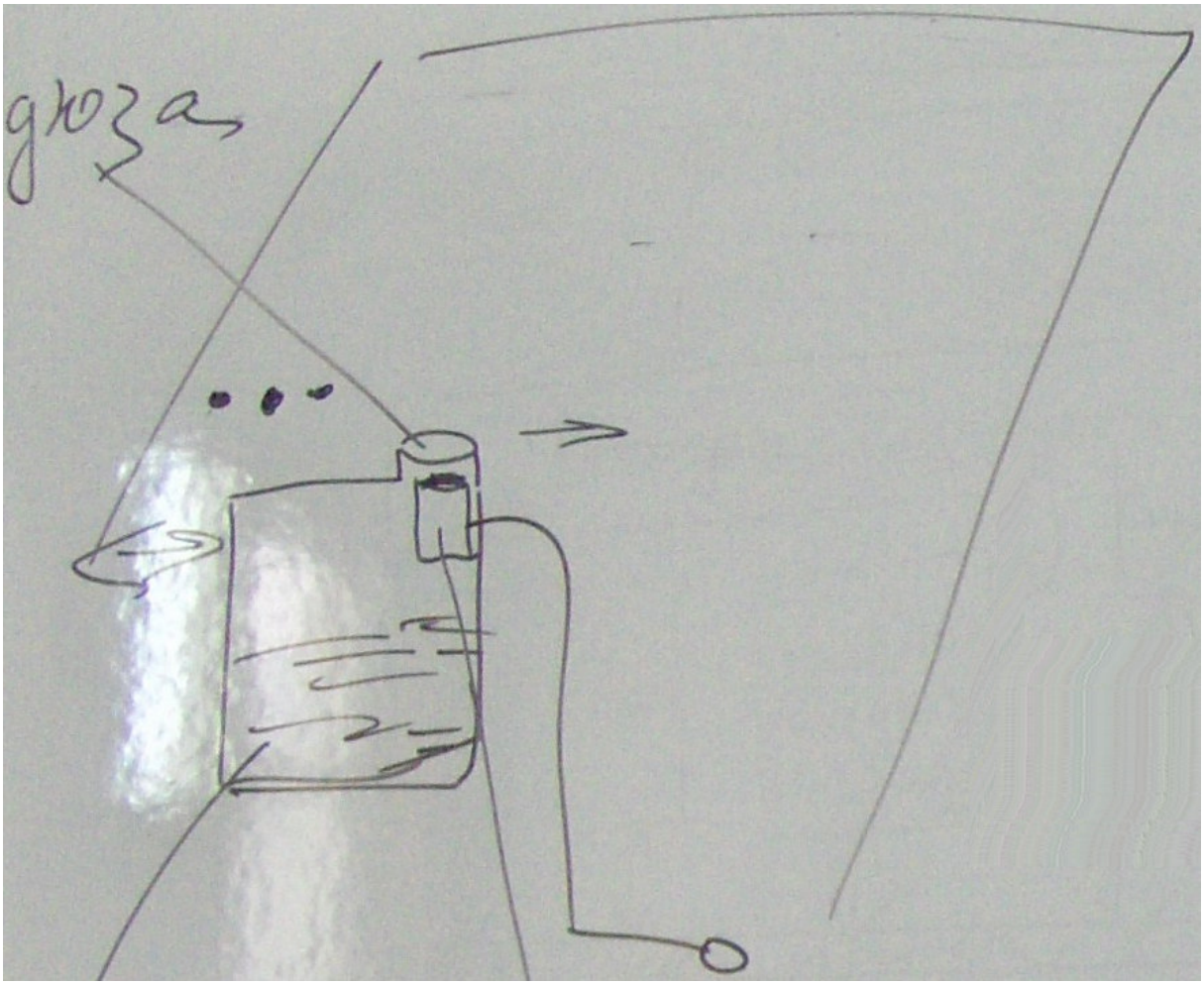


(схема на глава на матричен притнер)

При персоналните компютри се преминава към матрични печатащи устройства при които един стълб от 9 или 24 печатащи игли се движи хоризонтално на листа и при преминаване от единия край до другия тя застава на дискретни позиции наречени стълбове. За едно преминаване печата един ред. При определена позиция на главата тя от кодовете на символа взима ред от ASCII таблицата и търси фонтова матрица за търсения симбвол. На първия стълб от фонтовата матрица където има единици се дава електромагнитен импулси съответните точкисе отпечатват и така докато се стигне до края на реда. След края на реда се извиква нов ред от символи. Може да се печата и отзад напред като логиката за това е проста. Между два реда валяка придвижа хартията във ветикална посока. Матричния принтер е удобен за печат на символи. Може да печата и поредица от точки но ред по ред. Тогава работи само със средната игла и тя се претоварва. Тези принтери печатайки във windows режим се развалят. За символното печатане има заредена фонтова таблица която може да е вградена за принтера или да се сложи допълнително. Тези принтери са удобни за масови обработки. Хартията е непрекъсната и поредово се отпечатва върху нея. Печатането е бавно защото стълбчето се мести в

