БУРГАСКИ СВОБОДЕН УНИВЕРСИТЕТ

ЦЕНТЪР ПО ИНФОРМАТИКА И ТЕХНИЧЕСКИ НАУКИ

Учебно помагало

за държавен изпит

спец. “ Компютърни системи и технологии”

част 2

теми: 34-35 ; 40-54

автори: проф. Лазаров
доц. Моллова
доц. Гичев

Бургас
2008

Тема 34.**Устройства за въвеждане на информация.** Клавиатури, мишки, скенери. Основен принцип на работа. Свързване към компютъра.

Предназначението на входните устройства е да въвеждат данни в компютъра. Основна тяхна функция е да преобразуват постъпващата в аналогов вид, от различни източници информация, в разбираем за компютъра цифров код.

Клавиатура

Клавиатурата е главното входно устройство на една компютърна система. Тя се използва за въвеждане на кодовете на определен набор от символи.

Основните **видове** клавиатури са:

- 83 - клавишна PC и XT клавиатура (излязла от употреба);
- 84 - клавишна AT клавиатура (излязла от употреба);
- 101 - клавишна разширена (Enhanced) клавиатура;
- 104 - клавишна Windows клавиатура.

Основни компоненти в състава на една клавиатура са *клавишните превключватели***.** Съществуват два типа превключватели:

- Механични
- Капацитивни

Механичните от своя страна се разделят на:

- Чисто механични превключватели
- Механични превключватели с елемент от пенопласт
- Превключватели с гумени шапки

Начин на работа на клавиатурата

Подредените в редове и колони клавишни превключвателите образуват т.нар. *клавиатурна матрица.* При натискането на клавиш се свързва един ред с една колона на клавиатурната матрица. Затварянето на веригата се отчита като прилагане на логическа единица към редовете, след което се преглеждат колоните за наличие на логическа единица. При откриване се отчита пресечна точка на ред и колона, което е затваряне на прекъсвача. Координатите на пресечната точка се трансформират в 7 битов позиционен код. Генерираният код се разширява до 8 битов по следната логика: при натиснат клавиш се добавя 0 и се нарича *код на натискане,* а при отпуснат клавиш се добавя 1 – *код на отпускане.*

Управлението на клавиатурата може да се раздели на две нива:

- вътре в самото периферно устройство - от едночипов микрокомпютър, наричан още микроконтролер или още клавиатурен компютър
- извън периферното устройството- контролер на клавиатурата, разположен на дънната платка

Работата на *микроконтролера* се свежда до изпълнението на пет програми, записани в неговата ROM памет.

- програма за вътрешна диагностика и първоначално зареждане - стартира се при получаване на команда от контролера на клавиатурата. Програмата осигурява извършването на вътрешна диагностика на едночиповия микрокомпютър и проверка на състоянието на клавиатурната матрица. Нормалното й завършване показва, че клавиатурата е в изправност.

- програма, осигуряваща сканиране на клавиатурната матрица - тя се изпълнява циклично от едночиповия микрокомпютър и е основна за всяка клавиатура, при положение, че не са постъпили други команди от клавиатурния контролер.В края на всеки цикъл се определя състоянието на активираните от оператора бутони, както и последователността им

- програма, определяща позиционния код на активирания бутон. По този код се извършва четене в ROM паметта на едночиповия микрокомпютър, от където се извежда действителния код, който трябва да бъде изпратен към компютъра. Този код вече се записва в буфера, разположен в RAM паметта на едночиповия микрокомпютър.

- програма, извършваща информационния обмен с клавиатурния контролер - стартира се при едно от условията: наличие на код за изпращане в буфера на клавиатурата или необходимост от служебен обмен с клавиатурния контролер по негово искане. Осъществява пълния комуникационен протокол с клавиатурния контролер от страна на клавиатурата.

- програма за управление на светлинните индикатори –стартира се при получаване на команда и включва съответния светлинен индикатор в зависимост от съдържанието на байта с опциите в командата.

Контролерът на клавиатурата служи като междинно стъпало в осъществяване на интерфейса на клавиатурата с централния процесор Комуникацията му със системната шина се извършва чрез:

- входен буфер
- изходен буфер
- регистър на състоянията

Основната му функция е следната: получава данни от клавиатурата в последователен формат, проверява ги по четност, преобразува кодовете от последователен в паралелен и предоставя данните за ползване от системата като байт данни в *изходния буфер*.

При зареждане на изходния буфер с данни се изработва сигнал за прекъсване на централния процесор. Необходимата информация за евентуални грешки при приемане на данните от клавиатурата се записва в *регистъра на състоянието*.

Изпратените от централния процесор към клавиатурата данни и команди се записват във *входния буфер* на контролера на клавиатурата. Контролерът преобразува байта данни обратно в последователен формат и го изпраща към клавиатурата, като автоматично вмъква и бит за проверка по четност. Клавиатурата трябва да потвърди всички изпратени към нея данни. Без получено от нея потвърждение контролерът на клавиатурата не може да изпрати следващия байт.

Свързване към компютъра

Клавиатурата се свързва с компютъра с четирипроводен интерфейсен кабел.

- един проводник за предаване на данните
- втори проводник се използва като обратна връзка за информационния сигнал
- трети проводник се използва за синхронизация между клавиатурата и компютъра
- четвърти проводник снабдява клавиатурата с 5 v постоянно напрежение

Съществуват няколко вида конектори за свързване на клавиатурата към компютъра. Най-популярни са:

- 5-пинов DINконектор, използван в повечето PC системи с Baby-AT форм-фактор на дънната платка
- 6-пинов mini-DIN конектор. Последният се използва главно в PS/2 системите и повечето PC-та с LPX ATX, NLX дънни платки.
- интерфейс за клавиатура с универсалната серийна шина USB.

Мишка

Представител е на групата *интерактивни* или още *посочващи устройства.*

Мишката е входно устройство, което служи за бързо предвижване на курсора върху екрана. Обикновено функциите, изпълнявани от мишката, са дублирани и могат да се изпълняват и от клавиатурата.

Мишките могат да се класифицират по четири основни признака:

- технологията, която използват
- броя на бутоните
- начин на свързване с компютъра
- протокол за кодиране на изпращаната информация към компютъра.

Според *технологията*, която използват биват:

- механични
- оптични мишки.

При ***механичната*** мишка има специална част, представляваща въртящо се топче, направено от материал, който се захваща дори за гладки повърхности. В конструкцията има три колелца, които се задвижват от топчето при преместването по някаква повърхност. Две от тези колелца се наблюдават по електронен път. Завъртайки се, те предават степента на това завъртане на компютъра. Двете колелца са перпендикулярни едно на друго, така че едното проследява движението по оста X, а другото - съответно по оста Y. Движението във всяко от четирите направления се оразмерява като стотни от инча и се предава на компютъра под формата на отделен сигнал за всяка отделна стъпка от движението. Третото колелце е просто колелце.

При ***оптичната*** мишка. няма движещи се части. Мишката е снабдена с оптични сензори в долната й част и мрежа, поставена върху специална подложка. При преместване на мишката сензорите засичат линиите на мрежата, разположени на подложката Типичната оптична мишка използва две двойки светодиоди и фотодетектори, които са разположени под прав ъгъл една спрямо друга Всяка от двойките светодиоди и фотодетектори открива движението във всяка посока по осите на мрежата.

Новите оптични технологии не изискват за функционирането на оптичната мишка специална подложка. Оптичния сензор е заменен от CCD (charge couple device) елемент. "Заснетите" отчети се подават към DSP процесор. Той ги анализира, сравнявайки ги с предишното отчитане.

За осигуряване на светлина, необходима за работата на оптичния сензор се използва светодиод или лазерен лъч. Във втория случай се говори за лазерни мишки. При тях се постига по-голяма прецизност за сметка на по-добрата осветеност, което от своя страна позволява на оптичния сензор да приема по-детайлни изображения.

Свързване към компютъра

Съществуват няколко вида конектори за свързване на мишката към компютъра. Най-популярни са:

- стандартен сериен интерфейс(с 9 или 25 извода)
- 6-пинов mini-DIN конектор(PS/2 конектор)
- USB порт

Скенери

Скенерите са автоматични устройства за въвеждане на графична информация.

Съществуват три основни типа скенери:

- ръчни
- настолни
- барабанни

Скенерите работят на принципа на оптичното заснемане на изображението от хартиен носител или фотофилм.

Основният принцип на работа на скенера е:

Изображението се осветява, отразената от него светлина се преобразува от фоточувствителни клетки до напрежение, което се конвертира в цифров (двоичен) вид и се изпраща към компютъра.

В зависимост от вида на оптичната система има три вида скенери:

- с устройства със зарядно пренасяне CCD (Charge Coupled Device).
- с контактен сензор за изображението CIS (Contact Image Sensor)
- с фотоумножителна тръба PMT (Photomultipller Tube)

CCD скенери. Оптичната система система се състои от: източник на светлина

- огледала
- лещи
- CCD елементи

Осветяването на изображението става с флуоресцентна лампа. Отразената светлина се насочва от система огледала, след което се фокусира от леща и достига до CCD елементите, които преобразуват попадналата върху тях светлина в напрежение

Всеки елемент има три филтъра - за червената, за зелената и за синята светлина. Някои от CCD скенерите извършват три отделни сканирания за всеки един от цветовете - червен, зелен и син, в други модели - и трите цвята на един път.

Сложната оптична система от огледала и лещи води до по-големи размери и тегло на скенерите.

Постигат високо качество на сканираното изображение.

CIS скенери. Редица от сензори се намира на няколко милиметра под стъклената повърхност. Осветяването се извършва от ред червени, сини и зелени светодиоди (LEDs), разположени близо един до друг за да образуват бяла светлина. Отразената от изображението светлина попада върху фоточувствителните елементи на сензора. С помощта на филтри светлината се разделя на трите основни цвята - червен, зелен и син. Вграденият в чипа аналого-цифров преобразувател конвертира информацията от сензора в двоичен вид.

Отсъствието на оптична система прави този тип скенери много леки и тънки.

Използването на светодиоди за източник на светлина води до намаляване на консумацията.

***PMT скенери*.** Светлината, постъпваща към сензора се разделя на три лъча, всеки от които преминава през червен, зелен и син филтър. Оттам попада върху фотоумножителни тръби, където се усилва, а след това се преобразува в електрически сигнал.

Този тип скенери, известни като *барабанни,* имат най-високи качества. Използват се за предпечатна подготовка и цифровизиране на висококачествени фотоматериали.

Основен параметър на скенерите е разделителната им способност. Измерва се в брой точки на инч - dots per inch (dpi) като се посочват *две* стойности.

Първата се нарича *оптическа* и се отнася до максималния брой точки по хоризонталата, които могат да бъдат сканирани.Това отговаря на броя на сканиращите елементи на един инч (колкото е по-висока оптическата разделителна способност, толкова е по-добро качеството на сканираното изображение).

Втората се нарича *апаратна* или *механична* и е свързана с максималния брой точки по вертикалата. Това отговаря на стъпката, с която се премества рамото, на което са разположени светлочувствителните елементи. Преместването се осъществява от стъпков двигател, чийто движение трябва да бъде бавно и равномерно.

Свързване на скенерите към компютъра

Срещат се **три** варианта за свързване на скенерите към компютъра :

- чрез паралелен порт
- чрез USB порт
- SCSI интерфейс.

Някои от евтините скенери работят с паралелен интерфейс. Все повече системи поддържат USB портове, поради което USB скенерите започват да изместват паралелните. Скенер със SCSI интерфейс изисква да се добави SCSI адаптер.

Тема 35. Устройства за извеждане на информация. Принтери, монитори.Видове. Принцип на работа. Свързване към компютъра

Устройствата за извеждане на информация служат за преобразуване на получената след обработката информация в цифров вид в разбираем за потребителя аналогов вид (букви, символи, картина , звук). Делят се на две основни групи:

- за временен изход

- за дълготраен изход

Устройствата за временен изход са предназначени за извеждане на визуална или звукова информация, която ще се използва от потребителя само в момента на извеждането й или в малък интервал от време след това (монитор, високоговорител).

Устройствата за дълготраен изход извеждат информацията напечатана на хартия (принтер, плотер)

Монитори

Съществуват два вида монитори:

- CRT монитор
- LCD монитор

CRT (Cathod - Ray Tube) - монитор с електронно-лъчева тръба.

Електронно-лъчевата тръба *е* вакуумна стъклена тръба, в единия край на която се намира излъчващият електрони катод, а в другия край се намира екран, покрит с тънък слой луминофор. Когато една точка от този слой бъде „бомбардирана" с електрони, тя светва за кратко време.

Излъчените от катода електрони се движат хаотично.За да се получи смислено изображение те трябва да се удрят в луминофорния слой точно на определени места и с определена сила. Това се извършва от система електроди - управляващ, ускоряващ и фокусиращ, обединени под името *електронен прожектор* или *електронна пушка* и отклонителна система, състояща се от две двойки бобини.

Формираният от електронната пушка лъч се придвижва под управление на отклоняващите бобини до различни точки на екрана. Започвайки движението си от горния ляв ъгъл и преминавайки през екрана до горния десен ъгъл, той оставя линия по екрана, наречена *растерна линия.* При достигане на десния край на екрана, линията се прекратява и електронният лъч се препозиционира в лявата част на екрана, на един ред под първата линия (*хоризонтален обратен ход на лъча)* .Започва изчертаването на втория ред, на третия и така сканирането по редове продължава докато електронният лъч стигне до долния десен ъгъл на екрана. Така се формира еднократно изчертаване на изображението. За да се получи устойчива картина, която може да се възприеме от човешкото око е необходимо многократно изчертаване на изображението. За целта електронният лъч се препозиционира отново в горният ляв ъгъл (*вертикален обратен ход на лъча*) и сканирането на екрана се повтаря. Честотата, с която се извършва вертикалния обратен ход на лъча се нарича *вертикална честота на сканиране* или още *честота на опресняване* на екрана.

За формиране на цветно изображение се използва цветна ЕЛТ. Тя използва комбинация от три цветни фосфора - червен, зелен и син-подредени в близкостоящи тройки- точки (триади), наречени ***пиксели*** (picture elements - елементи на образа). Като се използват три различни електронни прожектори (електронни пушки) за всеки елемент от тройката, тези елементи могат да светят с различна сила и така да образуват различни цветове на пиксела.

Черният цвят се получава при отсъствие на всякаква светлина, а бялият- при максимален интензитет и на трите основни цвята. Всеки един от трите основни цвята се получава при отсъствие на другите два. Цветовете могат да бъдат от 2 до над 16 000 000.

Мониторите се характеризират с големина на екрана, която се измерва в инчове - 14", 15", 17", 19", 21.

Качеството на произведеното на екрана изображение е функция на два фактора:

- Скоростта , с която изображението се пресканира на екрана (честотата на опресняване)
- Броят пиксели на екрана

Колкото по-висока е честотата на опресняване на екрана, толкова по устойчива е картината, липсва трептене на образа (не се уморяват очите).

Колкото повече са пикселите за даден размер на екрана, толкова по-фино е изображението. Количеството пиксели на екрана се нарича *разделителна способност*. Изразява се във формата "X по Y".

Недостатъци на CRT мониторите са големият им размер, високата консумация на електричество и чупливостта.

LCD (Liquid Crystal Display - дисплей с течни кристали).

Принципът на действие на LCD дисплеите се базира на ефекта на поляризирането на светлината, пропусната през течнокристално вещество в електромагнитно поле. Течният кристал, за разлика от обикновения, няма подредена вътрешна структура. Молекулите в него са разположени хаотично и могат свободно да се движат. Под въздействие на външно електрическо поле молекулите на течния кристал се нареждат в подредена структура. В зависимост от състоянието на течния кристал, той може да пропуска или не светлината през него.

LCD мониторът се състои от две паралелно разположени стъклени пластини, между които е поставен течнокристален материал. От вътрешната страна на пластините са нанесени ленти прозрачни проводници (електроди). На едната пластина те са хоризонтални, а на другата вертикални. Във всяка точка от течния кристал, където се пресичат хоризонтален и вертикален електрод се създава един *пиксел*. С подаването на напрежение на съответните двойки проводници може да се създаде потенциал във всяка точка на така образуваната матрица с цел поляризиране на светлината. Зад така формирания поляризиращ слой се намира лампата за осветление.

За да се получи изображение на екрана е необходимо активирането различни точки от него. За целта трябва да се активират хоризонталните и вертикални електроди на панела. Сканиращата редовете и колоните схема симулира действието на обхождащия електронен лъч в ЕЛТ, като адресира последователно всеки ред колона по колона. Въпреки, че вертикалният електрод е активиран за кратко време при всяко хоризонтално сканиране, пикселите остават засветени продължително време, тъй като сканиращата скорост е много висока.

LCD дисплеите могат да се изграждат в два варианта:

- с пасивна матрица
- с активна матрица

При дисплеите с *активна* матрица към всяка пресечна точка на ред и колона (пиксел) е добавен транзистор (реализиран непосредствено върху подложката на панела с помощта на

тънки слоеве) за подобряване моментите на превключване. Тези дисплеи са известни в практиката като TFT LCD (от TFT - Thin -Film Tranzistor).

TFT LCD имат по-бързи, отколкото при пасивните матрици схеми на сканиране. Реализират по-малко взаимно електрическо влияние между съседните клетки, което допуска прилагане на по-големи токове, с което пък се избягват сенките и ивиците, постига се по-висок контраст и яркост на изображението.

Изобразяването на информация от мониторите става под управление на специализирано за тях управляващо устройство, наричано в практиката ***видеоконтролер*** или ***видеоадаптер.***

Видеоадаптерът може да се реализира по три различни начина:

- Чипсет на дънната платка с интегрирано видео
- Самостоятелен видео чип на дънната платка
- Разширителна *видеокарта* (платка, монтирана в един от слотовете за разширение на компютъра).

Всички видеоадаптери съдържат определени основни компоненти:

- Video **BIOS;**
- Видеопроцесор;
- Видеопамет;
- Цифрово-аналогов преобразувател (Digital-to-Analog Converter - DAC);
- Шина за връзка

Видео BIOS

Видеоадаптерите включват **BIOS,** който е подобен като конструкция, но е напълно отделен от главния системен **BIOS.** Подобно на системния **BIOS, BIOS-ът** на видеоадаптера е под формата на **ROM** чип. Програмите в него позволяват да извежда информация на монитора по време на **POST** процедурата и първоначалното зареждане още преди каквито и да е софтуерни драйвери да са се заредили от хард диска в паметта.

Видеопамет

Видеоконтролерите имат собствена памет, наречена видеопамет, която се намира на точно определен адрес и е достъпна за програмистите. Всяко изображение се формира във видеопаметта, а екранното изображение се явява като нейно огледало. Колкото повече точки има на екрана и колкото повече цветове се изобразяват на него, толкова по-голям трябва да е обемът й. Количеството памет на видеоадаптера определя максималната разделителна способност на екрана и максималната дълбочина на цветовете (8-, 16-, 24- битов цвят).

Видеопроцесор

Изображението записано във видеопаметта, представлява поредица от единици и нули. Видеопроцесорът използва тази информация за формиране мястото и цвета на отделните пиксели на екрана.

Във видеоадаптерите се използват няколко основни типа графични чипове:

- Кадров буфер
- Графичен копроцесор;
- Графичен ускорител;
- 3D графичен процесор.

Буфер за кадрите изисква множество запаметяващи чипове, които запазват изображение, силно наподобяващо това, което се извежда на екрана. На всяка точка (пиксел) от екрана се полага съответно място във видеопаметта. Когато се даде на тази точка някаква числова стойност, пикселът се оцветява по съответния начин. Проблем при буферите за кадри е, че всеки един от тези пиксели трябва да бъде изчислен и установен от централния процесора на компютъра.

Графичния копроцесор е собствен процесор, разположен на видеокартата , проектиран за бърза обработка на пикселите. Има много бърза скорост на обработката, но и висока цена.

Графичният ускорител е специализиран чип, който реализира ограничен брой конкретни графични задачи. Работи под управление на централния процесор на компютъра. Има по-ниска скорост, но и по-ниска цена.

Цифро-Аналогов Преобразувател - DAC (Digital-Analog-Convertor)

Преобразуването на цифровите изображения, генерирани от компютъра в аналогови сигнали, които мониторът може да изобрази, се реализира от един цифрово-аналогов преобразувател (DAC-Digital-to-Analog Converter). Скоростта на RAMDAC се измерва в MHz. Колкото е по-бърз процесът на конвертиране, толкова по-висока е вертикалната честота на опресняване при което се избягва трептенето на образа (кадрова скорост 75Hz - 85 Hz и по-високи).

Шина за връзка

AGP шината е специално проектирана за видеосистемата на компютъра. Тя е разширение на PCI шината, но е предназначена за използване само от видеокартите, като им осигурява висококоростен достъп до основната памет на компютъра. AGP шината поддържа четири скорости: AGP 1X, AGP 2X, AGP 4X, AGP 8X.

Принтери

Принтерът е устройство, което служи за изобразяване на текст и графика върху дълготраен носител - хартия, фолио.

Има два основни вида принтери:

- ударни
- безударни

Ударните ползват механично устройство за нанасяне на отпечатък върху хартията (матричните принтери).

б*езударните* принтери ползват топлина, лазер или струя мастило. лазерните и мастилено-струйните.

Матричен принтер

Основен елемент е печатащата глава**.** Тя обединява в блок група игли, които удрят мастилена лента към хартията. Главата се движи хоризонтално по листа, за да бъде отпечатан даден ред, а всяка игла се активира, когато е необходимо да се формира елемент (точка) от даден символ. На всеки символ съответства матрица от точки, която се съхранява в специална ROM памет (наричана знакогенератор).

Матричните принтери се делят на две основни групи:

- Серийни
- Линейни

В серийните матрични принтери символите и изображенията се формират от печатащата глава. Печатащата глава съдържа печатащи игли подредени във вертикални колони в предния си край, и електромагнитен механизъм за "изстрелването" им.

В печатащите глави се използват две основни технологии:

- иглите се изстрелват от електромагнитна система подобна на използваната в класическите електромагнитни релета

- иглите се изстрелват от енергията съхранена в плоски пружини, държани в напрегнато (деформирано) състояние от постоянно магнитно поле

При подаване на сигнал за печат се създава електромагнитно поле, противоположно на това от постоянните магнити, при което пружината се освобождава и изстрелва иглата.

В *серийните* матрични принтери знаците се отпечатват от печатащата глава при хоризонталното и движение. Печатащата глава има определен брой печатащи игли подредени така, че до оформят вертикална колона в предния край на главата. При движение на главата по ширината на листа, печатащите игли се задействат избирателно за да отпечатват съответните символи.(Масово използваните глави имат **9** игли подредени в една колона или **24** игли подредени в две колони за по-добро качество на печат. В някои принтери предназначени за работа с високо натоварване се използват **18** иглени глави с две колони по **9** игли, позволяващи висикоскоростен печат).

Линейните принтери, аналогично на серийните ползват пинове, удрящи през мастилена лента върху хартията за да отпечатат точки, с които да синтезират изображението. Разликата е, че ползват чукчета вместо игли, подредени в хоризонтален ред и оформени като цялостен печатащ модул. Този модул е монтиран върху *совалка*, която вибрира в хоризонтално направление. Така всяко чукче печата поредица от точки в хоризонтална линия за едно преминаване на совалката.

Скоростта на печат се определя от броя символи, които печата принтерът за една секунда. Измерва се в cps (Characters Per Second

Мастилено-струен принтер

Мастилено-струйните принтери могат да се разглеждат като високотехнологична версия на матричните. Печатащата глава на тези принтери се състои от резервоар с мастило и електроника (интерфейсна верига, импулсни помпи и дюзи). В сравнение с матричния принтер вместо удар, който да пренася мастилото върху листа, то се впръсква под формата на капки от миниатюрни дюзи, всяка от които отговаря на една печатаща игла от ударните матрични принтери.

Капките се формират по един от следните два метода:

- Чрез *термичен шок* – мастилото в тънка тръбичка точно зад дюзата се нагрява, което води до увеличаване на налягането в тръбичката и то се изстрелва през отвора (bubble jet принтери)
- Чрез механична вибрация – мастилото се изтласква през дюзата посредством вибрации от *пиезо –кристал* (ink-jet принтери)

Съществуват две технологии за формиране на капката:

- Непрекъснат поток
- Капка по заявка

Печатащата глава мести мастиления патрон странично върху хартията. Разстоянието между дюзите позиционира точките, които принтерът нанася върху хартията перпендикулярно на движението на главата, а електрониката на принтера съгласува сигналите, изпратени към мастиления патрон, за да позиционира точките по дължината на хартията.

Тези принтери имат допълнителна цветна опция. Цветните мастилено-струйни принтери използват един или два патрона с мастило - с циан, магента и жълто в единия и допълнително черно в другия.(CMYK цветови модел)

Лазерен принтер

Принципът им на действие се основава на факта, че някои материали реагират на светлината по особен начин. Лазерните принтери използват фини, сухи мастилени частици (наречени тонер), за да създадат изображение върху хартия.

Процесът започва в точката на допиране на електростатично зареден валяк и фоточувствителен барабан. Електростатичният валяк нанася равномерно електрически заряд върху барабана, което причинява отблъскване на частиците на тонера от барабана. Фоточувствителният барабан се върти спрямо лазерния сноп (който се движи линейно напред и назад). Навсякъде, където лазерният лъч освети барабана, електрическият заряд изчезва и тези точки привличат тонера от валяка с тонер. Барабанът продължава да се върти, пренасяйки изписаното с тонер изображение до контакта с листа хартия. Трансферен валяк привлича тонера върху хартията, където той полепва. Комбинацията от стопяващ и поддържащ валяк загрява тонера, свързва го с хартията и създава трайно изображение.

Целта при лазерния принтер е да се накара лазерният лъч да пише по барабана. При завършване на сканирането на предишния ред барабанът автоматично се завърта към следващия. Лазерният лъч е модулиран. Той бързо се включва (за осветените места) и се изключва (за неосветените), по един път за всяка точка, за да се оформи побитовото изображение. Лазерът се контролира от растеризиращ процесор (RIP - Raster Image Processor). Задачата на **RIP** е да превърне реда от символи в побитово изображение, което може да бъде отпечатано. Всъщност RIP работи като видеоплатка, тъй като той интерпретира командите за изчертаване, изчислява позицията на всяка точка върху страницата и вкарва нейната стойност в паметта на принтера. Паметта на принтера е растерно организирана, както и при видеоекрана, всяка клетка от паметта отговаря на определена точка от листа. Лазерните принтери обработват наведнъж по една страница и трябва да я възприемат графично, преди да започнат да нанасят точките върху листа.

Лазерният принтер трябва да получи цялостна информация за това, което ще печата, за да формира правилно изображението. Тъй като работят с цели страници, и скоростта им на печат се измерва с ppm (Pages per Minute).

Цветните лазерни принтери използват четири комплекта тонер, за да формират изображението, извършвайки обикновено четири минавания около фоточувствителния барабан преди да отпечатат изображението върху страницата. Тъй като изображението трябва да бъде растеризирано отделно за всеки цвят, изискванията за памет нарастват значително.

Свързване към компютъра

Принтерите се свързват към паралелния порт (LPT1-LPT3) на компютърната система чрез 25-пинов женски D-тип конектор. Информацията се предава по 8 бита едновременно. Съвременните системи имат EPP (разширен паралелен порт) и ECP (порт с увеличени възможности). Принтерите могат да се свържат и чрез USB порта към компютърната система.

Тема 40. Кодиране на източника на информация. Дефиниция за побуквен код, префиксен код, неравенство на Крафт. Оптимално побуквено кодиране. Дефиниции за източник на информация, дължина на кодираното съобщение и цена на кода. Алгоритъм на Шенон-Фано. Алгоритъм на Хафман за кодиране и декодиране.

1. Обща характеристика на побуквеното кодиране

Нека S е крайна изходна азбука. Думите над тази азбука са изходни текстове или съобщения. Нека $B = \{b_1, b_2, ..., b_m\}$ е крайна кодова азбука за кодиране на изходните съобщения.

Дефиниция: Функцията $k : A \to B^+$, такава, че $\kappa(a_i) \neq \kappa(a_j)$ при $i \neq j$ се нарича **побуквен код**, а думите $k_1 = k(a_1)$, $k_2 = k(a_2)$, …, $k_n = k(a_n)$ **- кодови думи**.

В компютърните системи се използват побуквените кодове EBCDIC, ASII и др. На всяка буква от азбуката и служебните символи от клавиатурата те съпоставят дума от азбуката {0, 1} с фиксирана дължина (8 двоични букви).

Означава се с $d(k_i)$ дължината на k_i-та кодова дума. Побуквеният код $k : A \to B^+$ е равномерен, ако $d(k_1) = d(k_2) = ... = d(k_n) = d$, където d е **дължина** на равномерния код.

Функцията $k : A \to B^+$ се разширява до $k^* : A^+ \to B^+$. Тогава за всяка дума $\alpha = \alpha_{i_1}\alpha_{i_2},...,\alpha_{i_l}$, $\alpha \in A^+$, се дефинира думата $k^*(\alpha) = k_{i_1}k_{i_2},...,k_{i_l}$, $k^*(\alpha) \in B^+$, която се нарича **кодирано съобщение**. Процедурата, построяваща по зададено съобщение съответното кодирано съобщение, се нарича **кодиране**.

Дефиниция: Побуквеният код $k : A \to B^+$ е **разделим или осигурява еднозначност на декодирането,** ако $\forall \beta \in B^+$, $\exists$ не повече от една $\alpha \in A^+$, такава, че $k^*(\alpha) = \beta$.

Процедурата, която по зададена $\beta \in B^+$ намира, ако съществува, съответната $\alpha \in A^+$ такава, че $k^*(\alpha) = \beta$ се нарича **декодиране**. Равномерните кодове осигуряват еднозначно декодиране. Кодираното съобщение се разбива на поддуми с дължина, равна на дължината на кода. Всяка поддума се замества с първообраза си в съответствие с дефиницията на кода.

Пример: Дадена е изходната азбука $A = \{a, b, c\}$, кодовата азбука $B = \{0,1\}$, кодовите думи $k_a = 01$, $k_b = 11$, $k_c = 0111$ и кодираното съобщение 01110111. То се декодира като *abab*, *abc*, *cab*, *cc*.

Извод: Неравномерните кодове не винаги осигуряват еднозначност на декодирането.

Дефиниция: Кодът k е префиксен, ако никое k_i не е префикс на k_j, $i \neq j$.

Теорема: Всеки префиксен код е разделим.

Обратното твърдение не е вярно. Например кодът $k_a = 01$, $k_b = 11$, $k_c = 0111$ за азбука $A = \{a, b, c\}$ не е префиксен, но е разделим.

Неравенство на Крафт: Нека $k : A \to B^+$ е разделим код, $|A| = n$, $|B| = m$ и $d(k_i) = l_i$. Тогава $\sum_{i=1}^{n} m^{-l_i} \leq 1$.

Теорема: За всеки разделим код $k : A \to B^+$ $|A| = n$, $|B| = m$ с дължина на кодовите думи $l_1 \leq l_2 \leq ...l_n$ съществува префиксен код $\bar{k}$, думите на който имат същите дължини.

2. Оптимално побуквено кодиране

Оптимален побуквен код $k : A \to \{0,1\}^+$ е този, с който, ако се кодира съвкупност от текстове, получената дължина на кодираните текстове да не е по-голяма от сумарната дължина на текстовете, кодирани с произволен друг разделим код $\bar{k} : A \to \{0,1\}^+$.

В общия случай съвкупността от текстове не е крайна, тази задача се решава със статистически методи.

Нека $\alpha \in A^+$ е произволен текст с дължина $d(\alpha)$ от съвкупността. Символите $a_i \in A$ в α се отличават с различна честота на появяване. На всяка буква a_i от азбуката $A \mid A| = n$ се съпоставя вероятност p_i, където $0 < p_i < 1$, $\sum_{i=1}^{n} p_i = 1$, $i = \overline{1,n}$. Тогава очакваният брой букви a_i в текста α се определя с израза: $p_i d(\alpha)$.

Дефиниция: Нека $A = \{a_1, a_2, ..., a_n\}$ е крайна азбука, а $p_1, p_2, ..., p_n$, $0 < p_i < 1$, $\sum_{i=1}^{n} p_i = 1$, са вероятностите за появата на съответните букви в думите от езика $L \subseteq A^+$. Тогава $I_L = (A; p_1, p_2, ..., p_n)$ се нарича **източник на информация** или **източник.**

Нека $k : A \to \{0,1\}^+$ е разделим побуквен код за A, $|A| = n$ с дължина на кодовите думи $l_1, l_2 ... l_n$. Нека α е съобщение от L с дължина $d(\alpha)$, т.е. $\alpha \in L$. **Броят символите, които буква $a_i \in \alpha$ внася в дължината на кодираното съобщение $k^*(\alpha)$ се определя с израза**: $l_i p_i d(\alpha)$. **Сумата** $\sum_{i=1}^{n} l_i p_i d(\alpha) = d(\alpha) \sum_{i=1}^{n} l_i p_i$ е очакваната **дължина на кодираното съобщение**. Следователно, очакваната дължина на всяко кодирано съобщение $k^*(\alpha)$ зависи от $d(\alpha)$ и константата $C(k, I_L) = \sum_{i=1}^{n} l_i p_i$, определени еднозначно от I_L и кода k. За да бъде кодираното съобщение с минимална дължина при зададен I_L, необходимо и достатъчно е k да се избере такова, че $C(k, I_L)$ да има минимална стойност.

Дефиниция: При зададен източник $I_L = (A; p_1, p_2, ..., p_n)$ и $k : A \to \{0,1\}^+$ - разделим побуквен код за I_L с дължина на кодовите думи $l_1, l_2 ... l_n$ константата $C(k, I_L) = \sum_{i=1}^{n} l_i p_i$ се нарича **цена на кода**, която при заден източник се означава $C(k)$.

Разделим код $k_0 : A \to \{0,1\}^+$ за I е **оптимален побуквен код**, ако за всеки друг разделим код k за I_L е в сила неравенството $C(k_0) < C(k)$.

Цената на оптималния побуквен код $C(k_0)$ при зададен $I = (A; p_1, p_2, ..., p_n)$ не може да бъде по-малка от константа $H(I) = \sum_{i=1}^{n} p_i \log_2 \left(\frac{1}{p_i} \right)$, наречена **ентропия** на I, т.е. $C(k_0) \geq H(I)$.

Теорема: За всеки разделим код k на източника $I = (A; p_1, p_2, ..., p_n)$, с дължини на кодовите думи $l_1, l_2 ... l_n$ е в сила $C(k) \geq H(I)$.

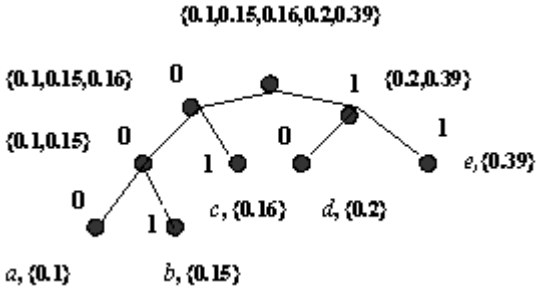
3. Алгоритъм на Шенън-Фано

Дадено: Източник $I = (A; p_1, p_2, ..., p_n)$.
Търси се: Префиксен код.

1. Вероятностите се сортират по големина, $(p_1 \geq p_2 \geq ... \geq p_j \geq ... \geq p_m)$, $m = n$: стъпка $r = 1$.
2. Разбива се множеството от вероятности на две групи $(p_1, p_2, ..., p_j)$ и $(p_{j+1}, p_{j+2}, ..., p_m)$, така, че $\left| \sum_{i=1}^{j} p_i - \sum_{i=j+1}^{m} p_i \right|$ да бъде минимална.
3. На буквите, чиито вероятности са в първата група, присвоява r-та кодова буква 0, а на останалите - r-та кодова буква 1.
4. За група от вероятности с един елемент, процедурата завършва, а за всяка от останалите групи в многостъпковата процедура елементите се означават с номера от 1 до m и се преминава рекурсивно ($r = r + 1$) към стъпка 2.

Алгоритъмът на Шенън-Фано съпоставя на всеки източник кореново двоично дърво. Всички върхове, с изключение на корена, съответстват на подмножества от вероятности на букви, които имат еднакви първи r кодови букви в кода, изграждан от алгоритъма. Левият и десен син на съответния връх са подмножествата от вероятности на букви на съпоставеното му множество, които имат за $r + 1$ – ва кодова буква съответно 0 и 1.

Пример: Даден е източник $I = (\{a, b, c, d, e\}; 0.1, 0.15, 0.16, 0.2, 0.39)$. В съответствие с алгоритъма на Шенон-Фано се построява двоичното дърво (Фиг.1).



Фиг. 1. Двоично дърво

Търсеният код за елементите a, b, c, d е както следва: $k_a = 000$, $k_b = 001$, $k_c = 01$, $k_d = 10$, $k_e = 11$. Цената на кода е $C(k) = 3(0.1+0.15)+2(0.16+0.2+0.39)=0.75+1.5 = 2.25$.

Теорема: Ако $I = (A; p_1 \geq p_2 \geq ... \geq p_n)$ е източник на информация съществува оптимален префиксен код $k_0 : A \to \{0,1\}^+$ с дължина на кодовите думи $l_{01} \leq l_{02} \leq ... \leq l_{0n-1} = l_{0n}$, като кодовите думи k_{0n-1} и k_{0n} имат вида $k_{0n-1} = \alpha 0$, $k_{0n} = \alpha 1$.

Теорема: Ако $k = \{k_1, k_2, ..., k_n\}$ с дължина на кодовите думи $l_1, l_2 ... l_n$ е оптимален код за източника $I = (A; p_1 \geq p_2 \geq ... \geq p_n)$ и вероятност p_i се представя във вида $p_i = q_1 + q_2$, като $p_n \geq q_1 \geq q_2$, тогава $k' = \{k_1, ..., k_{i-1}, k_{i+1}, ..., k_n, k_i 0, k_i 1\}$ е оптимален код за източника $I' = (B; p_1 \geq ... \geq p_{i-1} \geq p_{i+1} \geq ... \geq p_n \geq q_1 \geq q_2)$, където B е произволна азбука с $n + 1$ букви.

4. Алгоритъм на Хафман

Дадено: Източник $I = (A; p_1 \geq p_2 \geq ... \geq p_n)$.

Търси се: Оптимален код k за I.

1. От източника I се строят източниците I_{n-1}, I_{n-2}, …, I_2 с $n-1$, $n-2$,…, 2 букви на кодовите думи, като всеки път се сумират най-малките вероятности на текущия източник, като се записват от кои две стойности се е получила сумата.
2. За източника I_2 оптималният код се състои от думите 0 и 1.
3. От оптималния код I_2 се строят последователно оптималните кодове на източниците I_3, I_4, ..., I_{n-1}, I, като кодовата дума, съответстваща на вероятността, получена като сума от двете най-малки вероятности от предния източник, се разширява отдясно с 0 и 1. На получените две думи се присвояват най-малките вероятности на този източник. На всички останали думи се присвояват старите вероятности.

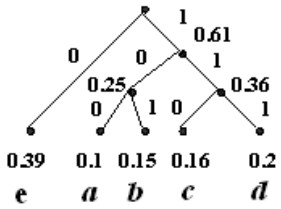
а. Кодиране - двоично дърво на Хафман

На стъпка 1 на върховете на двоичното дърво се преписва по една вероятност, а на стъпка 2 – по една дума при следните правила:

В права посока:
1. Изобразяват се n върха, като n отделни дървета, с начални вероятности.
2. Всеки път, когато се сумират двете най-малки вероятности, присвоени на корените-дървета r_1 и r_2, се избира нов връх - r, на който се присвоява получената сума, като дърветата с корени r_1 и r_2 стават ляво и дясно поддърво на новия връх.
3. Когато се получат две поддървета, процедурата завършва, като на корена на полученото двоично дърво се присвоява числото 1.

В обратната посока:
1. На корена на двоичното дърво се приписва думата ε.
2. Всеки път, когато на един връх се приписва думата α, на левия му син се присвоява думата $\alpha 0$, а на десния - $\alpha 1$, докато по този начин на всеки връх на дървото се присвоява дума.
3. Листата на полученото дърво са надписани с вероятностите на началния източник, а присвоените думи на всеки връх са думите на оптималния код за този източника – **код на Хафман**.

Пример: Да се илюстрира **алгоритъмът на Хафман** чрез изграждане на кореново двоично дърво върху източника $I = (\{a,b,c,d,e\};\{0.1, 0.15, 0.16, 0.2, 0.39\})$ от примера за алгоритъма на Шенън-Фано. На фиг. 2 е представено двоичното дърво на Хафман. Търсеният оптимален код е : $k_{0a} = 100$, $k_{0b} = 101$, $k_{0c} = 110$, $k_{0d} = 111$, $k_{0e} = 0$. Цената на кода е $C(k_0)$ = 3(0.1+0.15+0.2)+1(0.39)= 2.22.



Фиг. 2.Двоично дърво на Хофман

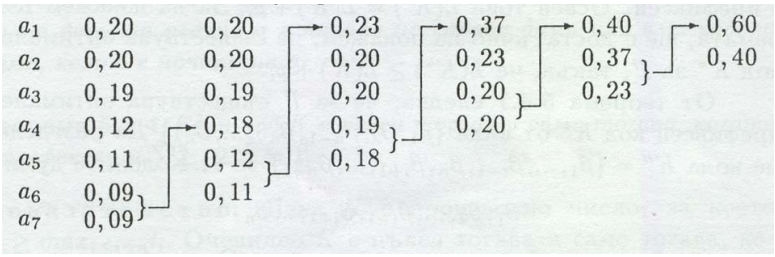
б. Декодиране

1. Процедурата стартира с началния символ на кодираното съобщение и корена на дървото. Ако текущата буква е 0, се преминава към левия син на текущия връх, ако е 1 –десния син.
2. Ако върхът, в който се попада не е лист (например от 1 до 0, 61), обхождането на дървото се продължава със следващата буква на кодираното съобщение (напр. 1). Ако върхът е лист буквата, присвоена на съответния лист е била кодирана.
3. Ако се стигне до края на съобщението, то декодирането е завършено. В противен случай се преминава към корена на дървото и описаните действия се повтарят.

Пример: Кодираното с кода на Хафман съобщение 01110110101 се декодира чрез дървото на Хафман като *edecb*.

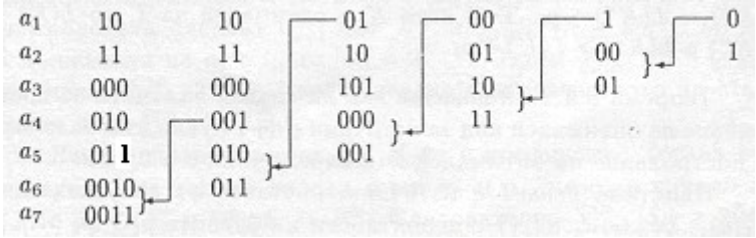
Пример: Да се кодира по алгоритъма на Хафман източникът $I = (\{a_i\}, \{p_i\})$, където $i = \overline{1,7}$, $p_1 = p_2 = 0,20$, $p_3 = 0,19$, $p_4 = 0,12$; $p_5 = 0,11$; $p_6 = p_7 = 0,09$.

1. Изграждане източника $I_{n=7}$ на източниците I_6, I_5,…, I_2 с 6, 5, ..., 2 букви в кодовите думи, като се сумират двете най-малки вероятности (Фиг. 3). Оптималният код се състои от думите 0 и 1.



Фиг. 3. Редуциране на източника $I_{n=7}$ до I_2.

От оптималния код I_2 се строят последователно оптималните кодове на източниците I_3, I_4, ..., I_6, $I = I_7$, като кодовата дума, съответстваща на вероятността, получена като сума от двете най-малки вероятности от предния източник, се разширява отдясно с 0 и 1. На получените две думи се присвояват най-малките вероятности на този източник. На всички останали думи се присвояват старите вероятности (Фиг. 4).



Фиг. 4. Изграждане на оптималните кодове на източниците от I_2 до $I = I_7$.

Тема 41. Линейни кодове за защита от грешки. Дуални кодове.
Дефиниция. Генераторна (пораждаща) матрица. Проверочна матрица на линеен код. Кодиране с линейни кодове. Декодиране на линейни кодове. Таблица на Слепиан. Синдромно декодиране (алгоритъм).

Дефиниция. Генераторна (пораждаща) матрица. Проверочна матрица на линеен код. Кодиране с линейни кодове. Декодиране на линейни кодове. Таблица на Слепиан. Синдромно декодиране (алгоритъм).

1. Линейни кодове - определения

Допуска се, че азбуката F_q е поле на Galois (Галоа) $GF(q)$, където q е проста степен и нека $(F_q)^n$ е векторно пространство $V(n, q)$. Векторът $(x_1, x_2,\ldots, x_n)$ се записва по следния начин $x_1x_2\ldots x_n$.

Линеен код над $GF(q)$ е подпространство на $V(n, q)$ за някое положително цяло число n. Следователно, подмножеството C на $V(n, q)$ е линеен код т.и.с.т., когато:

1. $\mathbf{u} + \mathbf{v} \in C$ за всички $\mathbf{u}$ и $\mathbf{v}$ в C.

2. $a\mathbf{u} \in C$ за всички $\mathbf{u} \in C$, $a \in GF(q)$.

Бинарен код е линеен, тогава и само тогава, когато сумата от две кодови думи е кодова дума.

Ако C е k-мерно подпространство на $V(n, q)$, тогава линейният код C се нарича $[n, k]$ код. В случай, че трябва да се означи минималното разстояние d на C, кодът се записва като $[n, k, d]$ код.

1. q – ичен $[n, k, d]$ код е също q – ичен $[n, q^k, d]$ код, но не всеки $[n, q^k, d]$ код е $[n, k, d]$ код.

2. Нулевият вектор $\mathbf{0}$ принадлежи на линейния код.

3. Линейните кодове са група от кодове.

Дефиниция: Матрица с размери $k \times n$, чиито редове формират базис на линеен $[n, k]$ – код се нарича **генераторна матрица** на кода и се означава с G.

Примери:

1. Кодът $C_2\begin{cases}000\\011\\101\\110\end{cases}$ е с размери [3, 2, 2] и има генераторна матрица $\begin{bmatrix}011\\101\end{bmatrix}$.

2. Код с дължина $n = 7$, $M = q^k = 16$ и минимално разстояние $d = 3$ е с размери [7, 4, 3] и има генераторна матрица

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

3. q-ичен код с повторение с дължина n над $GF(q)$ е $[n, 1, n]$–код с генераторна матрица $[1\ 1\ \ldots.\ 1]$.

КОДИРАНЕ И ДЕКОДИРАНЕ С ЛИНЕЙНИ КОДОВЕ

1. Кодиране с линеен код

Допуска се, че C е $[n, k]$-код върху GF(q) с генераторна матрица G. Кодът C съдържа q^k кодови думи, т.е. те могат да се използват за формиране на q^k отделни съобщения. Тези съобщения се идентифицират с q^k на брой k-орки от $V(k, q)$, като векторът на съобщението $\mathbf{u} = u_1\, u_2\, \ldots\, u_k$ се кодира чрез умножение отдясно с генераторната матрица G. Ако стълбовете на G са $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n$, то

$$\mathbf{u}G = \sum_{i=1}^{k} u_i \mathbf{c}_j\ ,\ j = \overline{1,n}\ ,$$

където $\mathbf{u}G$ е кодовата дума с дължина n над C – линейна комбинация от колоните на генераторната матрица.

Кодиращата функция $\mathbf{u} \rightarrow \mathbf{u}G$ изобразява $V(k, q)$ върху k-димензионно подпространство (именно код C) от $V(k, q)$.

Ако G е в стандартна форма кодиращото правило е по-просто. Предполага се, че $G = [I_k | A]$, където $A = [a_{ij}]$ е $k \times (n - k)$ матрица. Векторът на съобщението $\mathbf{u}$ се кодира по следния начин

$$\mathbf{x} = \mathbf{u}G = x_1\, x_2 \ldots\, x_k\, x_{k+1} \ldots\, x_n,$$

където $x_i = u_i$, $1 \le i \le k$ са цифрите на съобщението, а $x_{k+j} = \sum_{i=1}^{k} u_i a_{ij}$, $1 \le j \le n - k$, са проверовъчните елементи. Елементът a_{ij} е на i – тия ред и j –тия стълб на генераторната матрица. Проверовъчните цифри представляват **излишъкът**, който се добавя към съобщението, за да се осигури защита от шума.

Пример: Нека C е бинарен [7,4]-код със стандартна матрица

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Вектор съобщението (u_1, u_2, u_3, u_4) се кодира по следния начин

$(u_1, u_2, u_3, u_4, u_1 + u_2 + u_3, u_2 + u_3 + u_4, u_1 + u_2 + u_4)$.

Например:

0	0	0	0		0	0	0	0	0	0	0
1	0	0	0	се кодира	1	0	0	0	1	0	1
1	1	1	0		1	1	1	0	1	0	0

Декодиране с линеен код

Предполага се, че по информационния канал се изпраща кодовата дума $\mathbf{x} = x_1 x_2 \ldots x_n$. Приетият вектор е $\mathbf{y} = y_1 y_2 \ldots y_n$. Векторът на грешката **e** се дефинира последния начин $\mathbf{e} = \mathbf{y} - \mathbf{x} = e_1 e_2 \ldots e_n$.

Декодерът взема решение по вектора **y,** коя кодова дума **x** е изпратена или, което е еквивалентно, кой вектор на грешката **e** се е случил.

Дефиниция: Нека C е $[n, k]$-код върху $GF(q)$ и **a** е някой вектор в $V(n, q)$. Тогава множеството $\mathbf{a} + C = \{ \mathbf{a} + \mathbf{x} \mid \mathbf{x} \in C\}$ се нарича комножество (клас) на C.

Лема: Ако $\mathbf{a} + C$ е комножество на C и векторът $\mathbf{b} \in \mathbf{a} + C$, тогава

$\mathbf{b} + C = \mathbf{a} + C$.

Теорема на Лагранж:

Ако C е $[n, k]$ –код върху $GF(q)$, тогава

1. Всеки вектор на $V(n, q)$ се намира в някое съседно множество (клас или комножество) на C.
2. Всяко комножество съдържа точно q^k вектори.
3. Две комножества са или разединени, или съвпадат (частичното препокриване е невъзможно).

Пример: Нека C е бинарен $[4, 2]$ – код с генераторна матрица

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

т.е. $C = \{0000, 1011, 0101, 1110\}$.

Тогава комножествата на C са

$0000 + C = C$ (самото множество)

$1000 + C = \{1000, 0011, 1101, 0110\}$,
$0100 + C = \{0100, 1111, 0001, 1010\}$,

$0010 + C = \{0010, 1001, 0111, 1100\}$.

Комножеството $0001 + C = \{0001, 1010, 0100, 1111\}$, което е същото, както комножеството $0100 + C = \{0100, 1111, 0001, 1010\}$, което може да се предвиди от лемата, тъй като $0001 \in 0100 + C$. Подобен е случая, например при $0111 + C = 0010 + C$.