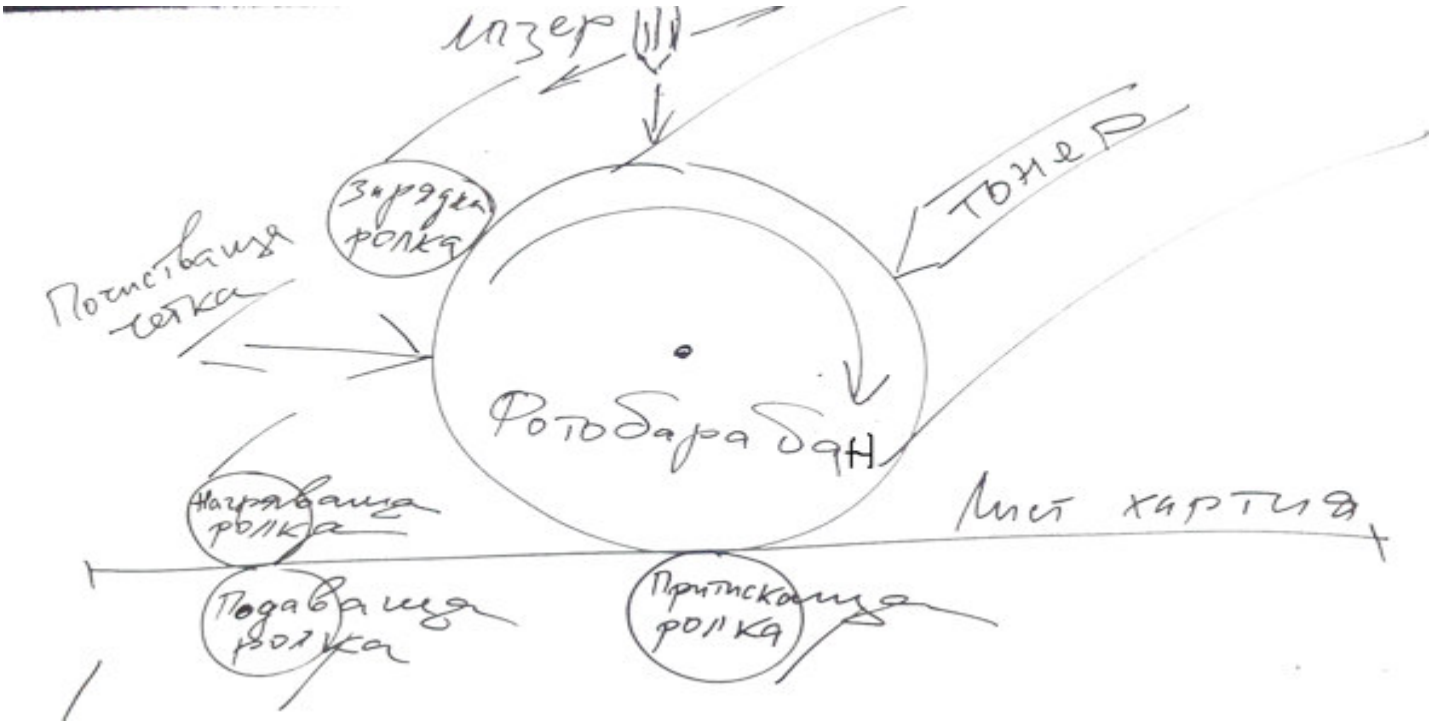
продължение на целия лист . Има към 800-1000 позиции през които трябва да премине. Стандартната фонтова матрица е 9x14. Поради това за по големи обеми се правят много стълбчета, подобно на барабанения. Тук вече има цял ред от много игли. Този принтер печата около 600-800 реда в минута и струва до 20 000 евро. Това са така наречените линейни принтери.

Мастилено-струени принтери



изображението се формира точка по точка. Мастилницата е пълна с мастило. Има вътрешен драйвер. Електроимпулс изхвърля една капка през дюза. Мастилото преминава отляво надясно. Ако изображението е цветно се слагат 3 мастилници една до друга и трябва да плюят трите капки една върху друга. Става бавно, ред по ред. Хартията трябва да не разлива мастилото и то да изсъхва на капка. Мастилото не трябва да засъхва по дюзата. Тези принтери имат рентабилност при цветно печатане.

Лазарен принтер



Състои се от фотобарабан, който се върти. Повърхността минава през почистваща четка, която маха остатъци от предишно печатане. Повърхността минава покрай зареждаща ролка която е електростатична и зарежда повърхността положително. Тя се състои от специално покритие което може да се наелектризира електостатично. Има немагнитен заряд и се зарежда плътно положително. Повърхността идва под един лазар и той се мести по нея. На лазара се подават точките които трябва да се отпечатат. При тъмна точка той разрежда с лъч повърхността, където е бяло прескача. Сивите се разреждфат междинно. Минавайки през лазара повърхността е заредена на участъци и разредена на участъци. Така тя минава през касата с тонера който се състои от положително заредени мастилени частици със слаб заряд. Тези частици полепват по точките които почти нямат заряд, а където заряда е положителен нищо не полепва. Тонера не е течен а от прахови частици. Барабана се върти и тази повърхност идва върху листа хартия и частиците минават от барабана към листа. Минава отново да се почисти и точките за следващия лист се зареждат. Има нагряваща ролка върху която има теглеща ролка. Нагряващата ролка изпича тонера и хартията излиза затоплена. Ако листа се отпечата от другата страна, част от тонера се отлага върху теглещата ролка.

Характерно за лазарния принтер е че печата на дискретен хартиен източник – лист или страница. Цялата информация се вкарва като растерно изображение, не може страницата да спре. Не може изображението да се подаде на участъци