Стандартна решетка по **Slepian** за $[n, k]$-код C е $q^{n-k} \times q^k$ – решетка от всички вектори в $V(n, q)$, в която първият ред се състои от код C с **0** на най-левия край, а останалите редове са комножествата $\mathbf{a}_i + C$, всичките подредени в съответния ред, с лидера на множеството - на левия край. Стандартната решетка може да се конструира в съответствие с алгоритъма:

1. Изписват се кодовите думи на C, стартирайки с $\mathbf{0}$, като първи ред.
2. Избира се някакъв вектор $\mathbf{a}_1$, не в първи ред, с минимално тегло. Изписва се комножеството $\mathbf{a}_1 + C$ като втори ред чрез поставяне на вектора $\mathbf{a}_1$ под вектора $\mathbf{0}$ и векторът $\mathbf{a}_1 + \mathbf{x}$ под $\mathbf{x}$ за всяко $\mathbf{x} \in C$.
3. От тези вектори, не в редовете 1 и 2, се избира $\mathbf{a}_2$ с минимално тегло и се изписват комножествата $\mathbf{a}_2 + C$, както т.2, за да се получи третият ред.
4. Продължава се по същия начин, докато всички комножества се изпишат като всеки вектор на $V(n, q)$ се появява точно само веднъж.

Пример: Стандартната решетка за кода от предния пример има вида

кодови думи $\rightarrow$

0000	1011	0101	1110
1000	0011	1101	0110
0100	1111	0001	1010
0010	1001	0111	1100

$\uparrow$

лидери на комножествата

В стандартната решетка, всеки елемент е сума от кодовата дума на върха на неговата колона и лидера на комножеството на най-левия край на неговия ред.

Декодиране чрез стандартна решетка

Приема се вектор **y** (например, 1111 – от горния пример). Намира се неговата позиция в решетката. Тогава декодерът взема решение, че векторът на грешката **e** е лидерът (0100) на комножеството, намиращ се на най-левия край на **y**, като **y** се декодира като кодова дума $\mathbf{x} = \mathbf{y} - \mathbf{e}$ (1011), намираща се на върха на колоната, която съдържа **y**.

Извод: Приеманият вектор се декодира като кодовата дума на върха на неговата колона в стандартната решетка.

Векторите на грешките, които се използват за корекция са точно лидерите на комножествата, независимо от това коя кодова дума се предава. Чрез избор на вектор с минимално тегло във всяко комножество като лидер на комножеството, се гарантира, че декодирането чрез стандартна решетка е схема на декодиране по най-близкия съсед.

ОРТОГОНАЛНИ (ДУАЛНИ) КОДОВЕ, ПРОВЕРОВЪЧНА МАТРИЦА И СИНДРОМНО ДЕКОДИРАНЕ

Ако е зададен линеен $[n, k]$ – код C, то ортогоналният код на C, който се означава $C^{\perp}$, дефинира множество от тези вектори на $V(n, q)$, които са ортогонални на на всяка кодова дума от C, т.е.

$$C^{\perp} = \{\mathbf{v} \in V(n, q) \mid \mathbf{v} \cdot \mathbf{u} = 0 \text{ for all } \mathbf{u} \in C\}.$$

Лема 2

Нека C е $[n, k]$-код с генераторна матрица G. Тогава векторът **v** от $V(n, q)$ принадлежи на $C^{\perp}$, тогава и само тогава, когато **v** е ортогонален на всеки ред от G, т.е.

$$\mathbf{v} \in C^{\perp} \Leftrightarrow \mathbf{v}\, G^T = 0,$$

където G^T означава транспонирана матрица на G.

Теорема: Допуска се че C е $[n, k]$-код върху $GF(q)$. Тогава ортогоналният код $C^{\perp}$ на C е линеен $[n, n - k]$-код с димензия $\dim C^{\perp} = n - k$.

Примери:

1. Кодът $C = \begin{cases} 0000 \\ 1100 \\ 0011 \\ 1111 \end{cases}$ с размер [4,2] има ортогонален код $C^{\perp} = C$ с размер [4,2] с генераторна матрица вида $G = \begin{cases} 1100 \\ 0011 \end{cases}$

2. Кодът $C = \begin{cases} 000 \\ 110 \\ 011 \\ 101 \end{cases}$ с размер [3,2] има ортогонален код $C^{\perp} = \begin{cases} 000 \\ 111 \end{cases}$ с размер [3,1] с генераторна матрица вида $G = \{111$.

Дефиниция: Проверовъчна матрица H за $[n, k]$-код C е генераторната матрица на $C^{\perp}$.
Матрицата H е $(n\text{-}k) \times n$, матрица, удовлетворявайки уравнението $GH^T = \mathbf{0}$, където H^T означава транспонираната на H; $\mathbf{0}$ е нулева матрица. От лема 2 и теорема 5 следва, че ако H е проверовъчна матрица на C, тогава

$C = \{\mathbf{x} \in V(n, q) \mid \mathbf{x}H^T = 0\}$.

По този начин всеки линеен код е напълно определен чрез проверовъчната матрица.

В примера 1 матрицата $\begin{bmatrix} 1100 \\ 0011 \end{bmatrix}$ е генераторна матрица и проверовъчна матрица, докато в примера 2 матрицата [111] е проверовъчна матрица

Редовете на проверовъчната матрица са проверовъчни елементи върху кодовите думи. Те показват, че определени линейни комбинации от координатите на всяка кодова дума са нули. Кодът е изцяло определен чрез проверовъчната матрица, например, ако

$H = \begin{bmatrix} 1100 \\ 0011 \end{bmatrix}$, тогава C е кодът, координатите на кодовите думи на който принадлежат на множеството

$\{(x_1, x_2, x_3, x_4) \in V(4,2) \mid x_1 + x_2 = 0, x_3 + x_4 = 0\}$, т.е. $C = \begin{cases} 0000 \\ 1100 \\ 0011 \\ 1111 \end{cases}$

Уравненията $x_1 + x_2 = 0$ и $x_3 + x_4 = 0$ се наричат проверовъчни уравнвния.

Ако проверовъчната матрица има вида $H = [111]$, тогава C се състои от тези вектори на $V(n, 2)$, сумата от координатите на които по модул 2 е равна на нула.

Теорема 6. Ако $G = [I_k \mid A]$ е генераторна матрица със стандартна форма на $[n, k]$-код C, тогава проверовъчната матрица за C има вида $H = [-A^T \mid I_{n-k}]$.

Пример:
Код с размер [7, 4, 3] има генераторна матрица със стандартна форма от вида

$$G = \left[\begin{array}{c|c} & 101 \\ I_4 & 111 \\ & 110 \\ & 011 \end{array}\right],$$

тогава проверовъчната матрица има вида

$$H = \left[\begin{array}{c|c} 1110 & \\ 0111 & I_3 \\ 1101 & \end{array}\right].$$

Дефиниция: Проверовъчната матрица H е в стандартна форма ако има вида

$H = [\mathrm{B} \mid I_{n-k}]$.

Ако кодът е дефиниран с проверовъчна матрица в стандартна форма, която е във вида $H = [B \mid I_{n-k}]$, тогава генераторната матрица за кода има вид $H = [I_k \mid -B^T]$. Много кодове, например, код на Hamming, се дефинират чрез проверовъчна матрица или, което е еквивалентно, чрез множество от проверовъчни уравнения. Ако кодът е зададен чрез проверовъчна матрица H, която не е в стандартна форма, тогава H може да се редуцира до стандартна форма по същия начин както се получава стандартна форма за генераторната матрица.

СИНДРОМНО ДЕКОДИРАНЕ

Допуска се, че H е проверовъчна матрица на $[n, k]$–код. Тогава за някакъв вектор $\mathbf{y} \in V(n, q)$, векторът-ред $1 \times (n - k)$

$$\mathrm{S}(\mathbf{y}) = \mathbf{y}\, H^T$$

се нарича **синдром** на **y.**
1. Ако редовете на H са $\mathbf{h}_1, \mathbf{h}_2, \ldots \mathbf{h}_{n-k}$, тогава

$$\mathrm{S}(\mathbf{y}) = (\mathbf{y} \,.\, \mathbf{h}_1, \mathbf{y} \,.\, \mathbf{h}_2, \ldots \mathbf{y} \,.\, \mathbf{h}_{n-k})$$

2. $\mathrm{S}(\mathbf{y}) = \mathbf{0} \Leftrightarrow \mathbf{y} \in C$.

Лема:

Два вектора **u** и **v** се намират в едно комножество (съседно множество или клас) на C, т.и.с.т., когато те имат един и същ синдром.

Съществува взаимно-еднозначно съответствие между комножествата и синдромите.

При стандартното решетъчно декодиране, ако n има малка стойност лесно се локализира приеманият вектор **y** в решетката. Но, ако n има голяма стойност, трудно се открива лидерът на ко-множество, който съдържа **y**.

Пример:

Да се изчисли синдромът S(**e**) за всеки лидер на комножество **e** и разшири стандартната решетка чрез изброяването на синдромите, като допълнителна колона.

Пример: Нека C е бинарен [4, 2] – код с генераторна матрица

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix},$$

т.е. C = {0000, 1011, 0101, 1110}.

Чрез прилагане на теорема 6 се получава проверовъчната матрица

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}.$$

Синдромите на лидерите на комножествата са:

S(0000) = 00
S(1000) = 11
S(0100) = 01
S(0010) = 10.

Стандартната решетка получава вида

Лидер на ко-множеството				Синдроми
0 0 0 0	1 0 1 1	0 1 0 1	1 1 1 0	0 0
1 0 0 0	0 0 1 1	1 1 0 1	0 1 1 0	1 1
0 1 0 0	1 1 1 1	0 0 0 1	1 0 1 0	0 1
0 0 1 0	1 0 0 1	0 1 1 1	1 1 0 0	1 0

Декодиращ алгоритъм:

1. Ако се приеме вектор **y,** се изчислява синдромът

$S(\mathbf{y}) = \mathbf{y}H^T$.

2. Определя се мястото на $S(\mathbf{y})$ в колоната на синдромите на стандартната решетка.
3. Определя се мястото на **y** в съответния ред и се декодира като кодова дума, която се намира на върха на колоната, която съдържа **y**.

Например, ако е приет векторът 1111, синдромът S(1111) = 01. Векторът 1111 се оказва в третия ред на решетката. Следователно декодираният вектор е 1011.

При числена реализация на декодиране чрез стандартна решетка е необходимо да се запазят само две колони (синдромите и лидерите на ко-множествата) в паметта на компютъра. Това се нарича *синдромна претърсваща таблица*.
Синдромната претърсваща таблица за кода от предходния пример има вида

Синдром **z**	Лидер на ко-множество $f(\mathbf{z})$

0 0	0 0 0 0
1 1	1 0 0 0
0 1	0 1 0 0
1 0	0 0 1 0

Декодиращ алгоритъм

1. За приетият вектор **y** се изчислява синдромът $S(\mathbf{y}) = \mathbf{y}H^T$.
2. Нека $\mathbf{z} = S(\mathbf{y})$ е синдромът. Определя се мястото на **z** в първата колона на претърсващата таблица.
3. Декодира се **y** като $\mathbf{y} - f(\mathbf{z})$.

Например, ако **y** = 1111, тогава $S(\mathbf{y}) = 01$, а приетият вектор се декодира като

$$1111 - 0100 = 1011.$$

Пример: Нека C е бинарен код с генераторна матрица $G = \begin{bmatrix} 10110 \\ 01011 \end{bmatrix}$. Тогава проверовъчната матрица е $H = \begin{bmatrix} 10100 \\ 11010 \\ 01001 \end{bmatrix}$

Чрез изчисляване на синдромите на лидерите на ко-множествата чрез уравнение $S(\mathbf{y}) = \mathbf{y}H^T$, се определя горната част на претърсващата таблица, следователно (втората колона се записва първа):

Синдром **z**	Лидер на ко-множество $f(\mathbf{z})$
0 0 0	0 0 0 0 0
1 1 0	1 0 0 0 0
0 1 1	0 1 0 0 0
1 0 0	0 0 1 0 0
0 1 0	0 0 0 1 0
0 0 1	0 0 0 0 1

Ако се приеме вектор **y**, изчислява се $S(\mathbf{y})$ и се декодира ако $S(\mathbf{y})$ се появява в **z** колона. Ако $S(\mathbf{y})$ не се появява се изисква отново изпращане на съобщението.

Пример:
1. Ако **y** = 11111, тогава $S(\mathbf{y}) = 010$ и се извършва декодиране: 11111 – 00010 = 11101.

2. Ако **y** = 10011, тогава $S(\mathbf{y}) = 101$, което не се появява в таблицата, при което се предполага, че най-малко 2 грешки са се случили.

Тема 41. Циклични кодове

Основни дефиниции. Пръстен от полиноми. Генераторен и проверовъчен полином. Несистематично и систематично кодиране. Алгоритъм за декодиране.

1. Обща характеристика

Цикличните кодове са клас кодове с богата алгебрична структура и се прилагат ефективно с елементарни устройства, известни като *преместващи регистри*. Кодове, като бинарни Hamming кодове, Golay кодове, са еквивалентни на цикличните кодове.

Дефиниция: Кодът C е цикличен ако той е линеен код и всяко циклично преместване на елементите в кодовата дума е кодова дума, т.е. когато кодовата дума $a_0 a_1 \ldots a_{n-1}$ е в C, тогава и кодовата дума $a_{n-1} a_0 a_1 \ldots a_{n-2}$ е в C.

Пример:
1. Бинарният код {000, 101, 011, 110} е цикличен.

3. Пръстен от полиноми по модул $f(x)$

Нека $f(x)$ е фиксиран полином в $F[x]$. Два полинома $g(x)$ и $h(x)$ в $F[x]$ са *конгруентни по модул* $f(x)$, означени чрез $g(x) \equiv h(x) \pmod{f(x)}$, ако $g(x) - h(x)$ е делимо на $f(x)$.

В съответствие с разделителния алгоритъм всеки полином $a(x)$ в $F[x]$ е конгруентен по модул $f(x)$ на единичен полином $r(x)$ със степен по-малка от $\deg f(x)$; $r(x)$ е принципният остатък, когато $a(x)$ се разделя на $f(x)$. Означава се с $F[x]/ f(x)$ множеството от полиноми в $F[x]$ със степен по-малка от $\deg f(x)$, които се сумират и умножават по модул $f(x)$.

Пример: да се изчисли $(x + 1)^2$ в $F_2[x] / (x^2 + x + 1)$:

$(x + 1)^2 = x^2 + 2x + 1 = x^2 + 1 \equiv x \pmod{x^2 + x + 1}$.

Следователно, $(x + 1)^2 = x$ в $F_2[x]/ x^2 + x + 1$.

Извод: $F[x]/ f(x)$ е *пръстен от полиноми* (*върху* F) *по модул* $f(x)$.

Ако $f(x) \in F_2[x]$ е със степен n, тогава пръстена $F_q[x]/ f(x)$ се състои от полиноми със степен $\leq (n\text{-}1)$. Всеки от n-те коефициента на такъв полином принадлежи на F_q и, оттук

$|F_q[x]/ f(x)| = q^n$.

Таблиците за сумиране и умножение за $F_2[x]/ x^2 + x + 1$ имат вида

+	0	1	x	$1 + x$
0	0	1	x	$1 + x$
1	1	0	$1 + x$	x
x	x	$1 + x$	0	1
$1 + x$	$1 + x$	x	1	0

×	0	1	x	$1 + x$
0	0	0	0	0
1	0	1	x	$1 + x$
x	0	x	$1 + x$	1
$1 + x$	0	$1 + x$	1	x

Както се вижда от таблицата за умножение всеки ненулев елемент има мултипликативна инверсия и, следователно, $F_2[x]/ x^2 + x + 1$ е действително поле.

Дефиниция: Полиномът $f(x)$ е *редуцируем* (приводим) ако $f(x) = a(x) . b(x)$, където $a(x), b(x) \in F[x]$, ако deg $a(x)$ и deg $b(x)$ са по-малки от deg $f(x)$. В противен случай $f(x)$ е *нередуцируем* или *неприводим*.

Всеки моничен полином може да се разложи (факторизира) по единствен начин в произведение от нередуцируеми монични полиноми.

4. Циклични кодове

Полиномът $f(x) = x^n - 1$ има особено значение, тъй като пръстенът $F[x]/(x^n - 1)$ от полиноми по модул $(x^n - 1)$ е единствен, който може да се разглежда в контекста на цикличните кодове. За простота се означава $F[x]/(x^n - 1) = R_n$, където полето $F = F_q$ ще бъде под подразбиране.

Тъй като $x^n \equiv 1 \pmod{x^n - 1}$, всеки полином по модул $x^n - 1$ може да се редуцира чрез заместване на x^n с 1, x^{n+1} с x, x^{n+2} с x^2 и т.н.

Нека векторът $a_0 a_1 \ldots a_{n-1}$ в пространството $V(n, q)$ да се свърже с полинома

$$a(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}$$

в пръстена R_n, където n е броят на елементите на полето.

Цикличният код се разглежда като подмножество на $V(n, q)$ и като подмножество на пръстена R_n. Умножението на полинома $a(x)$ с x има като резултат

$$x.\ a(x) = a_0 x + a_1 x^2 + \ldots + a_{n-1} x^n = a_{n-1} + a_0 x + a_1 x^2 + \ldots + a_{n-2} x^{n-1},$$

което съответства на кода $a_{n-1}\ a_0\ a_1 \ldots a_{n-2}$. Следователно, умножението с x кореспондира на *реализацията на единично циклично преместване*. Умножението с x^m съответства на циклично преместване на m позиции.

ПРОВЕРОВЪЧЕН ПОЛИНОМ И ПРОВЕРОВЪЧНА МАТРИЦА

1. Основни дефиниции и теореми

Дефиниция: Нека C е цикличен $[n, k]$ – код с генераторен полином $g(x)$. В съответствие с теорема 3, $g(x)$ е фактор на $x^n - 1$, т.е. съществува полином $h(x)$, който е в сила

$$x^n - 1 = g(x)\, h(x)$$

Тъй като $g(x)$ е моничен полином, то и $h(x)$ е моничен полином. В съответствие с теорема 4, $g(x)$ има степен $n - k$, а $h(x)$ има степен k. Полиномът $h(x)$ се нарича *проверовъчен полином* на C.

2. Кодиране с циклични кодове

Определение: Цикличното кодиране е *систематично*, когато информационните символи са част от кодовата дума, а противен случай е *несистематично*.

***а*. Несистематично кодиране**

Нека C е цикличен код, над полето F_q с пораждащ полином $g(x)$, съдържащ q^n кодови думи с дължина n. Съобщението, което трябва да се кодира е вектор **u,** дефиниран с последователност от символи $u_0, u_1, ..., u_{k-1}$. На този вектор съответства полином съобщението от вида

$$u(x) = u_0 x^0 + u_1 x + ... + u_{k-1} x^{k-1} \text{ или } \quad u(x) = u_0 + u_1 x + ... + u_{k-1} x^{k-1}.$$

Пример: Да се запише информационният полином, ако е дадено съобщение като вектор - последователност от бинарни символи и извърши кодиране на съобщението.

Даден е векторът на съобщението **u** = 1 0 0 1 0 1 1,

където $u_0 = 1$, $u_1 = 0$, $u_2 = 0$, $u_3 = 1$, $u_4 = 0$, $u_5 = 1$, $u_6 = 1$.
Тогава информационният полином има вида

$$u(x) = 1 + x^3 + x^5 + x^6.$$

Кодирането на вектора-съобщение се извършва чрез произведението $g(x).u(x)$, което дефинира полином на кодираното съобщение, т.е.

$C(x) = g(x).u(x)$

Декодирането на предадения вектор на кодираното съобщение се извършва на базата на кодовия полином и генераторния полином. При известен кодов полином $C(x)$ и генераторен полином $g(x)$, информационният полином се определя от израза

$u(x) = C(x)/g(x)$.

***б*. Систематично кодиране**

Даден е цикличен $[n, k]$ – код C и информационен полином

$$u(x) = u_0 + u_1 x + ... + u_{k-1} x^{k-1}.$$

Като се отчете, че при систематично кодиране последователността от информационни символи заема старшите елементи на кодовата дума, кодовият полином се представя във вида

$$C(x) = t(x) + x^{n-k} u(x),$$

където $t(x)$ е полином със степен по-малка от $(n - k)$.

Известни са два подхода на систематично кодиране: чрез генериращ полином и чрез проверовъчен полином.

- **Систематично кодиране чрез генериращ полином**

Нека $g(x)$ е генериращ полином на кода C. Тогава $g(x)$ дели кодовия полином $C(x) = t(x) + x^{n-k} u(x)$ без остатък. Допуска се, че полиномът $x^{n-k} u(x)$ се дели на $g(x)$ с частно $q(x)$ и остатък $r(x)$, т.е.

$x^{n-k}u(x) = q(x)g(x) + r(x)$

Тогава кодовият полином $C(x)$ се записва във вида

$c(x) = r(x) + t(x) + q(x)g(x)$

По определение $r(x) + t(x) = 0(\mathrm{mod}\, g(x))$, откъдето следва $t(x) = r(x)(\mathrm{mod}\, g(x))$.

Пример:

Нека C е $[n, k]$ – цикличен код с генериращ полином $g(x) = g_0 + g_1 x + \ldots + g_{n-k} x^{n-k}$. Да се кодира съобщението, дефинирано с вектора $\mathbf{u} = u_0 u_1 u_2 \ldots u_{k-1}$.
В съответствие с вектора на съобщението информационният полином има вида
$u(x) = u_0 + u_1 x + \ldots + u_{k-1} x^{k-1}$.

Дефинира се полиномът $f(x) = x^{n-k}u(x)$. Извършва се делението на полиномите $f(x) = x^{n-k}u(x)$ и $g(x)$ и се изчислява остатъкът $r(x) = t(x)$. Определя се кодовият полином чрез уравнението

$C(x) = t(x) + f(x)$.

Коефициентите на полинома $C(x)$ дефинират кодовата дума $[c_0 c_1 c_2 \ldots c_{n-1}] \in C$.

- **Систематично кодиране чрез проверяващ полином**

Допуска се, че C е цикличен (n, k) – код с генериращ полином $g(x)$ със степен $(n - k)$. Проверовъчният полиномна кода C се определя от полиномното отношение $h(x) = \dfrac{x^n - 1}{g(x)}$. Елементите на полинома $h(x)$ дефинират проверовъчната матрица H на кода C

$$H = \begin{bmatrix} h_k & h_{k-1} & \ldots & h_0 & 0 & 0 & \ldots & 0 \\ 0 & h_k & h_{k-1} & \ldots & h_0 & 0 & \ldots & 0 \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & 0 \\ 0 & . & 0 & h_k & h_{k-1} & . & . & h_0 \end{bmatrix}.$$

По определение всеки кодов вектор $\mathbf{c} = [c_0 c_1 c_2 \ldots c_{n-1}] \in C$ е ортогонален на редовете на матрицата H, т.е. $H \,.\, \mathbf{c}^{\mathrm{T}} = 0$, откъдето за всяко $i = \overline{0, n-k-1}$ следва

$c_i h_k + c_{i+1} h_{k-1} + \ldots + c_{i+k} h_0 = 0$.

При $h_k \neq 0$ и известни координати $c_{i+1}, c_{i+2}, \ldots, c_{i+k}$ на вектора $\mathbf{c}$ може да се определи елементът c_i.

Пример:

Даден е [*n*, *k*] – цикличен код, проверивъчният полином *h*(*x*) и векторът на съобщението **u** = $u_1 u_2 \ldots u_k$. Да се извърши систематично кодиране на **u**.

По дефиниция елементите на вектора **u** заемат старшите *k* позиции на елементите $c_{n-k}\, c_{n-k+1} \ldots c_{n-1}$ на кода **c**, т.е. **c** = $[c_0, c_1, \ldots, c_{n-k-1}, u_1, u_2, \ldots, u_k]$. Елементите $c_0\, c_1 \ldots c_{n-k-1}$ се определят от следното уравнение, дефинирано за обобщаващия елемент c_{n-k-j}

$c_{n-k-j} = -h_k^{-1}\left(c_{n-k-j+1}h_{k-1} + c_{n-k-j+2}h_{k-2} + \ldots + c_{n-k-j+k}h_{k-k}\right)$ или

$$c_{n-k-j} = -h_k^{-1}\left(c_{n-k-j+1}h_{k-1} + c_{n-k-j+2}h_{k-2} + \ldots + c_{n-j}h_0\right)$$

където $j = \overline{1, n-k}$.

Умножението с h_k^{-1} се прилага при кодиране с не бинарен код.

3. Декодиране на циклични кодове

Даден е [*n*, *k*] – цикличен код *C* над полето F_q с генериращ полином *g*(*x*). Формира се съкратената таблица на синдромите на комножества (съседни множества), дефинирани с полиномите *a*(*x*)+c(*x*). Те се характеризират с лидери, чиито полиноми *a*(*x*) са с коефициенти пред x^{n-1}, различни от нула. По определение всеки кодов полином *c*(*x*), съответстващ на кодов вектор от цикличния код *C*, се дели на генераторния полином *g*(*x*) без остатък. Освен това, полиномите на произволни кодови думи *a*(*x*)+c(*x*), които имат един и същ остатък при делене на генераторния полином *g*(*x*), принадлежат на едно и също комножество. Следователно синдромът с полином *s*(*x*) на комножество е остатъкът от деленето на полинома на *a*(*x*) на *g*(*x*), т.е.

a(*x*) =*q*(*x*)*g*(*x*) + *s*(*x*)

където степента на *s*(*x*) е по-малка от (*n* – *k*). Броят на комножествата в пространството от *n* – мерни вектори на цикличен [*n*, *k*] – код *C* е равен на q^{n-k}

Пример:

Допуска се, че по информационния канал е изпратена кодова дума **c** с дължина *n*. Получената кодова дума е **y** = **c** + **e**, където **e** е векторът на грешката, причинен от шума в канала. Да се декодира полученият вектор **y**, ако е известен генериращият полином *g*(*x*).

От израза т.е. *s*(*x*) = *y*(*x*)/*g*(*x*) се пресмята синдромът S(**y**) на **y,** елементите на който са коефициентите на *s*(*x*), и се сравнява със синдромите от таблицата. Ако S(**y**) е синдром от таблицата, грешката се поправя. Ако S(**y**) не е синдром от таблицата, то у циклично се завърта и отново се изчислява синдромът на новополучената от цикличното завъртане кодова дума. Този синдром отново се сравнява със синдромите от таблицата. Ако синдромът не е в таблицата, процедурата се повтаря, докато не се изчисли синдром, намиращ се в таблицата.

Допуска се, че са направени *r* циклични завъртания на приетия вектор **y** и е изчислен *r*-ят синдром. Нека лидерът на този синдром е $\mathbf{e}_r$. Грешката $\hat{\mathbf{e}}_r$ в кодовата дума ще се получи, ако

лидерът $\mathbf{e}_r$ циклично се завърти точно $n - r$ пъти. Предадения вектор **c** по канала ще се получи от уравнението $\mathbf{c} = \mathbf{y} - \hat{\mathbf{e}}_r$.

Тема 43. Кодове на Хеминг.

Определение. Кодиране и декодиране с кодове на Хеминг.

1. Дефиниция и характеристика на кодовете на Hemming

Кодовете на Hemming са фамилия кодове за корекции на единични грешки. Това са линейни кодове върху крайно поле $GF(q)$. Бинарните Hamming кодовете се дефинират с тяхната проверовъчната матрица:

Дефиниция: Нека r е положително цяло число и H е $r \times (2^r - 1)$ матрица, чиито колони са ненулеви вектори на $V(r, 2)$. Тогава кодът с проверовъчна матрица H се нарича бинарен Hamming код и се означава чрез Ham$(r, 2)$.

1. Кодът Ham$(r, 2)$ има дължина $n = 2^r - 1$ и размер $k = n - r$. Следователно $r = n - k$ е броят на проверовъчните символи във всяка кодова дума и е известно като *излишък* на кода.
2. Колоните на H могат да се вземат в произволен ред и кодът Ham$(r, 2)$ при зададен излишък r е всеки един от определен брой еквивалентни кодове.

Примери:

1. Дадено е $r = 2$: $H = \begin{bmatrix} 110 \\ 101 \end{bmatrix}$ - проверовъчна матрица.

В съответствие с теоремата относно стандартната форма на генераторната матрица, следва, че G = [111]. Следователно, Ham(2, 2) е бинарен троен код с повторение.

2. Дадено е $r = 3$. Проверовъчната матрица с размер $[n - k, n]$

за Ham(3, 2) има вида $H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$

В матрицата H колоните са представени в естествен ред на нарастващи бинарни числа от 1 до 7. За да се получи H в стандартна форма колоните се представят в друг ред.

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

В съответствие с теоремата относно стандартната форма на генераторната матрица, следва, че генераторната матрица с размер $[k, n]$ за Ham(3, 2) има вида

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Кодът Ham(3, 2) еквивалентен на перфектен [7, 4, 3]-код.

Теорема 1: Бинарният Hamming код Ham(r, 2) при $r \geq 2$:
1) е [$2^r - 1$, $2^r - 1 - r$]-код;
2) има минимално разстояние 3 (следователно това код за корекция на единична грешка в кодовата дума);
3) е перфектен код.

2. Декодиране с бинарни Hamming кодове

Тъй като Ham(r, 2) е перфектен код за корекция на единична грешка, лидерите на комножествата са точно $2^r = n + 1$ вектори на $V(n, 2)$ с тегло ≤ 1.

Синдромът на вектора 0 ... 010 ... 0 (с единица на j-та позиция) е (0 ... 010 ...0).H^T, което е само транспозиция на j-та колона на H.

Следователно, ако колоните на H са подредени в ред на нарастващи бинарни числа, т.е. j-та колона на H е бинарно представяне на числото j, то може да се приложи следният алгоритъм.

1. Ако се приема векторът **y**, то се изчислява неговият синдром $S(\mathbf{y}) = \mathbf{y}\,H^T$.
2. Ако $S(\mathbf{y}) = 0$, тогава се приема, че е изпратена кодовата дума **y**.
3. Ако $S(\mathbf{y}) \neq 0$, тогава, ако се приеме единична грешка, $S(\mathbf{y})$ дава бинарното представяне на позицията на грешката, т.е. грешката може да бъде коригирана.

Например, нека да е дадена матрицата $H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$.

Ако приетият вектор е **y** = 1101011, тогава $S(\mathbf{y})$ = 110, показвайки грешка в шестата позиция. Тогава **y** може да се декодира като 1101001.

3. Разширени бинарни Hamming кодове

Разширен бинарен Hamming код $\hat{\text{Ham}}(r,2)$ е код получен от Ham(r, 2) чрез добавяне на общи проверовъчни елементи.

Минималното разстояние нараства от 3 на 4. Разширеният код е линеен, оттук $\hat{\text{Ham}}(r,2)$ е [2^r, 2^r - 1- r, 4]-код. Разширеният код $\hat{\text{Ham}}(r,2)$ е не по-добър от Ham(r, 2), когато се използва за декодиране. Допълнителните цифри за всяка кодова дума намалява скоростта на предаване на информацията. При минимална дистанция 4, $\hat{\text{Ham}}(r,2)$ е изключително подходящ за не пълно декодиране. Той може едновременно да коригира всяка единична грешка и да открие двойна грешка.

Нека H е проверовъчна матрица за Ham(r, 2). Проверовъчната матрица $\overline{H}$ за разширения код може да се получи от H чрез преобразуването

$$H \to \overline{H} = \begin{bmatrix} & & 0 \\ & & 0 \\ & H & \ldots \\ & & 0 \\ 11 & \ldots & 11 \end{bmatrix}.$$

Последният ред дефинира обобщеното проверовъчно уравнение върху кодовите думи, т.е. $x_1 + x_2 + \ldots + x_{n-1} = 0$.

Ако H се представи с колони с нарастващ ред на бинарните номера, съществува точен декодиращ алгоритъм, илюстриран за $r = 3$, за коригиране на единична грешка и да открие двойна грешка.

Пример: Кодът $\text{Hâm}(3, 2)$ има следната проверовъчна матрица

$$\overline{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Синдромът на вектора на грешката 00 … 010 … 0 (с единица на j-та позиция) е транспонираната j-та колона на $\overline{H}$.

Непълен декодиращ алгоритъм

Допуска се, че е приет вектор $\mathbf{y}$. Синдромът се определя с израза $S(\mathbf{y}) = \mathbf{y}H^T$. Предполага се, че $S(\mathbf{y}) = (s_1, s_2, s_3, s_4)$. Тогава:

1. Ако $s_4 = 0$ и $(s_1, s_2, s_3, s_4) = \mathbf{0}$, приема се, че няма грешки.
2. Ако $s_4 = 0$ и $(s_1, s_2, s_3) \neq \mathbf{0}$, допуска се, че са се случили най-малко две грешки, което изисква ретрансмисия на съобщението.
3 Ако $s_4 = 1$ и $(s_1, s_2, s_3) = \mathbf{0}$, приема се, че единична грешка има на последната позиция в кодовата дума.
4. Ако $s_4 = 1$ и $(s_1, s_2, s_3) \neq \mathbf{0}$, приема се, че единична грешка е допусната на j-та позиция, където j е номерът на позицията, чийто бинарно представяне е (s_1, s_2, s_3).

Тема 44. Методи за повишаване надеждността на компютърните системи. Резервиране. Нива на резервиране и отказоустойчивост

Класификация на методите за повишаване на надеждността

Производителността на компютърните системи (КС) се намира в пряка зависимост от надеждността на работата им. Поради това особено важно значение има повишаването на надеждността на КС като цяло.

Съществуващите методи за повишаване на надеждността могат да се разделят на три групи, ако като показател се вземат *етапите на съществуване на техническите обекти*:

- методи за повишаване на надеждността на КС в процеса на проектирането им - на този етап се *залага* потенциалната надеждност;
- методи за повишаване на надеждността на КС в процеса на производството им - на този етап се реализира заложената надеждност;
- методи за повишаване на надеждността на КС в процеса на експлоатацията им - на този етап се *поддържа* осигурената на предните етапи надеждност.

На етапа на ***проектиране*** се изпълняват дейности като:

- разработване на по-прости схеми на модули, възли, блокове и КС като цяло, с минимален брой запоявания;
- използване на конструктивни елементи с високи надеждностни показатели;
- разработване на схеми с използване на унифицирани възли и модули, проверени за надеждност в други КС;
- използване на методи на резервиране (въвеждане на структурен излишък) на различни конструктивни нива (комплектуващ елемент, функционален елемент, модул, устройство);
- анализиране на отказите, като се определят и описват възможните първични откази на различни нива и влиянието им върху техническите характеристики на КС като цяло;
- съставяне на методи за търсене, намиране и отстраняване на отказите;
- проектиране на система за контрол и диагностика и определяне на вероятностните й характеристики.

На етапа ***производство*** основните дейности включват:

- тренировка на елементите, възлите, модулите и устройствата;
- използване на съвременни технологични процеси;
- подобряване качеството на входния, текущия и изходящия контрол на изделията по надеждност.

На етапа ***експлоатация*** се изпълняват дейности като:

- замяна на елементи по време на профилактичните работи;
- пропускане на контролни и диагностични тестове от специализирани контролни пакети;
- ремонт (настройка или замяна) на елементи, в които са открити неизправности.

Основно средство за повишаване на надеждността е въвеждането на ***излишък.*** Увеличаването на количествените показатели на надеждността се свързва със следните видове излишък:

- структурен;
- функционален (алгоритмичен);
- информационен;
- излишък по време;
- излишък по натоварване.

Често използван в инженерната практика е ***структурния*** излишък. Той представлява едно от основните средства при проектирането на отказоустойчиви системи (fault-tolerant systems) с цел постигане на висока експлоатационна надеждност.

Резервиране

Резервиране се нарича въвеждането в структурата на устройството на допълнителен брой елементи, функционални блокове и връзки, в сравнение с минимално необходимите за изпълнение на зададени функции при зададени условия на работа.

По начина на свързване на резервните елементи с основните се различават *три* вида резервиране:

- постоянно;
- чрез заместване;
- пълзящо.

Постоянно се нарича такова резервиране, при което резервните елементи са присъединени към основните в продължение на цялото време на работа и се намират в еднакъв за тях работен режим.

Обикновено постоянното резервиране предполага *натоварено* състояние на резервните елементи (*горещ резерв*).

Резервиране ***чрез заместване*** се нарича такова резервиране, при което резервните елементи заменят основните след отказа им.

Обикновено резервирането чрез заместване предполага *ненатоварено* състояние на елементите (*студен резерв*). Заместването може да стане автоматично и ръчно.

При резервирането (постоянно или чрез заместване) се различават три начина на свързване на елементите:

- последователно;
- паралелно;
- смесено.

Последователният начин се използва когато преобладават откази от типа *късо съединение*. Тогава

$$P_{посл.} = \prod_{i=1}^{n} p_i ,$$

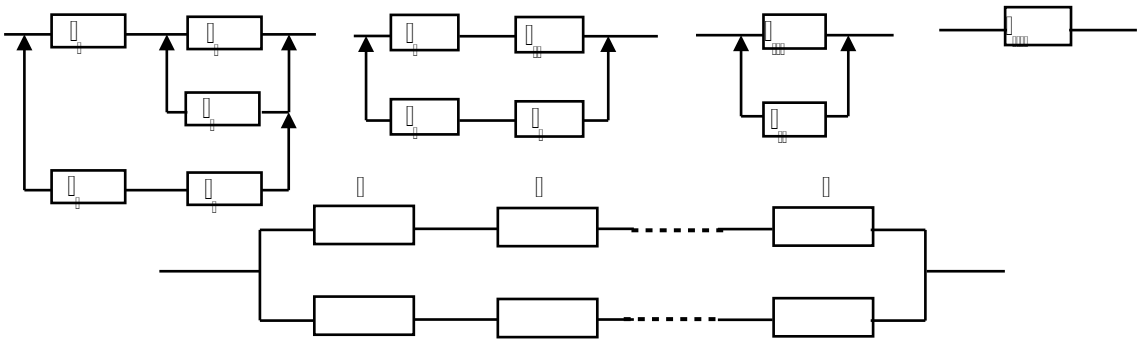
където: p_i-вероятността за безотказна работа на отделните елементи.

Паралелният начин се използва, когато преобладават откази от типа *прекъсване*. Тогава

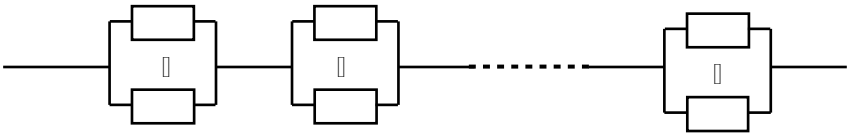
$$P_{пар.} = 1 - \prod_{j=1}^{m} \left(1 - p_j\right).$$

Смесеният начин се използва, когато и двата вида откази са равновероятни. $P_{СМ}$ се определя като се извърши постепенно опростяване на схемата (фиг. 1.7):

фиг. 1.7 Постепенно опростяване на схема



фиг. 1.8 Общо резервиране



фиг. 1.9 Разделно резервиране

Общо резервиране е това, при което се резервира като цяло апаратура, функционален блок, възел, модул.

Разделното резервиране е това, при което устройството, функционалния блок, възела се резервират на отделни участъци.

Разделното и общото резервиране могат да се осъществяват както чрез постоянно включване, така и чрез заместване.

При сравнение на резервиране при натоварен и ненатоварен резерв чрез аналитични изрази е направен извода, че при равни други условия, система с ненатоварен резерв е по-надеждна от система с натоварен резерв.

Ненатовареният резерв (във вид на процесор, компютър) за разлика от натоварения, изисква някакво допълнително време за зареждане в него на необходимите данни (състояние на регистрите, константи, програми). Затова ненатовареният резерв не може мигновено да продължи работата на отказалата система. Освен това, в много случаи е важно да се запазят междинните резултати, съществуващи в момента на отказа и следователно отсъстващи в ненатоварения резерв. В такива случаи често се използва *комбинирано* резервиране, т.е. първата резервна подсистема работи в режим на натоварен резерв (горещо резервиране), напълно дублира цялата информация, а останалите подсистеми - като ненатоварен резерв (студено резервиране).

При ***пълзящото*** резервиране всеки резервен елемент може да замени всеки основен елемент. За осъществяването му е необходимо устройство, което автоматично да намира неизправния елемент и да включва вместо него резервния.

Нива на резервиране в компютърните системи

Универсални КС. В универсалните КС се среща резервиране на няколко различни нива:

- компютър;
- устройство;
- кодове.

Резервирането на ***ниво компютър*** се свежда до наличието на повече еднотипни компютри, отколкото са необходими за решаване на поставените задачи. В такъв случай надеждността на системата се оценява като система с пълзящ резерв.

Резервирането на ***ниво устройство*** се въвежда в по-ниските нива на йерархия в структурата на универсалните КС. Най-често се изпълнява като резервиране на ниво периферни устройства (ПУ). В съвременните КС има като правило няколко запомнящи устройства на магнитни дискове, оптични дискове, няколко устройства за въвеждане, няколко устройства за извеждане на информация. Ако е необходимо да се получи показател за надеждността на системата като цяло се допуска, че за решаване на задачите се изисква някакво минимално количество ПУ от даден тип, а останалите ПУ представляват *пълзящ резерв*.

Резервирането на ***ниво кодове*** е широко използван метод на резервиране в КС. Използват се кодове с откриване и коригиране на грешки. Употребата на тези кодове дава възможност да се поправят определен брой грешки в каналите за предаване на данни или да се възстанови информацията в случай на отказ на някои клетки в ОП и ПЗУ или пътечките в ЗУМЛ или ЗУМД. Надеждността на такива устройства се оценява като надеждност на резервирани системи с *пълзящ резерв* ..

Отказоустойчиви компютърни системи

Отказоустойчивост - свойство на архитектурата на КС, което позволява на потребителя (в случай на универсална КС) или на функционалната програма (в случай на управляваща КС) да продължи работа и тогава, когато в апаратните или програмните средства възникнат откази.

По начина на реализация, отказоустойчивостта се дели на активна и пасивна**.**

Активна отказоустойчивост - базира се на три процеса:

- откриване на отказа;
- локализация на отказа;
- реконфигурация на системата.

Отказите се откриват с помощта на средства за контрол, локализират се с помощта на средства за диагностика и се отстраняват *автоматично* чрез реконфигурация на системата (преустройство на структурата на изчислителните средства на системата по такъв начин, че нейните отказали части да се отстранят от участие в работата й).

Пасивна отказоустойчивост - системата не губи своите функционални свойства в случай на отказ на отделни нейни елементи, тъй като отказът се *маскира* от въведения структурен излишък. Пример за пасивна отказоустойчивост е системата с мажоритарен орган. На ниските нива на йерархията на апаратурата отказоустойчивостта се реализира също чрез логически схеми с преплитане. Пасивната отказоустойчивост е свързана с увеличението на количеството на апаратурата няколко пъти и се употребява в случаите, когато КС изпълнява особено отговорни функции и тогава, когато са недопустими даже кратки прекъсвания в работата на КС, а както и за осигуряване на безотказност на най-важните блокове или устройства на КС.

Използването на активната отказоустойчивост се характеризира с по-икономичен разход на апаратни средства отколкото при пасивния подход. Това обаче е свързано с някаква загуба на

време при възстановяване работата на системата след отказа, а както е възможна и загубата на някаква част от данните. Активната отказоустойчивост е реализируема само в мултипроцесорните системи. В същото време използването на пасивната отказоустойчивост гарантира практически непрекъсната работа на КС и съхранение на цялата информация. Тези обстоятелства определят областите на използване на активната и пасивната отказоустойчивости. Първата се прилага в отговорни, но много сложни и скъпоструващи мултипроцесорни системи, където е важна икономията на употребяваните апаратни средства, но допускащи кратковременни прекъсвания в работата.

Тема 45. Контрол при предаване и съхраняване на информацията в компютрите. Основен принцип на схемния контрол.Код по четност/нечетност.

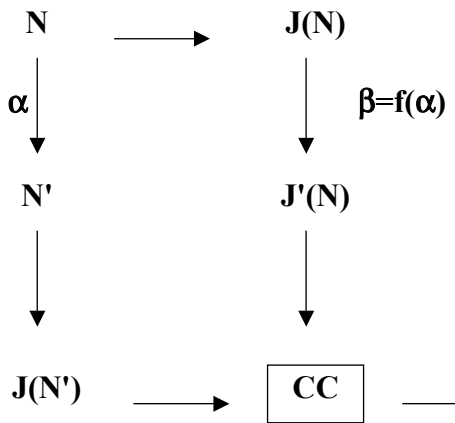
Основен принцип на схемния контрол

За откриване на грешки чрез схемен контрол се използва *кодиране* на информацията в КС, като към машинните думи се прибавя *информационен излишък (контролен признак).* Основната функция на информационния излишък е да раздели обработваните думи на определен брой класове. Попадането на дадена дума в друг клас, а не в този, който се посочва в информационния й излишък, е индикация за грешка.

Схемите за контрол *(апаратен излишък)* имат задачата:

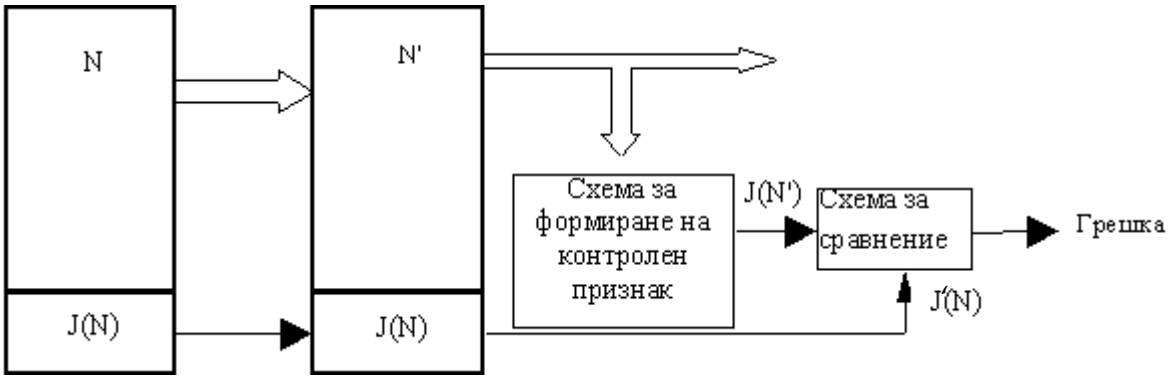
- да формират контролен признак за всяка дума от машината, т.е. да извършат кодиране на информацията;
- при обработка на думите в машината да откриват грешка, като следят съответствието между очаквания и получения клас.

Нека е дадена комбинация от *N* бита. (фиг. 2.1) При обработването на тези *N* бита се изработва по определено правило контролен признак *J(N).* Ако се извършва операция α над операнда *N,* при което се получава резултат *N'*, над контролния признак *J(N)* се извършва операция $\beta = f(\alpha)$, при което се получава *очакваният* контролен признак на резултата *J'(N).* Освен това след изпълнение на операцията α схемите за контрол изработват независимо контролен признак *J(N')* на комбинацията битове *N',* който се нарича *получен* контролен признак на резултата. Същността на откриване на грешка се състои в сравнение на така получените два контролни признака на резултата (очаквания *J'(N*) и получения *J(N')),* като тяхното несъвпадение е индикация за грешка в операцията.



Принципът на схемния контрол може да се илюстрира със следните два примера:

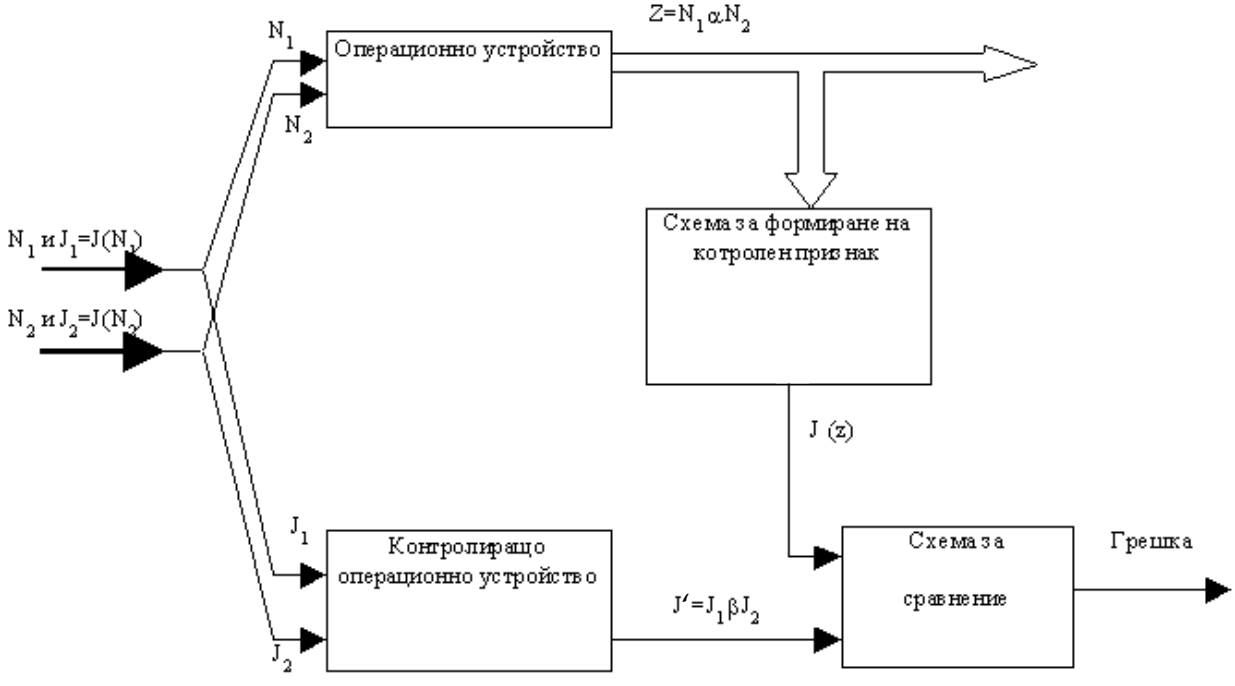
Пример 1: Нека се изпълнява микрооперация "прехвърляне на операнд *N* от регистър *A* в регистър *B*". В регистър *A* е поместен операндът *N* заедно с контролният си признак *J(N)*. За да се осъществи проверката на правилното изпълнение на микрооперацията, се формира контролен признак на резултата *J(N')* - получен контролен признак. Това се извършва от схема за формиране на контролен признак. Ако се приеме, че в регистър *A* контролният признак е правилен, може да се каже, че той е очакваният признак на резултата, тъй като микрооперацията е *прехвърляне без обработка*. Схемата за сравнение проверява съвпадението на получения след микрооперацията контролен признак *J(N')* с очаквания *J'(N)*. Несъвпадението е индикация за грешка.



Фиг.1 Контрол при предаване на операнди между два регистъра

Пример 2: Нека в операционно устройство се извършва микрооперация α върху

Фиг.2 Контрол при изпълнение на операция над два операнда



операндите N_1 и N_2 (фиг. 2.3), при което се получава резултатът $z = N_1 \alpha N_2$. Операндите се съпровождат от контролните си признаци $J_1 = J(N_1)$ и $J_2 = J(N_2)$. Операционното устройство за контрол извършва операция $\beta = f(\alpha)$ върху $J(N_1)$ и $J(N_2)$, като се получава очакваният признак на резултата $J'(J_1,J_2) = J(N_1)\beta J(N_2)$. Схемата за формиране на контролен признак изработва признак $J(z)$ на получения след микрооперацията α резултат z, така наречения получен контролен признак. Сигнал за грешка се формира при несъвпадение на двата контролни признака - очаквания и реално получения.

Пълният схемен контрол представлява съвкупност от подобни контролиращи схеми, проверяващи всички микрооперации. В резултат от проверката се изработва *общ сигнал за грешка (машинна грешка).* В повечето КС при откриване на грешка в контролиращите схеми за отделните микрооперации се зарежда и съответен тригер от специален регистър, наречен *регистър на грешките*. Съдържанието на регистъра на грешките може да служи за изходна информация на системата за автоматична диагностика.

Известно е, че прекъсванията в КС се реализират на приоритетни нива. С най-висок приоритет, т.е. *веднага* се обслужва заявка за прекъсване по машинна грешка.

Контролни признаци

Формирането на контролни признаци се основава на използването на кодове за откриване на грешки.

Кодирането е предназначено да формира за всяка комбинация от битове N контролен признак $J(N)$, чрез който N може с известна вероятност (наричана още *разрешаваща способност*) да се отдалечи от всички други комбинации. Основните изисквания, които се предявяват към кодовете са следните:

- кодът трябва да открива най-често срещаните видове грешки;
- разрешаващата способност на кода трябва да се достига с минимален брой разряди за контролен признак;
- формирането на контролния признак и създаването на контролиращите схеми трябва да се извършва с минимален апаратен излишък.

За съжаление тези изисквания са противоречиви, тъй като увеличаване на разрешаващата способност се постига с по-големи контролни признаци, което от своя страна довежда до увеличаване на контролиращата апаратура. В КС се използват предимно кодове, които откриват еднократни грешки, тъй като надеждността на логическите елементи е висока и вероятността за поява на многократна грешка е много по-малка от вероятността за поява на единична грешка. Това не е валидно при обработка на големи масиви информация, както например във външните запомнящи устройства, където възникват пакети от грешки. В такива случаи се използват специални кодове за откриване и коригиране на многократни грешки: цикличен, код на Файер и др.

При схемния контрол *най-често* се използваните кодове за формиране на контролни признаци са код по четност/ нечетност, код на Хеминг, цикличен (CRC) код, остатък по модул.

Схемен контрол при предаване и съхраняване на информацията с използване на *код по четност/нечетност*

При контрола по четност откриването на грешката се основава на нарушаване на четността на разрядите на думата. Това означава, че приетото кодиране изисква всички думи от кода да имат четен или нечетен брой единици. Нарушаването на това условие означава наличие на грешка. При контрола по четност/нечетност трябва да се формира кодова дума чрез добавяне на контролния бит и след това да се контролира спазването на условието за четност или нечетност на единиците в кодовата дума. Това се осъществява със схеми за четност или схеми за нечетност. За по-кратко контролът но четност и контролът по нечетност най-често се наричат *контрол по четност* (*parity check*)**.**

Схеми за определяне на четността при предаване на информацията в паралелен код

Схемата, реализираща обща четност/нечетност може да се построи като последователна или пирамидална. И в двата случая тя се състои от елементарни схеми, определящи четност/нечетност на два двоични разряда.

Елементарната схема, определяща четността на два разряда се синтезира с методите на Булевата алгебра, използвайки таблицата на истинност на функцията четност/нечетност . Стандартното означение на функциите четност/нечетност е *P*.

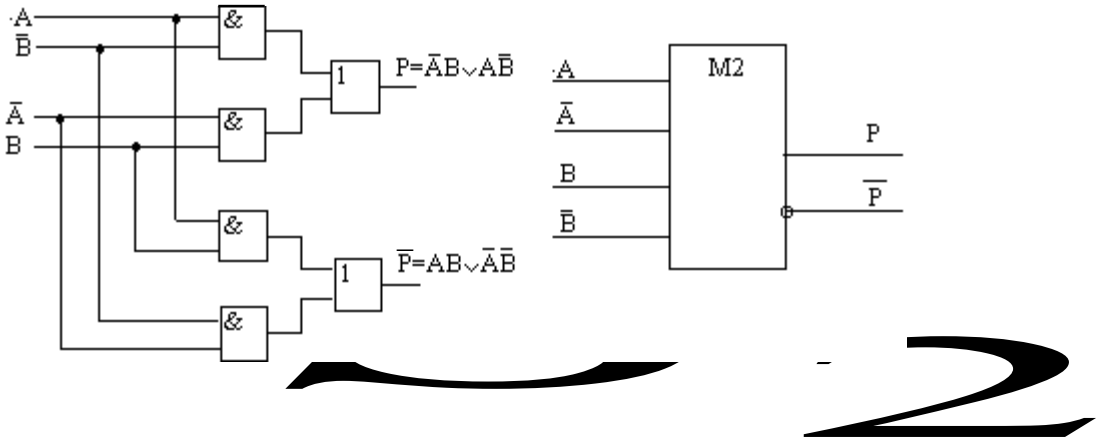
A	B	P	$\overline{P}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

От дизюнктивната нормална форма на функциите четност/нечетност

$$P = \overline{A}B \vee A\overline{B}$$

$$\overline{P} = \overline{A}\overline{B} \vee AB,$$

която е и минималната форма , се преминава към реализацията на логическата схема.:

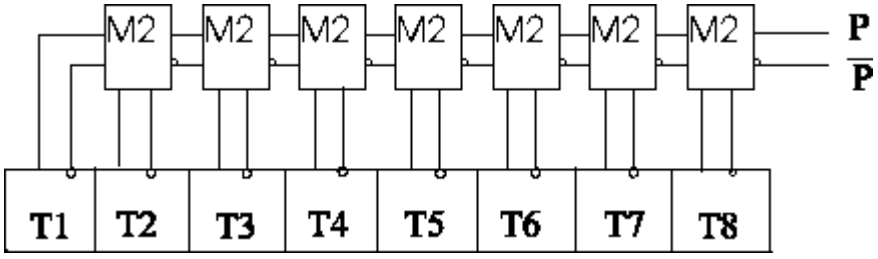


Тази елементарна схема се използва при кодиране и декодиране с код по четност на произволна *n*-разрядна дума, предавана в паралелен код. Схемите за кодиране и декодиране са едни и същи. При *кодиране* на входа на кодиращата схема (КС) се подават само

Фиг.4 Кодираща и декодираща процедура

информационните разряди на думата, а изхода на КС се подава към контролният разряд При *декодиране* се сравняват полученият от кодиращата схема бит с контролният бит и при *съвпадението* им се сигнализира за грешка (фиг.4).

Схемата за кодиране/декодиране може да се изгради като *последователна* или *пирамидална*

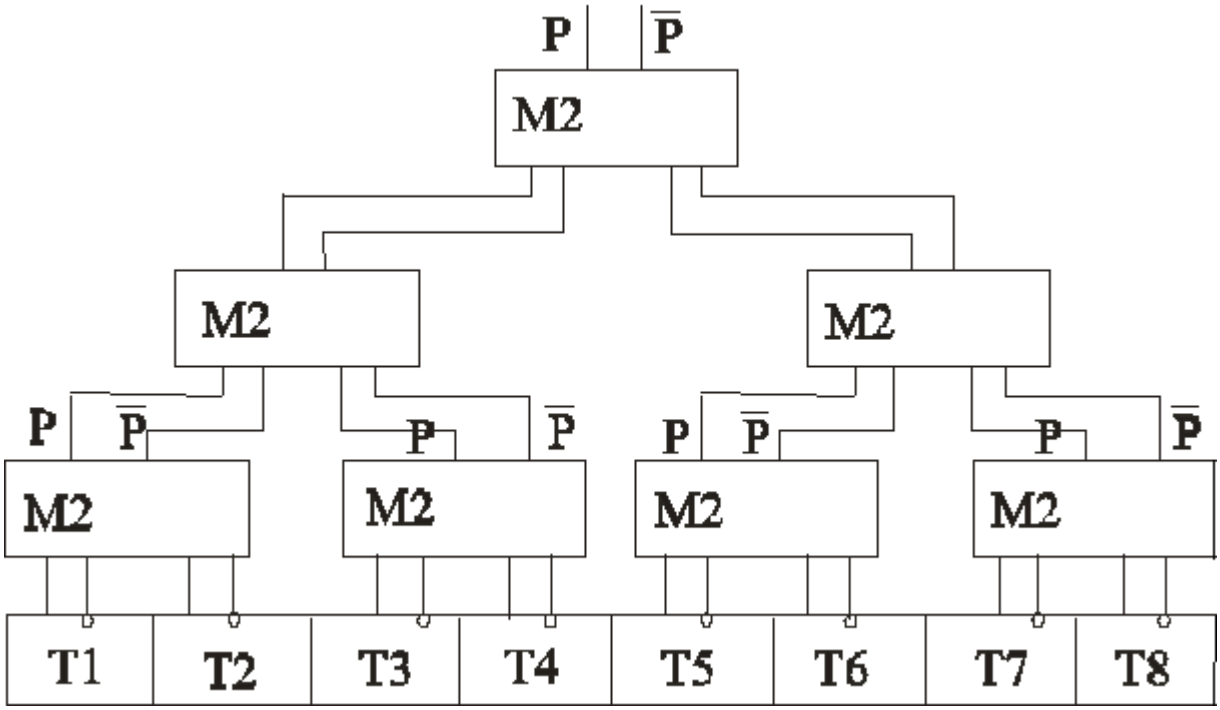


фиг. 5 Последователна схема за кодиране

фиг. 6 Пирамидална схема за кодиране

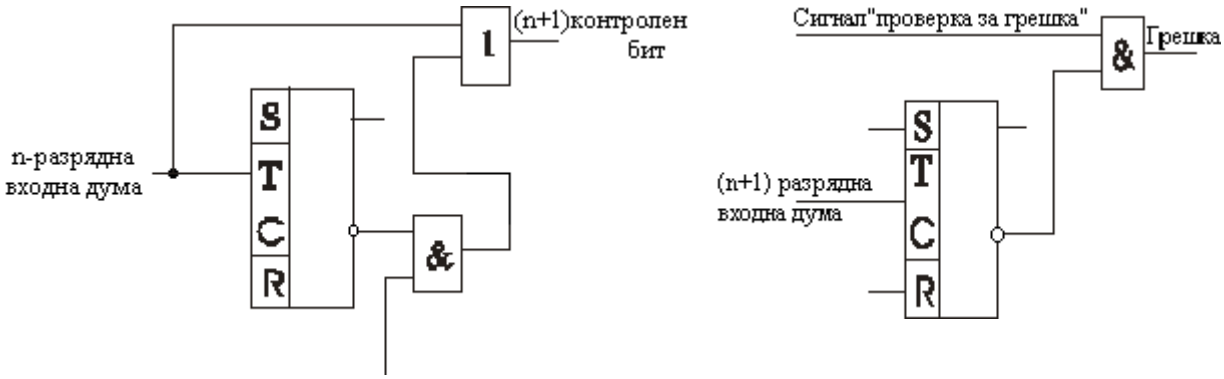
Сложността на последователната схема и закъснението за изработване на контролният бит са пропорционални на броя на информационните разряди.

Пирамидалните схеми са по-сложни, но с по-голямо бързодействие от последователните, тъй като закъснението за формиране на контролният бит се определя от броя на стъпалата, а техният брой е значително по-малък от броя на информационните разряди.



Схеми за определяне на четността при предаване на информацията в последователен код

За определяне на четност/нечетност на операнд, чиито разряди постъпват последователно, може да се използва тригер с броячен вход, който реализира функцията *нечетност*. Когато на тригера се подава единица той изменя състоянието си. Ако двоичните разряди, последователно постъпващи на броячният вход на тригера съдържат нечетен брой единици, тригерът ще изменя своето състояние нечетен брой пъти и в резултат ще се установи в състояние, противоположно на изходното. Тригерът предварително се нулира.



Фиг.7 Схема за кодиране и декодиране

При ***кодиране*** на входа на тригера постъпва *n*-разрядната информационна дума, а на изхода на схемата при *n+1* тактов импулс се формира контролен бит.

При ***декодиране*** се подават информационните разряди, включително и контролният разряд. При контрол по нечетност, ако няма грешка в предаването, тригерът винаги трябва да остане в състояние *1* (при четност - в *0*). Сигналът за откриване на грешката постъпва, след като през тригера премине цялата информация.

Тема 46. Контрол при обработване на информацията. Числов и цифров контрол по модул. Контрол по модул на аритметични и логически операции.

Дефинирани са два вида контрол по модул:

- числов;
- цифров.

Числов контрол по модул

Въвежда се информационен излишък чрез добавяне на контролни разреди (*k* на брой) към информационните (*m* на брой). Общият брой на разрядите на кодираната дума е $n = m+k$. Съдържанието на контролните разряди е равно на остатъка r_N от делението на дадено число *N* на определено положително число *q*, наречено *модул*.

Числовият контрол по модул често се нарича *остатъчен код.*

Броят *k* на контролните разреди трябва да бъде така подбран (а това зависи от избора на модула *q*), че да не намалява съществено бързодействието на контролираната схема, като същевременно се постига искането увеличаване на вероятността за откриване на еднократни и многократни грешки.

Нека са зададени две числа: *m*-разредно контролирано число *N* и модул *q*. При делението на числото *N* на модула *q* се получава частно *c* и остатък r_N (означава се още с r(N)), които са свързани със съотношението:

$$N = c.q + r_N$$

Всички числа са цели, като *q* и *r* са винаги положителни.

Съотношението може да се запише още във вида $N \equiv r_N \bmod q$ или

$$r_N \equiv N \bmod q, \qquad (2.4)$$

което се нарича сравнение: *N* е сравнимо с остатъка *r* по модул *q*.

В общия случай се казва, че две числа *A* и *B* са сравними по модул *q*,когато при делението им на *q* имат един и същ остатък, което се записва във вида:

$$A \equiv B \bmod q$$

Съображения при избиране стойността на модула

- Стойността на модула q се намира в границите $q = 2, 3,..., q_{MAX}$. Използуването на $q = 1$ няма смисъл, тъй като в този случай за всяко N се получава $r_N = 0$.
- Колкото повече се увеличава стойността на модула q, толкова повече се увеличава стойността на остатъка r_N, а от там и броят k на контролните разряди и сложността на контролната апаратура. До определена оптимална стойност на q надеждността на апаратурата се увеличава, след което започва да спада. При това винаги се спазва условието $k << m$.
- Модул $q = p$, където p-основата на бройната система, не се използва, тъй като ще се контролира само най-младшият разряд на контролираната дума N
- Модул $q = p^t$, където t означава брой младши разряди, ще контролира само тези t разряда.

Определяне на остатък по произволен модул

Един от начините за определяне на остатъка на дадено число по модул q, както беше показано по-горе, е непосредственото деление на числото с модула (следва от определението за остатък по модул).

Друг, по-рационален начин за намиране на остатъка, е чрез намиране на остатъците:

$$\sigma_I = r(p^i) \bmod q$$

на степените на основата на бройната система p. Величината σ_i се нарича тегловна функция на i-тия разряд. Тогава

$$r_A = \sum_{i=0}^{m-1} \sigma_i a_i \bmod q .$$

В таблицата по-долу е показано как се изчисляват тегловните функции σ_i за mod *3, 5* и *7*.

N	2^i	q = 3	q = 5	q = 7
		$\sigma_i \equiv 2^i \bmod 3$	$\sigma_i \equiv 2^i \bmod 5$	$\sigma_i \equiv 2^i \bmod 7$
0	$2^0 = 1$	$1 \equiv \mathbf{1} \bmod 3$	$1 \equiv \mathbf{1} \bmod 5$	$1 \equiv \mathbf{1} \bmod 7$
1	$2^1 = 2$	$2 \equiv \mathbf{2} \bmod 3$	$2 \equiv \mathbf{2} \bmod 5$	$2 \equiv \mathbf{2} \bmod 7$
2	$2^2 = 4$	$4 \equiv \mathbf{1} \bmod 3$	$4 \equiv \mathbf{4} \bmod 5$	$4 \equiv \mathbf{4} \bmod 7$
3	$2^3 = 8$	$8 \equiv \mathbf{2} \bmod 3$	$8 \equiv \mathbf{3} \bmod 5$	$8 \equiv \mathbf{1} \bmod 7$
4	$2^4 = 16$	$16 \equiv \mathbf{1} \bmod 3$	$16 \equiv \mathbf{1} \bmod 5$	$16 \equiv \mathbf{2} \bmod 7$
5	$2^5 = 32$	$32 \equiv \mathbf{2} \bmod 3$	$32 \equiv \mathbf{2} \bmod 5$	$32 \equiv \mathbf{4} \bmod 7$
6	$2^6 = 64$	$64 \equiv \mathbf{1} \bmod 3$	$64 \equiv \mathbf{4} \bmod 5$	$64 \equiv \mathbf{1} \bmod 7$
7	$2^7 = 128$	$128 \equiv \mathbf{2} \bmod 3$	$128 \equiv \mathbf{3} \bmod 5$	$128 \equiv \mathbf{2} \bmod 7$
8	$2^7 = 256$	$256 \equiv \mathbf{1} \bmod 3$	$256 \equiv \mathbf{1} \bmod 5$	$256 \equiv \mathbf{4} \bmod 7$

Пример 1. Нека N = 101110 и q = 3. Да се определи остатъкът $r_3(N)$
Съгласно (2.7) $r_N = \sigma_6.1 + \sigma_5.0 + \sigma_4.1 + \sigma_3.1 + \sigma_2.1 + \sigma_1.0 =$

$$= 2.1 + 1.0 + 2.1 + 1.1 + 1.1 + 1.0 = 7 \equiv 1 \bmod 3$$

Пример 2. Нека N = 101110 и q = 5. Да се определи остатъкът $r_5(N)$
Съгласно(2.7) r_N=2.1+1.0+3.1+4.1+2.1+1.0 = 11 ≡ 1 mod 5

Цифров контрол по модул

Съдържанието на информационната част е самото число *N,* а съдържанието на контролната част е остатъкът r_N от делението на *сумата на цифрите* на числото на модула *q,* т.е.

$$r_N = \sum_{i=1}^{k} n_i \bmod q .$$

При цифровия контрол по модул най-голямо разпространение е получил случаят, при който модулът *q* е равен на основата на бройната система *p* т.е.q = p.

При двоична система очевидно *q=2* и контролният разряд е само един , тъй като остатъкът при деление на *2* може да бъде *0* или *1*. В този случай се получава контрол по четност и контрол по нечетност, поради което *цифровия контрол по модул 2* се нарича *контрол по четност*.

Частни случаи на контрола по модул

Тук влизат две групи кодове, получили най-голямо разпространение, съответно с модули $q = p^t - 1$ и $q = p^t + 1$

При контрол по модул $q = p^t - 1$, за най-често използваната - двоичната бройна система, т.е. $p = 2$ модулът добива вида $q = 2^t - 1$. Тогава

q = 3, при t = 2,

q = 7, при t = 3,

q = 15, при t = 4.

При контрол по модул $q = p^t + 1$ за $p = 2$ модулът добива вида $q = 2^t + 1$. Тогава

q = 3, при t = 1

q = 5, при t = 2

q = 9, при t = 3.

При тези модули се получават допълнителни съотношения за намиране на остатъка r_N, които водят до по-прости схеми за неговото получаване. По тази причина те са намерили най-голямо приложение.

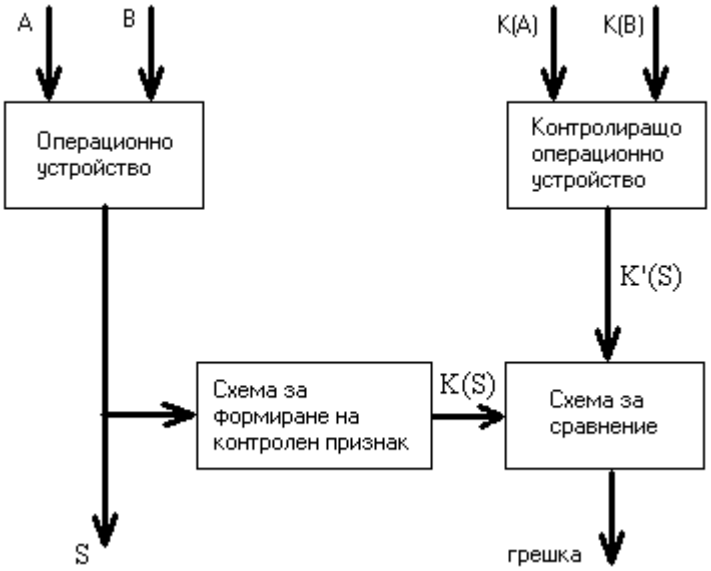
Контрол по модул на аритметични операции

Основна операция в цифровите машини е операцията *двоично събиране.* Известно е, че:

- операцията *изваждане* се привежда към операция събиране, чрез използването на обратен и допълнителен код за представяне на отрицателните числа;
- операцията *умножение* се свежда до събиране и последователни измествания в суматора, регистъра на множителя и регистъра на множимото, съобразно прилагания алгоритъм;
- операцията *деление* се свежда до изваждане (събиране в обратен или допълнителен код) и изместване на делителя и частното.

От това следва, че основно внимание трябва да бъде отделено на операцията събиране.

Контролирането на операцията *събиране* се извършва според основния принцип на схемния контрол :



Между операндите *A* и *B* се изпълнява операция α, при което се получава резултат

$$S = A\alpha B.$$

Операндите се съпровождат от контролни признаци *k(A)* и *k(B)*. Контролиращото устройство изпълнява операция $\beta = f(\alpha)$ върху *k(A*) и *k(B),* при което се получава очакваният контролен признак на резултата

$$k'(S) = k(A)\ \beta\ k(B).$$

Схемата за формиране на контролен признак изработва признак *k(S)* на получения резултат от операцията α над *A* и *B*.

Сигнал за грешка се формира при несъвпадение на двата контролни признака – очаквания и реално получения.

Значенията на контролните признаци се определят в зависимост от избраната стойност на модула *q*, по които се контролира операцията.

Функцията β на контролиращото операционно устройство, на изхода на което се формира очаквания контролен признак за операция двоично събиране $C=A+B$ се описва с израза

$$r_q(A + B) = [r_q(A) + r_q(B)] \bmod q.$$

Вижда се, че за да се контролира по модул q операцията събиране на две числа е необходимо да се извърши сумиране по модул q на техните контролни признаци

Пример .Да се контролира по модул 3 операцията събиране на числата A и B.

A	=	1010111	$r_3(A)$	=	00
B	=	10100	$r_3(B)$	=	10
A + B	=	1101011	$r'_3(A+B)$	=	$[r_3(A) + r_3(B)] \bmod 3$,
$r_3(A + B)$	=	10	т.е. $r'_3(A+B)$	=	00 +10 = 10

Полученият и очакваният контролен признак съвпадат, следователно операцията е изпълнена правилно.

Контрол по модул на логически операции

Контролирането на логическите операции, аналогично на аритметичните, се извършва съгласно основния принцип на схемния контрол Отново се поставя въпросът за определяне на функцията β, която се реализира в контролиращото операционно устройство за формиране на очаквания контролен признак.

Контрол на операцията логическо събиране

$$r_V \equiv (r_A + r_B - r_\wedge) \bmod q .$$

Контролирането на операцията логическо събиране трябва да се изпълни в следния ред:

1. Изпълнява се операцията логическо умножение на числата $A.B$

2. Определя се контролния признак на логическото произведение $r_\wedge \bmod q$.

3. В контролиращото устройство по формула (2.11) се изчислява очакваният контролен признак $r'_V \bmod q$.

4. Изпълнява се операцията логическо събиране $A \vee B$.

5. Определя се полученият контролен признак на резултата от операцията логическо събиране $r_V \bmod q$.

6. Извършва се сравнение на очаквания и получения контролен признак $r'_V = r_V$

Пример. Да се контролира по модул 3 операцията логическо събиране A∨B.

A = 11000011 $r_A = 00 \bmod 3$

B = 11111110 $r_B = 10 \bmod 3$

1) AÙB = 11000010 2) $r_\wedge = 10 \bmod 3$

3) $r'_V \equiv (r_A + r_B - r_\wedge) \bmod q = 00$

4) AvB = 11111111 5) $r_V = 00 \bmod 3$

6) $r'_V \equiv r_V = 00 \bmod 3$, т.е. операцията логическо събиране е изпълнена правилно.

Контрол на операцията логическо умножение

$$r_\Lambda \equiv (r_A + r_B - r_V) \bmod q .$$

Пример. Да се контролира по модул 7 операцията логическо умножение A∧B.

A = 11010101 $r_A = 011$

B = 10001110 $r_B = 010$

1) A∨B = 11011111 2) $r_V = 110$

3) $r'_\wedge = r_A + r_B - r_V = (011 + 010 - 110) \bmod 7 =$

$= -001 = 110 \bmod 7$

4) A∨B = 10000100 5) $r_\wedge = 110 \bmod 7$

6) $r'_\wedge \equiv r_\wedge = 110 \bmod 7$

Тема 47. Тестване и диагностика на компютрите. Тестване на компютрите. Вградено самотестване: POST и BIST. Външно тестване: диагностични пакети

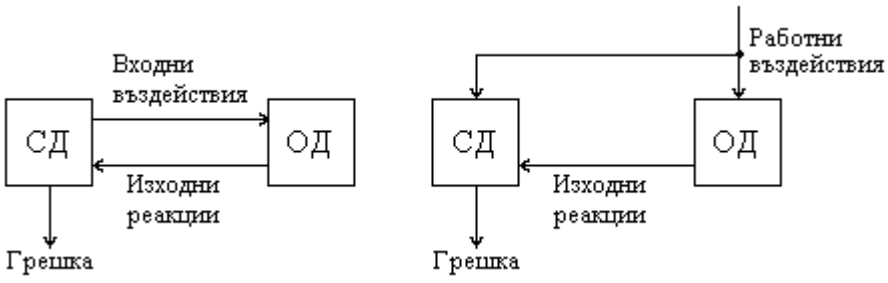
Тестова диагностика

Диагностиката е област от знания, включваща теорията и методите на организация на процеса за определяне състоянието на обектите, както и принципите за построяване на *средствата за диагностика* (СД). Когато изследваните обекти са технически тя се нарича *техническа диагностика*.

Една от важните задачи на техническата диагностика е определяне мястото (*локализиране*) на възникналите неизправности. Този процес се нарича диагностика на неизправностите

Средствата и обектът на диагностика образуват *система за диагностика*. В общия случай, протичащият в системата за диагностика процес, представлява многократно подаване към обекта на определени въздействия (входни набори) и многократно измерване и анализ на реакцията (изходните сигнали) на схемите на тези въздействия. Този процес се нарича *тестване* на обекта.

Съществуват два основни вида системи за диагностика: тестова (фиг. 1,а) и функционална (фиг. 1,б).



а) *б*)

фиг.1 Тестова(а) и функционална диагностика(б)

По-голямо приложение в практиката е намерила системата за тестова диагностика.

Нека обект на диагностика да бъдат *компютърните системи*. Тогава съдържанието на основните задачи на техническата диагностика могат да се дефинират така.

Първата задача е свързана с изучаване функционирането на логическите схеми. Компютърните системи са изградени на базата на двата основни типа схеми:

- комбинационни схеми (КС)
- последователностни схеми (ПС)

Функционирането им се подчинява на правилата на Булевата алгебра и променливите величини са двоични. Това определя основният тип неизправности, т.нар. *неизправности от логически тип*.

Втората задача е свързана с разработването на методи за построяване на тестове за откриване и локализиране на неизправностите.

Тест $T_к$ се нарича последователност от входни вектори $X_{k1}, X_{k2}, ..., X_{kl}$ и съответните им изходни вектори $Z_{k1}, Z_{k2}, ..., Z_{kl}$. Цялото число l се нарича дължина на теста. Тестовете биват два вида:

- контролни (откриващи неизправност);
- диагностични (локализиращи неизправност).

Контролният тест дава еднозначен отговор на въпроса дали е изправна схемата.

Диагностичният тест открива мястото на неизправността в схемата, т.е локализира я.

Третата задача е свързана със създаването на диагностични процедури, които определят начина за подаването на тестовете към логическата схема за извършване на процеса по откриване и локализиране на неизправностите.

Вградено самотестване

POST процедура

За компютрите от тип персонални компютри (PC), при всяко включване се стартира диагностична програма, наречена *самотест при включване (POST, Power On Self Test).*

POST е програма на BIOS, която автоматично получава управлението при включване на захранването или рестарт.

При всички компютри в тази програма се включват:

- тест на ОП;
- намиране на контролната сума на ПП;
- проверка на таймера;
- проверка на DMA;
- проверка на видеоконтролера;
- проверка на контролерите за прекъсвания;

- проверка на клавиатурата.

Процесът на диагностика продължава от няколко секунди до няколко минути и това време е толкова по - голямо, колкото повече е ОП на компютъра. Ако по време на проверката се открие някаква неизправност, на екрана се извежда съобщение, придружено със съответния звуков сигнал. При рестартиране на вече включен компютър се извършват всички проверки без теста на ОП.

POST-програмата цели да извърши бърза проверка на най - важните части на компютъра. Проверките са от типа *работи/не работи*, поради което локализацията на неизправността има относителен характер. За по-пълно тестване на компютъра, адаптерите към него и включените ПУ съществуват диагностични програми, записани на дискета, наречени *диагностични пакети.*

Програмата POST може да се раздели на три части в зависимост от това, как се сигнализира откритата неизправност:

- тестове, непредизвикващи съобщение за грешка;
- тестове, предизвикващи звуков сигнал при откриване на грешка;
- тестове, извеждащи съобщение за грешка на екрана.

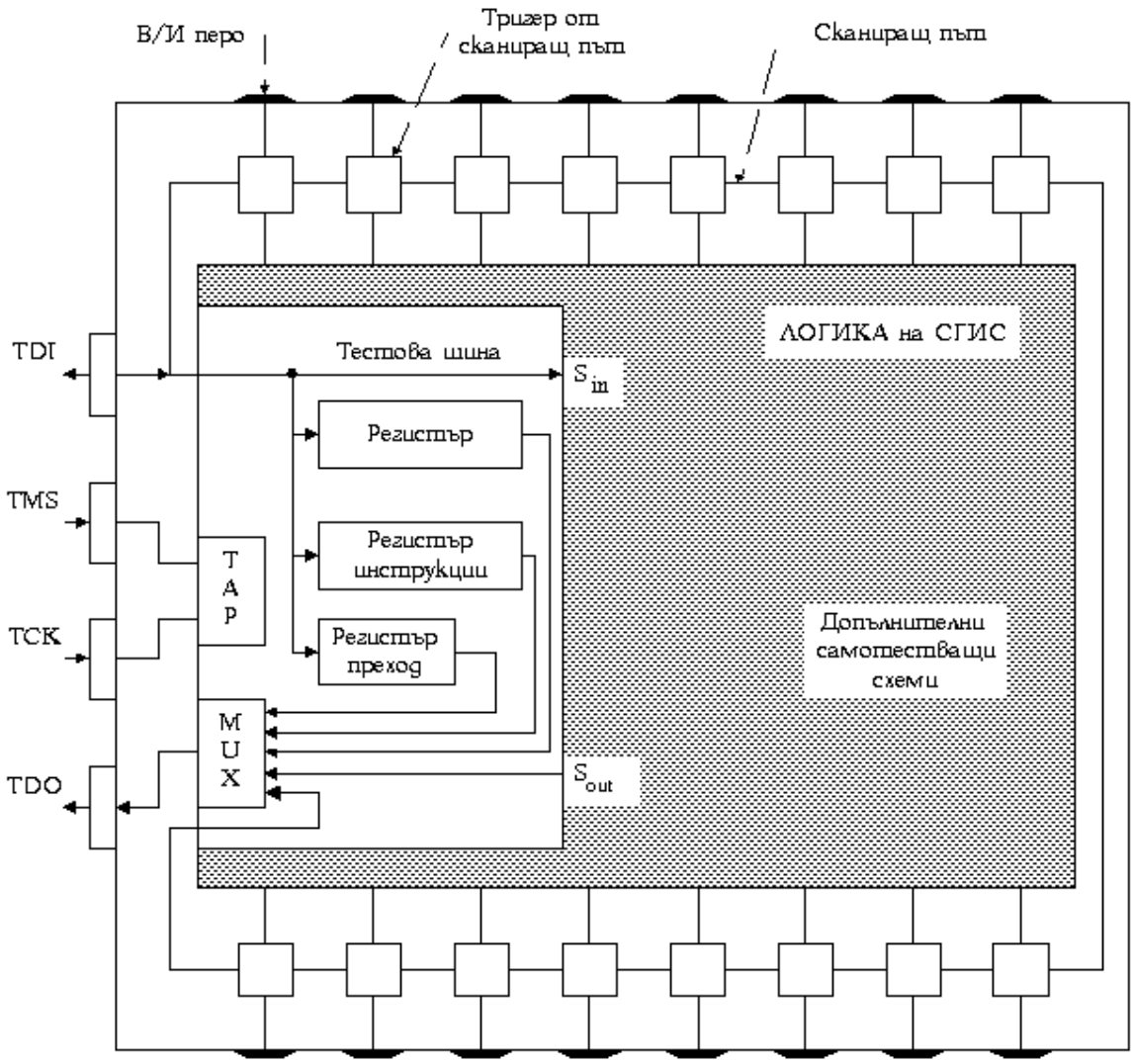
BIST технология

Методът на сканирането има редица предимства , но и някои съществени недостатъци недостатъци:

- преструктуриране на схемата при въвеждането на допълнителна апаратура и от там намаляване на бързодействието;
- невъзможност за провеждане на динамично тестване на пределни работни честоти;
- конструктивно-топологични ограничения за постигане на максимална плътност на монтажа.

За решаването на тези проблеми е въведен т.нар. *метод на граничното сканиране (Boundary Scanning).* Този метод е стандартизиран в електронната индустрия, като най-масовият стандарт е IEEE1149. Идеята е да се свързва входно-изходната логика на интегрална схема или печатна платка като преместващ регистър, който я огражда (фиг.2). Според стандарта IEEE 1149.1 в СГИС се вгражда и тестваща логика *BIST (Built-In Self-Test)*, която може да управлява функционалното тестване на цялата ИС

Обединяването на метода на граничното сканиране с BIST е концепция за тестване на съвременните все по-бързи и все по-сложни СГИС.

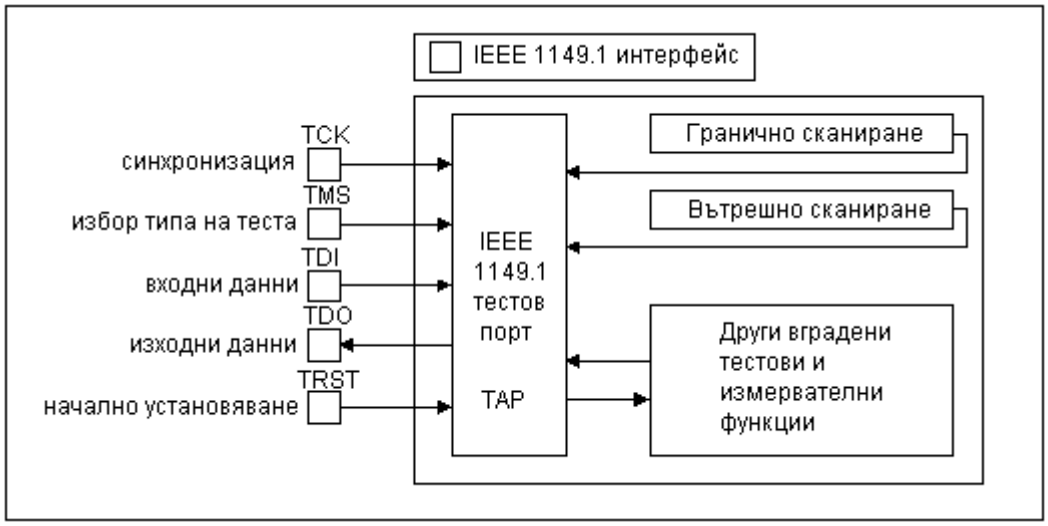


Фиг.2 Гранично сканиране JTAG на интегрална схема

Входно-изходната логика е тригер, който в режим на сканиране е свързан като преместващ регистър със сериен вход TDI (Test Data Input) и сериен изход TDO (Test Data Output). Управляващите сигнали са TMS (Test Mode Signal) и TCK (Test Clock). Тестовите инструкции и тестовите данни се изпращат чрез TDI-входа. Резултатите от тестването се изпращат чрез TDO в сериен вид. Последователността от тестовите операции се задава от тестовата система.

Фиг.3 Тестов порт

Управлението се извършва през TAP (Test Access Port). Типичните за порта четири входни и един изходен пин са показани на фиг. 3.



Гранично сканиране и BIST в процесорите PENTIUM

Pentium може да изпълнява *тест за гранично сканиране JTAG*, съгласно стандарта IEEE1149.1. Той позволява външна проверка на логическите устройства на Pentium и техните вериги дори след затварянето на чипа и неговото вграждане в компютъра. Освен това на процесора може да му бъде дадена команда за изпълнение на *вътрешно самотестване BIST*.

Тестът JTAG се осъществява чрез петте извода :

- TCK (извод T4) - тактова честота;
- TDI (извод T21) - вход за данни на теста;
- TDO (извод S21) - изход за данни от теста;
- TMS (извод P19) - избор на режим на теста;
- TRST (извод S18) - рестартиране на теста.

В чипа Pentium се използват регистър за сканиране на границите, JTAG логика и TAP контролер, до който се получава достъп посредством тестовия порт TAP. Този тест трябва да бъде извършен от външна логика, която използва JTAG тестовата шина с петте си извода. Регистрите за запис не са свързани с входовете/изходите на Pentium, а един с друг като оформят път между TDI и TDO. Данните в еднобитовите регистри за запис се придвижват от TDI към TDO на всеки TCS цикъл без да се променят по време на този процес. Посредством регистрите за запис, намиращи се на “границите”, ”стойностите на границите” се проверяват и се изпращат към TDO. Всички тригери за запис , входно/изходни и контролни обединени заедно образуват т.нар. *сканиращ граничен регистър*. Контролните тригери определят посоката на трансфера за двупосочните изводи (D63-D0) или способността за изключване (ADS) - високо импедансно състояние.

Външно тестване: диагностични пакети

CheckIt

Контролно-диагностичният пакет CheckIt има структура, включваща главен диагностичен файл и вторични диагностични файлове.

Главният диагностичен файл реализира проверката на основните модули на микрокомпютъра и зарежда вторичните файлове.

Вторичните диагностични файлове служат за проверка на допълнително включените към микрокомпютъра адаптери и периферни устройства. Предвидена е възможност за разширяване на контролно-диагностичния пакет чрез включване на нови вторични диагностични файлове.

CheckIt автоматично определя основната конфигурация на микрокомпютъра и тестовете, които са достъпни за потребителя. Определя се :

- свързан ли е компютърът в мрежа;
- съставът на системата;
- производителят на BIOS;
- обемът на основната, разширената и допълнителната RAM памет;
- свързана ли е към компютъра мишка.

След определяне на основната конфигурация се стартира основното меню на главния диагностичен файл. То включва:

- **SySInfo** - съдържа функции за извеждане на пълна информация за конфигурацията, карта на паметта, информация за прекъсванията, драйверните програми на устройствата и таблица на CMOS;
- **Tests** - включва обща проверка, проверка на паметта, ЗУТМД, ЗУГМД, системната платка, последователния и паралелния портове, принтера, видеомонитора и неговия контролер, входните устройства;
- **Benchmarks** - контролни задачи за изследване на производителността на микрокомпютъра;
- **Tools** - включва групата от вторичните файлове и подпрограма за добавяне на нови;
- **Setup** - подпрограма за определяне начина на извеждане на резултатите от диагностичните проверки;
- **Exit** - изход от контролно-диагностичния пакет.

Интерфейсът на програмата е добър, с падащи менюта, прозорци за извеждане на резултатите от проверките и диаграми за тяхното нагледно изобразяване.

Контролно-диагностичният пакет Checkit разполага със средства за оценка на производителността на компютъра. Оценката се извършва на базата на резултатите от изпълнението на контролни задачи, определящи производителността на процесора, видеомонитора и HDD устройства..

Производителността на централния процесор се определя на базата на контролната задача Dhrystone. Dhrystone е една от най-добрите контролни задачи с общо предназначение. Тя се използва предимно за оценка на интерфейса между архитектурата на компютъра и езиците от високо ниво. Задачата е изградена на основата на събрани статистически данни в областта на системното програмиране, като е използвано разпределението на различните типове оператори. Контролната задача съдържа 100 оператора на език от високо ниво (ADA, C) със следното

разпределение: присвоявания (53 %), управляващи оператори (32 %), обръщания към подпрограми и функции (15 %).

Производителността на процесора при изпълнение на математически операции се тества чрез контролна задача Whetstone. Whetstone е една от най-старите контролни задачи. Тя се основава на анализа на голям брой програми, написани на езика ALGOL60. Задачата включва изчисления с плаваща запетая, стандартни математически функции и функции от линейната алгебра. Тя съдържа няколко модула, използващи оператори от различен тип, които се изпълняват многократно. Структурата на модулите и броят на изпълнението им са избрани така, че честотата на използване на всяка инструкция (наречена инструкция на Whetstone), генерирана от програмата, да съвпада с честотата, получена чрез статистическа обработка от 949 потребителски програми в Оксфордския университет.

Оценката на производителността посредством измерване на скоростта за извеждане на екрана на видеомонитора се извършва по два показателя:

- определяне на скоростта за извеждане на информация върху екрана посредством функциите на BIOS;
- определяне на скоростта за извеждане на информация върху екрана посредством директно писане във видеопаметта.

Освен тези контролни задачи се изпълнява и контролна задача за определяне на скоростта на обмен на информация между ЗУТМД и микрокомпютъра, средното време за позициониране и средното време за позициониране на две съседни пътечки.

Контролно-диагностичният пакет CheckIt не дава информация за резидентните програми, а както и за прекъсванията (програмни и хардуерни).

QAPlus

По своя интерфейс с потребителя *QAPlus* е идентичен с контролно - диагностичния пакет CheckIt, но в структурата му присъства един съществен недостатък - не е възможно да се включат повече от два допълнителни теста към пакета.

Структурата на контролно-диагностичния пакет QAPlus е изградена от управляващата програма и вторични програми. Управляващата програма зарежда в паметта вторичните програми и им предава управлението. Вторичните програми са разделени в няколко групи;

- **Utility** - програми за форматиране на твърдият диск на ЗУТМД и за паркиране на главите за запис/четене;
- **Setup** - програми за установяване на параметрите;
- **Testing** - програми за тестване на модулите на системата;
- **Interact** - програми за проверка на мишката, адаптера за игри, високоговорителя, клавиатурата, за определяне на лошите чипове RAM и за проверка на серийния порт чрез програмирането му;

- **SysInfo** - програми, които дават информация за прекъсванията, конфигурацията на системата, драйверните програми за устройствата, обкръжението, а така също контролни задачи за определяне на производителността на микрокомпютъра;
- **Reports** - програма за формиране на отчет за откритите неизправности;
- **Help** - програма, която дава информация на потребителя как да използва контролно-диагностичния пакет.

Оценяването на производителността на микрокомпютъра става, като се изпълняват контролните задачи Dhrystone и Whetstone, а така също и контролна задача за определяне на скоростта на обмен със ЗУТМД, средното време за позициониране между съседни пътечки и средното време за позициониране на случайна пътечка.

Тема 48. Крайни автомати.

Принцип на работа на крайните автомати. Детерминирани и недетерминирани крайни автомати. Автомат на Мили. Автомат на Мур.

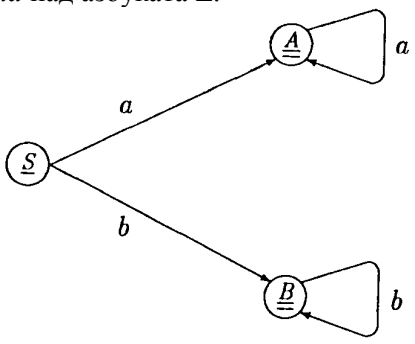
Определение

Краен ориентиран граф с дефинирано начало и крайни върхове представлява *краен автомат без памет* A над азбуката $\Sigma = \{a_1, a_2, ...a_n,\}$.

Ако $G = \langle \Sigma, N, S, P \rangle$ е автоматна граматика, то тя се представя чрез ориентиран граф с върхове – нетерминалните символи от N и дъги – терминалните елементи на Σ. Например, автоматната граматика

$G = \langle \{a,b\}, \{S, A, B\}, S, P \rangle$ с продукции $P = \{S \to aA|bB, A \to aA|a, B \to bB|b\}$

се представя с ориентирания граф на фиг. 1, където върхът $\underline{S}$ е начало на всеки път по графа. Всеки един от върхове $\underline{\underline{A}}$ и $\underline{\underline{B}}$ може да е краен за пътищата в графа. Пътят $S \to A \to A \to A$ еднозначно дефинира думата *ааа* над азбуката Σ.



Фиг. 1. Ориентиран граф

Детерминиран краен автомат (ДКА) е всяка наредена петорка $A = \langle N, \Sigma, \Delta, S, F \rangle$, където

- N е крайното множество от състояния на автомата (азбука на състоянията);
- Σ - крайното множество от входни букви (входна азбука);

- Δ: $N \times \Sigma \to N$ – функцията на преходите, която по текущото състояние и входната буква привежда автомата в ново състояние;
- S – началното състояние на автомата;
- $F \subset N$ - множеството от крайни състояния на автомата.

2. Принцип на работа на крайния автомат

Допуска се, че $w = a_1 a_2 ... a_I \in \Sigma^*$ е произволна дума над Σ. По начално състояние S и първия символ a_1 на w, чрез функцията на преходите Δ се определя следващото състояние на автомата n_1 с равенството

$n_1 = \Delta(S, a_1)$.

На $i+1$ - та стъпка, по текущото състояние n_i и входен символ a_{i+1} се определя следващото състояние:

$n_{i+1} = \Delta(n_i, a_{i+1}) \ \forall i$ в интервала $1 \le i \le I - 1$.

След изчерпване на всички входни символи a_i, $i = \overline{1-I}$, се стига до състояние

$n_I = \Delta(n_{I-1}, a_I)$.

Ако състояние $n_I \in F$ автоматът A разпознава думата w. При $n_I \notin F$ автоматът A не разпознава думата w.

Ако A е детерминиран краен автомат множеството

$$L(A) = \{w \in \Sigma^* | \Delta(S, w) \in F\}$$

е език, разпознаван от автомата A. $L(A)$ се състои от всички входни думи $w \in \Sigma^*$, с които A достига състояние F от начално състояние S.

Недетерминиран краен автомат е наредена петорка $A = \langle N, \Sigma, \Delta, S, F \rangle$, където функцията на преходите $\Delta : N \times \Sigma \to P(N)$ е изображение на $N \times \Sigma$ в множеството $P(N)$ от всички подмножества на N. Множествата от азбуки N, Σ, множеството от крайни състояния F и началният символ S са дефинирани, както при детерминиран краен автомат.

Разширява се дефиниционното множество на функцията на преход Δ върху множеството $P(N) \times \Sigma^*$. Дефинира се функцията Δ' чрез равенството

$$\Delta'(N', a) = \bigcup_{n \in N'} \Delta(n, a) \mid \forall N' \in P(N) \text{ и } a \in \Sigma.$$

При $|N'| = 1$, т.е. върху множеството $N \times \Sigma$ следва $\Delta' \equiv \Delta$.

За всяко $N' \in P(N)$ и $w = \beta a \in \Sigma^*$ се дефинира функцията Δ'' върху множеството $P(N) \times \Sigma^*$ чрез равенствата

$\Delta''(N', \varepsilon) = N'$,
$\Delta''(N', w) = \Delta'(\Delta''(N', \beta), a)$.

Върху множеството $P(N) \times \Sigma$ функцията $\Delta'' \equiv \Delta'$, а върху множеството $N \times \Sigma$ - функцията $\Delta'' \equiv \Delta$.

Множеството от думи, което разпознава недетерминиран краен автомат $A = \langle N, \Sigma, \Delta, S, F \rangle$, се дефинира като

$$L(A) = \{ w \in \Sigma^* | \Delta(\{S\}, w) \cap F \neq \emptyset \}$$

и се нарича **език, разпознаван или породен от** A.

Дефиниции:

1. Два крайни автомата A_1 и A_2 са **еквивалентни**, ако $L(A_1) = L(A_2)$.
2. Формалният език L над азбуката Σ се разпознава от детерминиран краен автомат точно тогава, когато може да се разпознае от някой недетерминиран краен автомат.

АВТОМАТИ НА МИЛИ И МУР

1. Определение

Автомати, разпознаващи думите над дадена азбука се наричат идентификатори на езици. Автомати, преобразуващи думите от език L_1 в думи от език L_2 (не обезателно различен от L_1), се наричат преобразуватели на езици. Автоматите-преобразуватели използват две азбуки (входна и изходна). На всяка дума от входната азбука или от определен език над нея съпоставя по една дума над изходната азбука. Автоматите работят последователно на тактове. На всеки такт се прочита по една буква от входната дума. В зависимост от входната буква и предходното състояние, автоматът преминава в ново състояние и извежда една буква от изходната азбука. Автоматът стартира от определено S_0 начално състояние и завършва ако е достигнал състояние S и входна буква a, за които не е дефиниран или са изчерпани всички букви на входната дума. Резултатът е получената изходна дума.

2. Автомат на Мили

Автоматът на Мили е наредена шесторка $M = \langle N, \Sigma, W, \Delta, \Phi, S_0 \rangle$, където N е крайното непразно множество – азбука на състоянията на автомата; Σ - крайната входна азбука; W - крайната изходна азбука; $\Delta : N \times \Sigma \to N$ - функцията на преходите; $\Phi : N \times \Sigma \to W$ - изходната функция; $S_0 \in N$ - началното състояние на автомата.

Функциониране на автомата на Мили:

Входна дума: $\alpha = a_1 a_2 ... a_I \in \Sigma^*$:

- по начално състояние S_0 и първа буква a_1 се определя следващо състояние $S_1 = \Delta(S_0, a_1)$ и изходна буква $b_1 = \Phi(S_0, a_1)$;
- по състояние S_{i-1} и i-та буква на входната дума, на i-тия такт се определя ново състояние $S_i = \Delta(S_{i-1}, a_i)$ и i-та буква на изходната дума $b_i = \Phi(S_{i-1}, a_i)$, където $i = 2,3,...I$.

Думата $\alpha = a_1 a_2 ... a_I \in \Sigma^*$ се преобразува в думата $w = b_1 b_2 ... b_I \in W^*$, т.е. $w = M(\alpha)$.

Автоматът M преобразува формалния език $L_1 \subset \Sigma^*$ в език $L_2 \subset W^*$, т.е.

$$L_2 = \{ w \in W^* | w = M(\alpha), \alpha \in L_1 \}.$$

Пример:

Разглежда се автомат на Мили от вида

$M = \langle \{S_0, S_1, S_2\}, \{0,1\}, \{a,b\}, \Delta, \Phi, S_0 \rangle$,

където $\Delta(S_0,0) = S_1$; $\Delta(S_0,1) = S_0$

$\Delta(S_1,0) = S_2$; $\Delta(S_1,1) = S_1$

$\Delta(S_2,0) = S_0$;

$\Phi(S_0,0) = a$; $\Phi(S_0,1) = a$;

$\Phi(S_1,0) = a$; $\Phi(S_1,1) = b$;

$\Phi(S_2,0) = b$.

При входна дума: 11010 се получава изходна дума *aaaba* при следните стойности на функциите на преходите:

$\Delta(S_0,1) = S_0$; $\Phi(S_0,1) = a$;

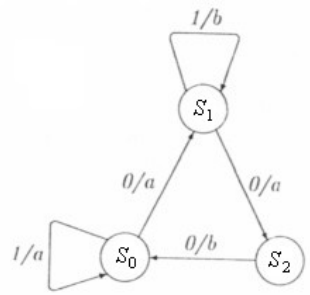
$\Delta(S_0,1) = S_0$; $\Phi(S_0,1) = a$;

$\Delta(S_0,0) = S_1$; $\Phi(S_0,0) = a$;

$\Delta(S_1,1) = S_1$; $\Phi(S_1,1) = b$;

$\Delta(S_1,0) = S_2$; $\Phi(S_1,0) = a$.

Графът на автомата на Мили е показан на фиг. 1.



Фиг. 1.Автомат на Мили

3. Автомат на Мур

Автоматът на Мур е наредена шесторка $N = \langle K, \Sigma, W, \Delta, \Phi, S_0 \rangle$, където K е крайното непразно множество – азбука на състоянията на автомата; Σ - крайната входна азбука; W - крайната изходна азбука; $\Delta : N \times \Sigma \to K$ - функцията на преходите; $\Phi : N \times \Sigma \to W$ - изходната функция на автомата на Мур - N; $S_0 \in K$ - началното състояние на автомата.

3.1. Функциониране на автомата на Мур

Входна дума: $w = a_1 a_2 ... a_I \in V^*$:

- от начално състояние S_0 се получава първата буква $b_1 = \Delta(S_0)$ на изходната дума; по начално състояние S_0 и първа буква a_1 се определя следващо състояние $S_1 = \Delta(S_0, a_1)$;
- на i-тия такт по състояние S_{i-1} се определя i-та буква на изходната дума $b_i = \Phi(S_{i-1})$ и новото състояние $S_i = \Delta(S_{i-1}, a_i)$ за $i = 2,3,...I$.

Автоматът на Мур завършва работата си в краен брой тактове, т.е. достига в състояние, в което не е дефиниран или буквите на $w = a_1 a_2 ... a_I \in V^*$ са изчерпани. Резултатът от работата на автомата N е изходната дума $w' = b_1 b_2 ... b_{I+1} \in W^*$. Очевидно, ако $w = \varepsilon$, то $w' = \Phi(S_0) = b_1$. Автоматът на Мур *преобразува* входната дума $w \in V^*$ в изходната дума $w' = b_1 b_2 ... b_{I+1} \in W^*$, което се изразява чрез функцията $w' = N(w)$.

Пример:

Разглежда се автомат $N = \langle \{S_0, S_1, S_2\}, \{0,1\}, \{a, b\}, \Delta, \Phi, S_0 \rangle$,

където

$\Delta(S_0,0) = S_1$; $\Delta(S_0,1) = S_2$

$\Delta(S_1,0) = S_2$; $\Delta(S_2,1) = S_0$

$\Phi(S_0) = a$; $\Phi(S_1) = a$;

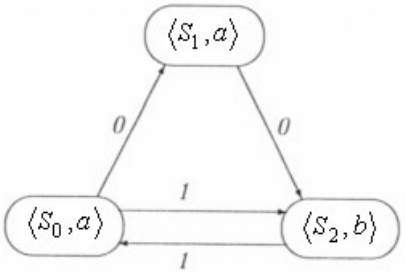
$\Phi(S_2) = b$.

Функциите Δ и Φ могат да се зададат чрез таблици:

Δ	0	1
S_0	S_1	S_2
S_1	S_2	-
S_2	-	S_0

Φ	a	a	b
	S_0	S_1	S_2

Диаграмата на автомата на Мур е представена като краен ориентиран граф на фиг. 2.



Фиг. 2. Автомат на Мур

Теорема за еквивалентност на автоматите на Мур и Мили

Теорема: Нека $N = \langle N, \Sigma, W, \Delta, \Phi, S_0 \rangle$ е автомат на Мур. Тогава съществува автомат на Мили, който е *еквивалентен* на *N*.

Дефинира се следният автомат на Мили:

$\hat{M} = \langle N, \Sigma, W, \Delta, \hat{\Phi}, S_0 \rangle$.

Като се отчете, че *i*-тият изходен символ при автомата на Мили се извежда от предходното състояние на *A* и *i*-тия входен символ, а при автомата на Мур *i*-тият символ се извежда от текущото състояние на автомата на Мур, към който се стига чрез функцията на преходите, дефинирана за предходното състояние и текущия входен символ, се записва следното съотношение

$\hat{\Phi}(S, a) = \Phi(\Delta(S, a))$, за $\forall S \in N$, където S е предходно състояние; a – текущ входен символ.

Нека $w = a_1\ a_2\ a_3 \ldots\ a_k$ е произволна дума от Σ^*. Тогава за думата, разпознавана от автомата на Мур $N(w)$ се записва

$$N(w) = \Phi(S_0)\Phi(S_1)\ldots\Phi(S_k),$$

където

$$S_1 = \Delta(S_0, a_1), S_2 = \Delta(S_2, a_2), \ldots, S_k = \Delta(S_{k-1}, a_k))$$

За думата, разпознавана от автомата на Мили $\hat{M}(w)$, се записва

$$\hat{M}(w) = \hat{\Phi}(S_0, a_1)\hat{\Phi}(S_1, a_2)\ldots\hat{\Phi}(S_{k-1}, a_k) =$$
$$= \Phi(\Delta(S_0, a_1))\Phi(\Delta(S_1, a_2))\ldots\Phi(\Delta(S_{k-1}, a_k)) =$$
$$= \Phi(S_1)\Phi(S_2)\ldots\Phi(S_k)$$

От съпоставянето на двете думи, получени с автоматите на Мур и Мили, се получава условието за еквивалентност на автоматите.

$$N(w) = \Phi(S_0)\hat{M}(w)$$

49. Машина на Тюринг

Дефиниция. Изчислителна конфигурация. Диаграма на преходите.

1. Обща характеристика

Машината на Тюринг е абстрактен автомат с неограничена външна памет, като всяка компонента на запомнената в нея информация е потенциално достъпна за прочитане и замяна.

Машината на Тюринг функционира като идентификатор и преобразувател на езици. Структурата на идентификатора на Тюринг съдържа *управляващо устройство с крайна памет*, което в дискретен момент се намира в едно от краен брой *вътрешни състояния*, и лента, която е крайна или безкрайна. Лентата е разделена на клетки. Във всяка клетка е записана точно един символ от азбука, наречена *лентова азбука*. Част от лентовите символи, образуват входна азбука, чрез която на лентата се записват входните думи. Един лентов символ се използва за означаване на *празна клетка*.

Управляващото устройство чрез четящ и пишещ елемент в дискретен момент чете символ, записан в клетка на лентата, записва на негово място нов лентов символ и се предвижва с една клетка на ляво (left) *–L* или надясно (right) *R*. Едно вътрешно състояние на управляващото устройство е *начално*, а част от вътрешните състояния са *заключителни*.

Машината работи на тактове. На лентата е записана дума от символи на входната азбука. Четящо-записващото устройство се намира на определено място, например върху

крайния ляв символ на думата. Машината започва да работи от началното вътрешно състояние. При всеки следващ такт в зависимост от вътрешното състояние прочетения лентов символ управляващото устройство определя в какво ново вътрешно състояние да премине, кой символ да запише в клетката на прочетения символ и коя нова клетка да прочете – тази отдясно или тази отляво на прочетената клетка. Машината на Тюринг спира работа, ако в управляващото устройство няма инструкция как да продължи.

Дефиниция: Детерминирана машина на Тюринг е наредена седморка $T = \langle N, \Sigma, \Delta, V, S_0, \varepsilon, F\rangle$, където N е крайно множество от вътрешни състояния на машината на Тюринг, наречено азбука на вътрешните състояния;

Σ - крайно множество от входни символи, наречено азбука ма входните символи (входна азбука);

Δ - функция на преходите в дефиниционната област $D(\Delta)$, за която $D(\Delta) \subseteq N \times V$ и област на стойностите $R(\Delta)$, за която $R(\Delta) \subseteq N \times V \times \{R, L\}$, $R, L \notin N \times V$;

V – крайно непразно множество от лентови символи, наречено лентова азбука, такава, че $\Sigma \supset V$;

S_0 – начално вътрешно състояние на машината на Тюринг;

$\Lambda \in V$ - Σ - лентов символ за празната клетка;

$F \subseteq N$ – множество на заключителните вътрешни състояния.

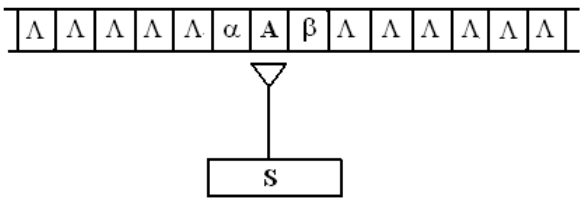
В случай, че областта от стойностите $R(\Delta)$ на функцията на преходите Δ се състои от подмножества на $N \times V \times \{R, L\}$, машината на Тюринг е недетерминитрана. Детерминираните и недетермонирани машинаи на Тюринг са еквивалентни относно разпознаваните от тях езици.

2. Изчислителни конфигурация и процеси в машината на Тюринг

Дефиниция:

Конфигурация на дадена машина на Тюринг $T = \langle N, \Sigma, \Delta, V, S_0, \Lambda, F\rangle$, е наредената тройка $\langle \alpha, S, A\beta\rangle$, където $S \in N$; $A \in V$; $\alpha, \beta \in V^*$.

Конфигурацията $\langle \alpha, S, A\beta\rangle$ показва в даден момент записа на лентата α, наляво от символа A, където е позиционирано четящото устройство, състоянието S на управляващото устройство, символът A, който прочита чрез четящото устройство и записа на лентата β, надясно от символа A, където е позиционирано четящото устройство. Думата $\alpha A \beta$ се нарича лентова дума на конфигурацията. Конфигурацията $\langle \varepsilon, S_0, A\beta\rangle$, където $A\beta \in V^+$ е начална за машината на Тюринг.



Фиг. 1. Машина на Тюринг

В съответствие с функционалната схема на Фиг. 1 управляващото устройство на T се намира в състояние S и прочита клетката със записан върху нея лентов символ A. Думата α се състои от символите, записани върху лентата от най-лявата непразна клетка до прочитаната в момента клетка. Думата β се състои от символите, записани върху лентата от най-дясната непразна клетка до прочитаната в момента клетка. Наляво от α и надясно от β се намират само празни клетки, означени с Λ.

При дадена конфигурация $\langle \alpha, S, A\beta\rangle$ на $T = \langle N, \Sigma, \Delta, V, S_0, \Lambda, F\rangle$, функцията на преходите Δ определя един от следните преходи (тактове на T) към нова конфигурация:

1. Допуска се, че $\Delta(S, A) = \langle Q, B, R\rangle$.

Тогава при $\beta = C\,\beta_1 \neq \varepsilon$

$\langle \alpha, S, A\beta \rangle \rightarrow \langle \alpha B, Q, C\beta_1 \rangle$, ако $\alpha B \notin \{\Lambda\}^*$;

$\langle \alpha, S, A\beta \rangle \rightarrow \langle \varepsilon, Q, C\beta_1 \rangle$, ако $\alpha B \in \{\Lambda\}^*$

При $\beta = \varepsilon$

$\langle \alpha, S, A \rangle \rightarrow \langle \alpha B, Q, \Lambda \rangle$, ако $\alpha B \notin \{\Lambda\}^*$;

$\langle \alpha, S, A \rangle \rightarrow \langle \varepsilon, Q, \Lambda \rangle$, ако $\alpha B \in \{\Lambda\}^*$.

2. Допуска се, че $\Delta(S, A) = \langle Q, B, L \rangle$.

Тогава при $\alpha = \alpha_1 D \neq \varepsilon$

$\langle \alpha, S, A\beta \rangle \rightarrow \langle \alpha_1, Q, DB\beta \rangle$, ако $B\beta \notin \{\Lambda\}^*$;

$\langle \alpha, S, A\beta \rangle \rightarrow \langle \alpha_1, Q, D \rangle$, ако $B\beta \in \{\Lambda\}^*$.

При $\alpha = \varepsilon$

$\langle \varepsilon, S, A\beta \rangle \rightarrow \langle \varepsilon, Q, \Lambda B\beta \rangle$, ако $\Lambda B\beta \notin \{\Lambda\}^*$;

$\langle \varepsilon, S, A\beta \rangle \rightarrow \langle \varepsilon, Q, \Lambda \rangle$, ако $\Lambda B\beta \in \{\Lambda\}^*$.

Последователността от краен брой преходи на една конфигурация на T в друга конфигурация се нарича *изчислителен процес на T*. Конфигурацията, за която функцията на преходите не определя никакъв преход към друга конфигурация, се нарича *крайна*.

Изчислителен процес на T, започващ с начална конфигурация и завършващ с крайна конфигурация, се нарича завършващ *изчислителен процес на T*, т.е. машината на Тюринг е приложима към лентовата дума на началната конфигурация.

Дефиниция: Машината на Тюринг $T = \langle N, \Sigma, \Delta, V, S_0, \Lambda, F \rangle$ разпознава входната дума $w \in \Sigma^*$, тогава и само тогава, когато съществуващ завършващ изчислителен процес на T, т.е.

$\langle \varepsilon, S_0, w \rangle \rightarrow \langle \alpha, Q, A\beta \rangle$,

където $Q \in F$. При $w = \varepsilon$ началната конфигурация е $\langle \varepsilon, S_0, w \rangle$

Множеството от всички ввходни думи, които T разпознава, преставлява *езикът* $L(T)$, *който T разпознава*.

Езиците, разпознавани от T се наричат *рекурсивно номерируеми*. Езиците, разпознавани от T, която за всяка входна дума задава завършващ изчислителен процес, т.е. спира след краен брой тактове, се наричат *рекурсивни езици*.

Функцията на преходите на машината на Тюринг се задава таблично или чрез диаграма на преходите. В *таблицата на преходите* на всяко вътрешно състояние съответства по един ред, на всеки лентов символ – по една колона, а в клетките на пресечните места са дадени стойностите на функцията за съответното вътрешно състояние и съответния лентов символ.

В *диаграмата на преходите на машината на Тюринг* всеки два върха съответстващи на състоянията S и Q се свързват с насочена дъга, ориетирана от S към Q с етикет $A/B, D$ (direction

$D = R$ right, $D = L$ left), като функцията на преходите се записва във вида $\Delta(S, A) = \langle Q, B, D \rangle$, където $A, B \in V$; $S, Q \in N$; $D \in \{R, L\}$.

Пример:

Дадена е машина на Тюринг, дефинирана с наредената седморка

$T = \langle N, \Sigma, \Delta, V, S_0, \Lambda, F \rangle$,

където $N = \{S_0, S_1, S_2, S_3, S_4, S_5, S_6\}$; $\Sigma = \{0, 1, X, \Lambda\}$; $F = \{ S_0\}$, с функция на преходите Δ, дефинирана с уравненията

$\Delta(S_0, 0) = \langle S_1, X, R \rangle$; $\Delta(S_3, 0) = \langle S_5, \Lambda, L \rangle$;

$\Delta(S_0, 1) = \langle S_2, X, R \rangle$; $\Delta(S_4, 1) = \langle S_6, \Lambda, L \rangle$;

$\Delta(S_1, 0) = \langle S_1, 0, R \rangle$; $\Delta(S_5, 0) = \langle S_5, 0, L \rangle$;

$\Delta(S_1, 1) = \langle S_1, 1, R \rangle$; $\Delta(S_5, 1) = \langle S_5, 1, L \rangle$;

$\Delta(S_1, \varepsilon) = \langle S_3, \Lambda, L \rangle$; $\Delta(S_5, X) = \langle S_0, X, R \rangle$;

$\Delta(S_2, 0) = \langle S_2, 0, R \rangle$; $\Delta(S_6, 0) = \langle S_6, 0, L \rangle$;

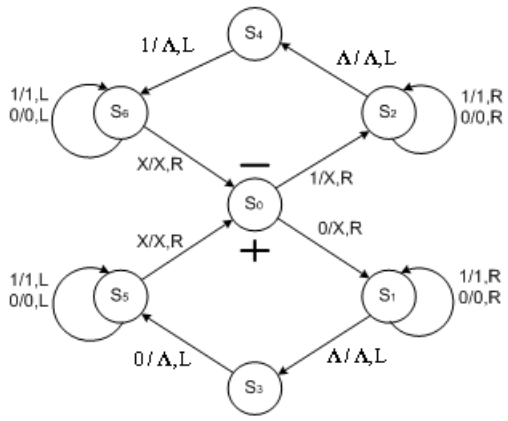
$\Delta(S_2, 1) = \langle S_2, 1, R \rangle$; $\Delta(S_6, 1) = \langle S_6, 1, L \rangle$;

$\Delta(S_2, \varepsilon) = \langle S_4, \Lambda, L \rangle$; $\Delta(S_6, X) = \langle S_0, X, R \rangle$;

Таблица на преходите

	Λ	0	1	X
S_0	-	X, R, S_1	X, R, S_2	-
S_1	Λ, L, S_3	$0, R, S_1$	$1, R, S_1$	-
S_2	Λ, L, S_4	$0, R, S_2$	$1, R, S_2$	-
S_3	-	Λ, L, S_5	-	-
S_4	-	-	Λ, L, S_6	-
S_5	-	$0, L, S_5$	$1, L, S_5$	X, R, S_0
S_6	-	$0, L, S_6$	$1, L, S_6$	X, R, S_0

Диаграмата на преходите е представена на Фиг.2.



Фиг. 2.Диаграма на преходите

Да се анализира работата на T над входните думи 001, 1001 и празната дума ε. Началните конфигурации за думите са както следва

$\langle \varepsilon, S_0, 001\rangle, \langle \varepsilon, S_0, 1001\rangle, \langle \varepsilon, S_0, \Lambda\rangle$

$\langle \varepsilon, S_0, 001\rangle \to \langle X, S_1, 01\rangle \to \langle X\,0, S_1, 1\rangle \to \langle X\,01, S_1, \varepsilon\rangle \to \langle X\,0, S_3, 1\rangle$

$\langle \varepsilon, S_0, 1001\rangle \to \langle X, S_2, 001\rangle \to \langle X0, S_2, 01\rangle \to \langle X00, S_2, 1\rangle \to \langle X001, S_2, \Lambda\rangle \to$
$\langle X00, S_4, 1\rangle \to \langle X0, S_6, 0\rangle \to \langle X, S_6, 00\rangle \to \langle \varepsilon, S_6, X00\rangle \to \langle X, S_0, 00\rangle \to$
$\langle XX, S_1, 0\rangle \to \langle XX0, S_1, \Lambda\rangle \to \langle XX, S_6, 0\rangle \to \langle X, S_5, X\rangle \to \langle XX, S_0, \Lambda\rangle$

Конфигурацията $\langle XX, S_0, \Lambda\rangle$ е крайна, тъй като стойността на функцията $\Delta(S_0, \Lambda)$ не е дефинирана. Машината на Тюринг разпознава думите 1001 и ε, тъй като S_0 е заключително състояние, но не разпознава 001, тъй като S_3 не е заключително състояние.

Машината отбелязва с X първата входната дума и преминава в състояние, което помни тази буква. Преминава през останалите букви на думата без да ги променя, докато стигне до първата празна клетка. След това се връща една клетка назад и сравнява последната буква на входната дума с първата. Ако те не съвпадат, T спира в незаключително състояние (думата не се разпознава). В случай, че съвпадат, изтрива тази буква и движейки се наляво, търси буква X, като не променя буквите в преминаваните клетки. Като открие символа X, машината T се връща с една клетка надясно и преминава в начално състояние. Повтаря се цокълът на сравняване на втората и предпоследна буква на думата. В случай, че няма повече букви за сравнение, T спира в заключително състояние. Следователно, машината T разпознава точно думите с четна дължина, които се четат еднакво отляво надясно и отдясно наляво.

Дефиниция: Ако $T = \langle N, \Sigma, \Delta, V, S_0, \Lambda, F\rangle$ функцията $f_T(w)$ е определима по Тюринг чрез T, ако удовлетворява следните условия

$f_T(w) = \beta$, ако процесът $\langle \varepsilon, S_0, Aw'\rangle \to \langle \beta', Q, B\beta''\rangle$ е завършващ изчислителен процес за $Aw' = w$ и $\beta' B \beta'' = \beta$;

$f_T(w)$ е неопределена в противен случай.

Възможно е изчислителния процес да продължи безкрайно (зацикляне). Това е причина в общия случай определимите по Тюринг функции да са частични. Тези функции позволяват машината на Тюринг да се интерпретира като преобразовател на езици, който, преобразува дадена начална лентова дума в изходна лентова дума.

Построяването на машина на Тюринг за определяне на дадена функция, изисква точен анализ на изчислителния процес, който определя стойностите на функцията и определяне на механизма, чрез който този процес се задава посредством машината на Тюринг. Поради простотата на операциите, които машината на Тюринг извършва и ограничения достъп й до външната памет – лентата, това е трудоемка задача.

За преодоляване на това ограничение е необходимо да се създаде език за програмиране. При това използваните операции ще са свързани само с действието на машината, без да се следи за разпределението на паметта – кога, къде и какво е записано на лентата.

Тема 50: Типове данни в C++. Прости типове, масиви, структури, класове. Управляващи оператори в C++. Оператори за условен преход. Циклични конструкции.

Програмният език C++ допуска използването на различни типове данни, които на най-ниско ниво могат да бъдат разделени в две основни групи: *базови* /вградени/ и *потребителски* /дефинирани от потребителя/. Основните базови типове данни можем да класифицираме по следния начин::

Базови типове данни в C++ с отделената за съхраняване на скаларен тип оперативна памет:

- Скаларни типове
  - Булев */bool/* 1 B
  - Цял */int/* 4 B
  - Реален */double/* 8 B
  - Символен */char/* 1 B
  - Изброен */enum/*
  - Указател
  - Псевдоним
- Съставни типове
  - Масив
  - Вектор
  - Низ

Ето една примерна декларация на променливи от базов тип:

```
void main()
{
int n,m=2;                                  //  Цели
double p=1.5,q;                             //  Реални
bool b1,b2=false;                           // Булеви
char s, st='A';                             // Символни
enum Weekday{Su,Mo,Tu,We,Th,Fr,Sa};         // Изброен тип
int *u=&n;                                  // Указател
char * str;                                 // Указател
int &pse=m;                                 // Псевдоним
}
```

Масивът е съставна статична структура от еднотипни данни от вече деклариран /базов или потребителски/ тип. Масивите се разполагат линейно в оперативната памет независимо дали са едномерни или с повече от една размерност. Името на масива е указател към първия му елемент, т.е. съдържа адреса в ОП, на който е разположен първият елемент на масива, този с индекс 0. Ето примерна декларация на масиви с различна размерност:

```
void main()
{int masiv1[5][10]={{1,2,3,4,5},
                    {6,7},
                    {8,9,10}};
double masiv2[7]={1.5,7,3.4};
char * str, str1[15];
```

}

Достъпът до елемент на масив с индекс *index* се осъществява с преизчисляване на относителния му адрес спрямо елемента с индекс 0 по следната формула:

$$Adr\_new = Adr + index * razmer,$$

където:

- *Adr_new* е адресът в ОП на елемента с индекс *index*
- *Adr* е адресът на първия елемент от масива /имащ индекс 0/, който е записан в клетката, съдържаща името на масива /указателя към елементите му/
- *index* е индексът на търсения елемент
- *razmer* е размерът в байтове на типа на елементите, указан при деклариране на масива...

Структурата и *класът* са съставни статични структури от не еднотипни данни от вече деклариран /базов или потребителски/ тип. Ето примерна декларация на структура:

```
struct record
        {private:
        int n;
        double p,q;

        public:
       double s;
       int function1(int x, double y);
        } ;
```

и клас:

```
class Sample
        {private:
        int data1;
        double data2;

        public:
         Sample (int a, double b);                  // Конструктор
        ~Sample();                                  // Деструктор
         int function(int x, double y);             // Мутатор
        void print() const;                         //Функция за достъп
        } ;
```

Структурата и класът съдържат членове – данни и членове – функции /методи/, достъпът до които се определя с ключовите думи *public, private* и *protected.* Частта, декларирана като *public* се нарича *интерфейс*, а частта *private - капсулирана част.* Ключовата дума *protected* определя правилата за наследяване.

Ако не са указани явно правилата за достъп, членовете на структурата се възприемат като *public*, а на класа – като *private*.

Членовете - функции на клас се разделят на *конструктори, деструктори, мутатори и функции за достъп*. Ако не са явно декларирани конструктори и деструктор, се създават *конструктор и деструктор по подразбиране*. В един клас може да се декларират няколко конструктора, всички с името на класа, различаващи се по броя на параметрите си, генериращи обектите на класа, и един деструктор за разрушаване на обектите. Явното дефиниране на деструктор е задължително при използването на динамичната памет на компютъра.

В класическата компютърна архитектура от фон Нойманов тип предаването на управлението се извършва последователно – на оператора, непосредствено следващ последния изпълнен оператор. Промяната на тази последователност е възможна с операторите за условен преход и цикличните конструкции. Ето пример за прост условен оператор:

```
#include <iostream.h>
void main()
{int n; cin>>n;
if(n<10) cout<<"Цифра"; else cout<<"Число"; }
```

Ако условието е изпълнено се изпълнява операторът /блокът оператори/ следващ *if,* в противен случай - операторът /блокът оператори/ следващ *else.* Ако е изпусната клаузата *else* управлението се предава на следващия след *if* оператор.

C++ позволява използването и на 3 типа *циклични конструкции*. Ето примерното им описание:

- Оператор за цикъл *for:*

```
#include <iostream.h>
void main()
{
for(int i=1;i<10;i++) cout<<i<<endl;
}
```

Схема на работа: Инициализация на управляващата променлива -> Проверка за валидност на стойността -> При истина – изпълнение, иначе - изход -> Актуализация на управляващата -> Проверка.

- Оператор за цикъл с постусловие *do/while:*

```
#include <iostream.h>
int main()
{int n;
do
{cout<<"\nВъведете положително число: "; cin>>n;} while(n<0);
return 0;
}
```

Схема на работа: Изпълнение -> Проверка за валидност на стойността -> При истина – повторно изпълнение, иначе - изход -> Проверка.

- Оператор за цикъл с предусловие *while:*

```
#include <iostream.h>
int main()
{int n=-1;
while(n<0)
{cout<<"\nВъведете положително число: "; cin>>n;}
return 0;
}
```

Схема на работа: Проверка за валидност на стойността -> При истина – изпълнение, иначе - изход -> Проверка.

Тема 51: Процедури и функции в C++. Итерация и рекурсия. Динамични структури в C++. Стек. Опашка. Дърво.

Процедурите и функциите в C++ са основни структурни единици, изграждащи програмите на езика. Точно една от процедурите или функциите има име *main* и с нея стартира изпълнението на програмата. Основните предимства на използването на процедури и функции се състоят в :

- Програмният код става по-ясен за четене и по-лесен за модифициране.
- Избягва се дублирането на едни и същи групи оператори /програмни фрагменти/ на различни места в кода.
- Икономисва се оперативна памет – кодът на процедурите и функциите се компилира еднократно и обръщението към тях се осъществява посредством предаване на управлението.

Синтактически за означаване на процедура се използва ключовата дума *void*, последвана от името на процедурата:

void <име на процедурата> (<формални параметри>) {<тяло на процедурата>}

За разлика от процедурите, при деклариране на функции задължително се указва и типът на функцията:

<тип на функцията > <име на функцията> (<формални параметри>) {<тяло на функцията>}

В този смисъл по начин на приложение процедурите са близки до операторите, а функциите – до променливите. При описанието на процедурите и функциите могат да се използуват един или повече параметри, наречени *формални*. Описанието на типа на формалните параметри е задължително. Замяната на *формалните* с *фактически* параметри става при обръщение към процедурата или функцията в процеса на изпълнение на програмата.

Всяка функция съдържа в тялото си един или повече оператори *return,* като стойността на израза, следващ оператора *return,* се пресмята и се присвоява на името на функцията. При среща на оператор *return* в тялото на функцията изпълнението й се прекратява и управлението се предава на главната програма.

Ето пример за използване на процедура и функция:

```
#include <iostream.h>
void Shapka(char* Ime) // Процедура с един формален параметър
{cout<<"\n Справка за месечното възнаграждение на \n";
cout<<Ime<<endl;}

int Uvelichenie(int Staz)  // Функция с един формален параметър
{Staz++;return Staz;}

int main() //Главна функция
{char * Rabotnik; int St;
cout<<"\nВъведете име: "; cin>>Rabotnik;
cout<<"\n и стаж :"; cin>>St;
Shapka(Rabotnik);
cout<<"\n има нов стаж "<<Uvelichenie(St)<<endl;
return 0;}
```

Итерацията и *рекурсията* са основни способи за реализация на алгоритми, при които стойностите на обектите зависят от управляваща променлива и респективно от вече пресметнати или необходимо да бъдат пресметнати стойности на обектите при други /една или повече/ стойности на управляващата променлива. *Рекурсивни* наричаме функциите, осъществяващи обръщение сами към себе си с вариране на формалните параметри. Рекурсивните функции са ресурсоядни и следва да се използват внимателно. Рекурсията може да се замени с итеративен алгоритъм и/или попълване на таблици с вече пресмятани резултати. Типичен пример в това отношение е рекурсивният алгоритъм за пресмятане на числата на Фибоначи.

Ще илюстрираме с итеративна и рекурсивна реализация на пресмятането на факториел /n!=1*2*3*…*(n-1)*n/.

```
#include <iostream.h>
int Factorial_I(int n)
{int fact=1, k;
for(k=1;k<=n;k++) fact*=k;
return fact;}

int Factorial_R(int n)
{if(n==1) return 1;
else return n*Factorial_R(n-1);}

int main()
{int Chislo;
cout<<"\nВъведете число: ";
cin>>Chislo;

cout<<"\n Факториел итеративно "<<Factorial_I(Chislo)<<endl;
cout<<"\n Факториел рекурсивно "<<Factorial_R(Chislo)<<endl;
return 0;}
```

Формално реализацията на *динамичните структури в C++ /опашки, стекове и дървета/* се осъществява с помощта на масиви. Различията между тях са на алгоритмично – логическо ниво.

Опашката /queue/ е *линейна динамична структура*, при която обслужването на заявките /обработката на елементите на опашката/ става последователно по реда на постъпването им, в съответствие с така наречения *First In – First Out /FIFO/* принцип.. Опашката се характеризира с *начало* и *край.* Всички елементи постъпват в опашката само в края й и я напускат само в началото. Опашката е празна, ако началото и краят й съвпадат. Ето илюстрация на опашка:

начало на опашката

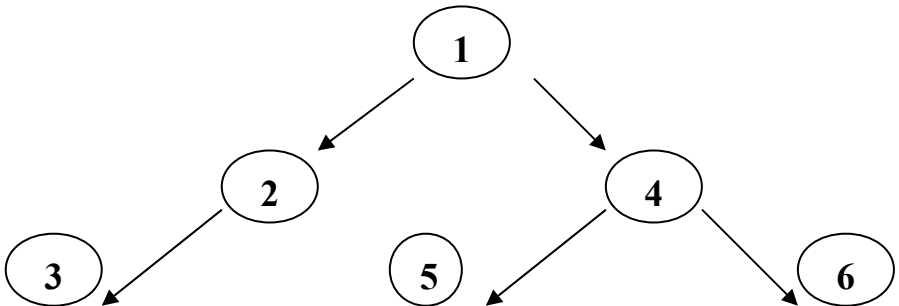
край на опашката

Стекът /stack/ е *линейна динамична структура*, при която обслужването на заявките /обработката на елементите на стека/ става последователно и обратно на реда на постъпването им, в съответствие с така наречения *Last In – First Out /LIFO/* принцип. Елементите постъпват и съответно напускат стека само през *върха на стека*. Обслужва се елементът на върха. Стекът е празен, ако не съдържа елемент на върха. Ето илюстрация на стек:

връх на стека

Дървото /tree/ е *нелинейна динамична структура*, при която обслужването на заявките може да се извърши в съответствие с избрана стратегия на обхождане – *в широчина* или *в дълбочина*. При обхождане на дървото *в широчина* се обслужват заявки в съответствие с нивото на елемента в дървото – първо коренът, после непосредствените му наследници /върхове от първо ниво/, след това всички върхове от второ ниво и т.н. При обхождане *в дълбочина* се обслужват всички леви поддървета до достигане на елемент, не притежаващ ляво поддърво. В този момент се обслужва един наследник от дясното поддърво и се връщаме към стратегията за следване на леви поддървета.

Схема на дърво, обхождано в дълбочина:



Тема 52: Конструиране на класове. Предефиниране на операции в C++. Наследяване на класове в C++.

Конструирането на обекти представители на класа е поверено на специални членове-функции на класа, наречени *конструктори*. *Конструкторът* е член-функция на класа, имаща същото име като класа. Класът може да притежава или да не притежава конструктори. При липса на явно дефиниран конструктор компилаторът създава служебен *конструктор по подразбиране* без формални параметри, чието предназначение е да резервира място в оперативната памет за разполагане на собствените членове данни на обекта.

Всеки клас може да има и повече от един конструктор, различаващи се по броя на формалните параметри. В този случай при конструиране на обект се търси съвпадение първо по брой, а после и по тип на формалните параметри. Съответно някои от параметрите на конструктора могат да бъдат декларирани като *параметри по подразбиране* като се спазва правилото, че след подразбиращ се параметър всички следващи трябва също да са подразбиращи се.

Ето пример за примитивни конструктори:

```
#include <iostream.h>
class Complex
{private:
double Re, Im;
public:
Complex() // Конструктор без параметри
{Re=0; Im=0;}
Complex(double,double); //Конструктор с два параметъра
};
Complex::Complex(double Real, double Imag)
{Re=Real; Im=Imag;}

int main()
{ Complex Q, S=Complex(1,1);
return 0;}
```

Конструкторът без параметри в примера е описан в тялото на класа. Конструкторът с два параметъра е дефиниран извън тялото на класа с използване на оператора за принадлежност "::".

Двата конструктора в примера могат да се обединят в един общ конструктор посредством използване на параметри по подразбиране по следния начин:

```
#include <iostream.h>
class Complex
{private:
double Re, Im;
public:
Complex(double=0,double=0); };
Complex::Complex(double Real, double Imag)
{Re=Real; Im=Imag;}

int main()
```

```
{ Complex Q, S=Complex(1,1);
return 0;}
```

При използване на *функции, имащи за параметър обект на клас, предаван само по стойност*, а не по адрес, се прилагат особени конструктори, наречени *копиращи конструктори. Копиращият конструктор* се използва за присвояване на стойностите на членовете данни между обекта - фактически параметър и клетките, запазени в оперативната памет за съхранение на членовете данни на обекта - формален параметър на функцията. Копиращите конструктори се създават служебно, но е допустимо да се опише и *явен копиращ конструктор*:

```
Complex::Complex(Complex const & C)
{Re=C.Re; Im=C.Im;}
```

Този конструктор се извиква при присвоявания от вида

```
Complex Q = S;
```

и в действителност с помощта на служебния указател *this* се преобразува в

```
Complex::Complex(Complex * this, Complex const & C)
{this -> Re=C.Re; this -> Im=C.Im;}
```

Предефинирането на операции в C++ е възможност на езика, осигуряваща удобства при записване на код и по-добра четимост на вече създадени кодове. Разполагаме с една от следните две възможности:

- Предефиниране посредством приятелска операторна функция:

```
#include <iostream.h>
class Complex
{private:
double Re, Im;
public:
Complex(double=0,double=0);

friend Complex operator + (Complex const & C1, Complex const & C2)
{Complex C(C1.Re + C2.Re, C1.Im + C2.Im);
return C;};
};

Complex::Complex(double Real, double Imag)
{Re=Real; Im=Imag;}

int main()
{ Complex Q, S=Complex(1,1);
Complex T=S+Q;
return 0;
}
```

Изразът *S+Q* в примера се интерпретира от компилатора като извикване на операторната функция **operator** + (S,Q). Лявата асоциативност на оператора се запазва, а в добавка получаваме допълнителни удобства при записване на по-сложни изрази.

- Предефиниране посредством член функция на класа:

```
#include <iostream.h>
class Complex
{private:
double Re, Im;
public:
Complex(double=0,double=0);

Complex operator + (Complex const & C1) const
{Complex C(Re + C1.Re, Im + C1.Im);
return C;};
};

Complex::Complex(double Real, double Imag)
{Re=Real; Im=Imag;}

int main()
{ Complex Q, S=Complex(1,1);
Complex T=S+Q;
return 0;
}
```

Изразът *S+Q* в този пример се интерпретира от компилатора като извикване на операторната функция *S*_**operator** + (Q). Лявата асоциативност на оператора отново се запазва.

Една от основните възможности, осигурявани от програмния език C++ е за *създаването на фамилии от класове*, при което *производният клас* наследява данните и методите на *базовия клас* /при *просто наследяване*/ или на *групата базови класове* /при *множествено наследяване*/. Управлението на *схемите на наследяване* се извършва с помощта на ключовите думи *public, private* и *protected* и е дадено в следната таблица:

Схема на наследяване	Атрибут в базов клас	Наследява се като
public	***private***	***private***
	public	***public***
	protected	***protected***
private	***private***	***private***
	public	***private***
	protected	***private***
protected	***private***	***private***
	public	***protected***
	protected	***protected***

Наследените компоненти се различават от собствените по правата на достъп. Производният клас има пряк достъп до компонентите *public* и *protected* на базовия клас, но няма достъп до обявените като *private.* Ако не е явно указана схемата на наследяване на конкретен базов клас, подразбира се схема на наследяване *private.*

Ето пример за просто наследяване с описание на конструкторите на базов и производен клас:

```
#include <iostream.h>

class base{   // Базов клас
private: int a1;
protected: int a2;
public:
base(int x=0,int y=0)  // Конструктор на базов клас
{a1=x; a2=y;}
};

class der: public base  // Производен клас, схема на наследяване public
{private : int d1;
protected: int d2;
public:
der(int x, int y, int z=0, int t=0): base(x,y)  // Конструктор на производен клас
{d1=z; d2=t;}                                    // с инициализация пред тялото
};

int main()
{ der x(1,2,3,4);
return 0;
}
```

Конструкторът на производния клас използва за създаване на обекти конструктора на базовия клас. Инициализацията е изнесена пред тялото на конструктора на класа наследник.

Тема 53: Шаблони в C++. Шаблони на функции и класове. Предназначение, видове, инициализация, параметри по подразбиране.

Идеята, която е заложена в основата на използването на *шаблони* в C++ е да се позволи описанието на *класове или отделни функции, зависещи от параметър*. Нерядко се налага създаването на програми, реализиращи алгоритми, които не зависят от типа на обработваните данни. В такива случаи е удобно типът на данните да се параметризира при описанието на методите и да се конкретизира при конкретното приложение на метода. Не лош пример е реализацията на стратегията на стека при обслужване на заявки в случаите, когато елементите на стека са например числа или символи.

Използването на шаблони */templates/* позволява *формализация на описанието* с обръщение към неопределени, *хипотетични данни*, които се конкретизират при изпълнението. Подобно обръщение към "обобщена функция" или клас става с параметри от конкретен тип. Компилаторът генерира *шаблонна функция*, заместващи параметрите на шаблона с типовете на подаваните фактически параметри *без да извършва преобразувания на типовете.* Ето пример за *шаблонна функция*:

```
#include <iostream.h>

template <class T>
void read(int n, T * c)   // Шаблонна процедура
{for(int i=0;i<n;i++)
    {cout<<"element["<<i<<"]="; cin>>c[i];}
 }
template <class T>
T minarray (int n, T * a)  //Шаблонна функция
{T min=a[0];
for(int i=1;i<n;i++)
        if(a[i]<min) min=a[i];
return min;
}

int main()
{ cout<<"n = "; int n; cin>>n;
int a[10];
read(n,a); // Обръщение към шаблонна процедура с параметър int
cout<<minarray(n,a)<<endl;  // Обръщение към шаблонна функция. Параметър int
double b[10];
read(n,b); // Обръщение към шаблонна процедура с параметър double
cout<<minarray(n,b)<<endl;//Обръщение към шаблонна функция с double
return 0;
}
```

Описанието на *шаблон на клас* се прави по сходни правила. Шаблонът на клас не е истински клас. Той се използва от компилатора за генериране на конкретните, наричани *шаблонни,* класове. Шаблонните класове се означават също и като *специализации на шаблона*. При описанието на шаблона на клас се използват

- Вградени /описани в декларацията/ и
- Не вградени /описани извън декларацията/

членове функции на шаблона. Всеки шаблон на клас може да съдържа както шаблонни, така и обикновени членове функции. Ето пример за деклариране на *шаблон на клас*:

```
template <class T, class S>
class EXAMPLE
{public:
T func1( T x, S y);
S func2(T x, S y);
private:
T a;

S b;}
```

При изграждане на специализацията се указват желаните типове от шаблона. Примерът по-долу:

```
template <class T, class S= int >
class EXAMPLE
{public:
T func1( T x, S y);
S func2(T x, S y);
private:
T a;
```

S b;}

```
int main()
{ typedef EXAMPLE <int,double> EX1;
EX1 e1;
int n =  e1.func(5,3.14);
return 0;
}
```

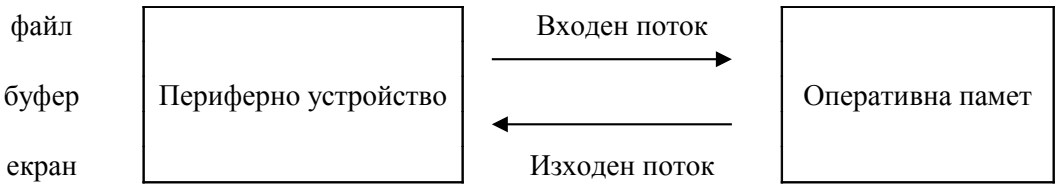
показва конкретна специализация на шаблона и деклариране на:

- Клас EX1 върху шаблона
- Обект e1, представител на класа EX1
- Използване на член функция на класа.

Вторият клас *S* при декларирането на шаблона има стойност по подразбиране *int.* При използване на стойности по подразбиране се спазва общото правило в C++ - след подразбиращ се параметър са допустими само подразбиращи се параметри.

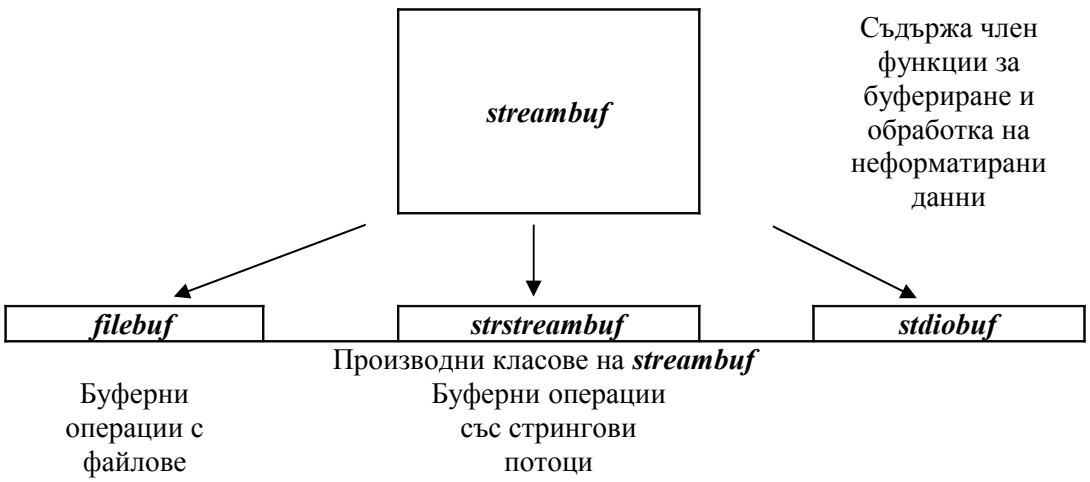
Тема 54: Поточен вход и изход в C++. Управление на потоците, дърво от библиотеки, вход и изход, стандартни потоци, файлови спецификации.

Входно – изходните операции в C++ се реализират с помощта на *потоци*. *Входният* и *изходният поток* може да се разглеждат като последователност от байтове:

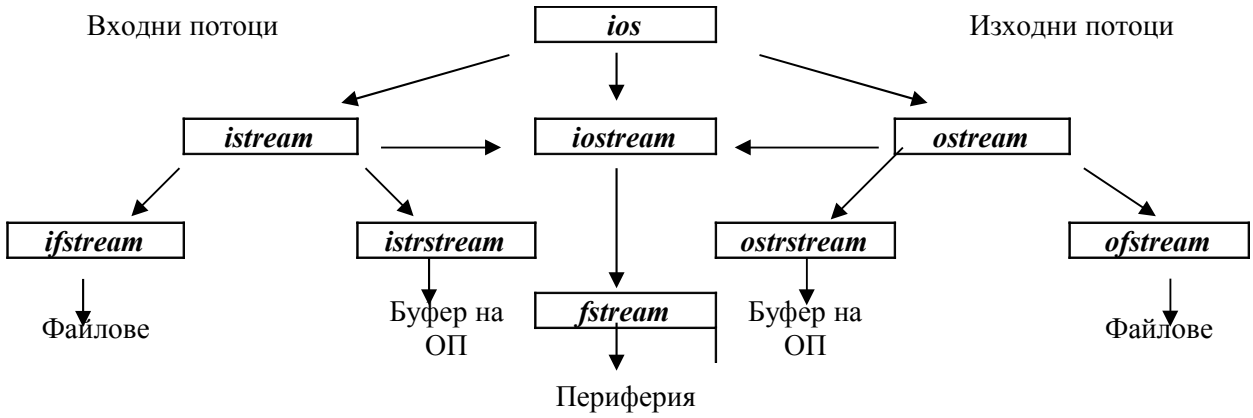


Потоковите операции са реализирани в библиотеката *iostream*, съдържаща над 20 класа, разделени в 2 паралелни йерархични клона:

- Йерархия от класове за управление на буферите



- Йерархия от класове за създаване и управление на потоците



Базовият клас *ios* и всичките му производни класове използуват класа *streambuf* като източник и приемник на данни.

Библиотеката *iostream* е реализирана в 5 *заглавни /header/ файла*:

1. ***iostream.h.*** Използва се при всички входно – изходни операции. Съдържа описание на всички основни класове и стандартните обекти на класове:
   a. ***cin***. Обект на клас *istream.* Стандартно входно устройство /обикновено клавиатура/. Първи аргумент на оператора **>>.**
   b. ***cout.*** Обект на клас *ostream.* Стандартно изходно устройство /обикновено екран/. Първи аргумент на оператора **<<.**
   c. ***cerr.*** Обект на клас *ostream.* Стандартно устройство за грешки /обикновено екран/. Осигурява *небуфериран изход* на съобщенията за грешки, т.е. постъпилите съобщения за грешки се извеждат незабавно.
   d. ***clog.*** Обект на клас *ostream.* Стандартно устройство за грешки /обикновено екран/. Осигурява *буфериран изход* на съобщенията за грешки, т.е. съобщенията се извеждат след напълване на буфера.
2. ***fstream.h*** Използва се за работа с дискови файлове.
3. ***strstream.h*** Използва се за работа със символни низове /стрингове/.
4. ***iomanip.h*** Форматиране за входно – изходния поток с манипулатори
5. ***stdiostream.h*** За съвместимост при смесване на C и C++ код.

Входно – изходните оператори **>>** и **<<** са предварително дефинирани за всички базови типове данни /*int, double, char, char **/ и могат да се използуват директно. Първият операнд на оператора >> /съответно оператора <</ е обект на класа *istream* /съответно на класа *ostream*/, и е свързан с входния /съответно изходния/ поток от данни. Вторият операнд следва да бъде обект или променлива, за която операторът е дефиниран.

Фрагментът

```
int x = 1;
cout<<x;
```

е насочен към стандартния изходен поток, а фрагментът

```
int x = 1;
ofstream fout("file.name",ios::out);
```

fout<<x;

към изходния файлов поток.

И двата оператора са ляво асоциативни и връщат като резултат псевдонима на потока и той може да се използва като първи аргумент на същата операция, т.е. записите

cin >> x >> y >>z;

и

((cin >> x) >> y) >>z;

са еквивалентни.

Като разделители на данните в потока се използват:

- за входния поток – *шпация, табулация, разделител за нов ред*
- за изходния поток *няма разделители*.

Ето и списък на някои по-често използвани член функции за обработка на входния и изходен поток в C++:

- *put().* Клас *ostream.* Каскадна функция.
  - Синтаксис: *ostream & put (char c);*
  - Семантика: Записва символа аргумент в изходния поток чрез който е активирана.
  - Пример:

    fout.put('A');
    cout.put('B');

- *write().* Клас *ostream.* Каскадна функция.
  - Синтаксис: *ostream & write (**const** char*str, **int** size);*
  - Семантика: Записва *size* символа от *str* в изходния поток чрез който е активирана.
  - Пример: /с каскада за *write ()/*

    fout.write("ABCDEF",5).write("GH",1);

    ще запише в изходния поток стринга "ABCDEG".

- *get().* Клас *istream.* Каскадна функция.
  - Синтаксис: *istream & get (char & ch);*
  - Семантика: Извлича от асоциирания входен поток един символ и го свързва с променливата *ch*. Важно е, че се *извличат и разделителите* от входния поток – шпации, табулации и т.н.
  - Пример: /с каскада/

    char C1,C2,C3,C4;
    fin.get(C1).get(C2).get(C3).get(C4);

    Още един пример:

```
char ch;
while(fin.get(ch)) fout<<ch;
while(fin.get(ch)) fout.put(ch);
/*  Горните два цикъла извеждат входния поток в изходния поток едно към едно, докато долният цикъл ще пропусне разделителите */
while(fin >> ch) fout<<ch;
```

- Вариант : *istream & get (char* var_str, int size, char_delim='\n');*

Извлича символи от входния поток и ги свързва с *var_str*. Извличането спира при:

- Достигане на разделителя, указан в третия аргумент /по подразбиране *'\n'* - "нов ред"/
- Край на файла /изчерпване на входния поток/
- Извличане на (*size- 1)* байта.

- *getline()*. Клас *istream*. Каскадна функция.
  - Синтаксис: *istream & getline (char* var_str, int size, char_delim='\n');*
  - Семантика: Действа аналогично на втория вариант на *get*, но пропуска разделителите.

- *read()*. Клас *istream*. Каскадна функция.
  - Синтаксис: *istream & read (char* var_str,* ***int*** *size);*
  - Семантика: Извлича *size* символа от входния поток, чрез който е активирана и ги присвоява на *var_str*.

- *peek()*. Клас *istream*.
  - Синтаксис: *int peek();*
  - Семантика: Връща кода ASCII на символа, който чака на входния поток *без да го извлича*.

- *putback()*. Клас *istream*.
  - Синтаксис: *istream & putback (char ch);*
  - Семантика: Вмъква във входния поток, чрез който е активирана символа, указан с *ch*. Най-често се използва за връщане във входния поток на вече прочетен или пък желан за вод символ.

Програмата е приета на УНС на ЦИТН № 2 от 22.04.2010.

Забележки.

1.Изпитният билет за ДИ съдържа три въпроса от програмата за ДИ и една задача.

2. Кратка разработка на темите от програмата и примерни решени задачи са дадени в учебно помагало за ДИ и са достъпни на сайта за електронни услуги на адрес http://www.bfu.bg/e-services.