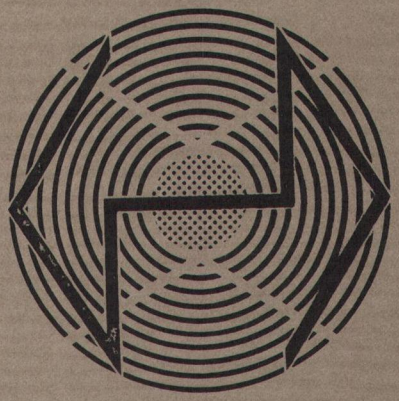


A/S REGNECENTRALEN

SCANDINAVIAN INFORMATION PROCESSING SYSTEMS

RC SYSTEM LIBRARY: FALKONERALLE 1 · DK-2000 COPENHAGEN F

RCSL No: 28-D1

Edition: October 1971

Author: Carsten Koch,
Jørgen Winther

Carl Henrik Dreyer
Okt 73

Title:

Introduction to CF-SYSTEM (Connected Files System)

Keywords: Database Management Information System, System Design, File Organisation, RC 4000, Algol, Fortran, Standard Procedure, Disc, Indexed Sequential File, List File, Chains, Introduction.

Abstract: This introduction describes the CF-SYSTEM as a tool for database design and application. Putting the importance to examples, it should facilitate the understanding of the preliminary manual for the CF-SYSTEM (RCSL No. 53-D9).

FORORD

Når man skal anvende en ny metode i stedet for "den gode gamle", skal der sædvanligvis en vis overtalelse til. Denne overtalelse skal gerne virke motiverende, samtidig med at den giver en orienterende forståelse. Det må desuden med eksempler, der behandler relativt kendte problemer, vises, hvorledes metoden skal anvendes i praksis.

CF-SYSTEM er en række procedurer til filorganisering og filbehandling, og udgør med tilhørende filer et databasesystem. Brugen af CF-SYSTEM er en ny metode, som i en række anvendelser vil være en forbedring af "de gode gamle". Denne beskrivelse er et forsøg på at motivere brugere til at anvende CF-SYSTEM i egnede opgaver. Med hovedvægten lagt på eksempler, skal den kunne lette forståelsen af den foreliggende datamatiske beskrivelse af CF-SYSTEM (RCSL No. 53-D9).

Indholdsfortegnelse.

1. HVEM HENVENDER BESKRIVELSEN SIG TIL?

Formålet med denne brugerorienterede beskrivelse af Regnecentralens Connected Files System (i det følgende kaldt CF-SYSTEM) er at lette vejen for brugere.

Beskrivelsen henvender sig til systemfolk, for at de på en nem og hurtig måde kan se mulighederne i systemet og blive motiverede for at anvende det i egnede opgaver. Beskrivelsen kan også benyttes som en første orientering for programmører, som i forbindelse med en given anvendelse må sætte sig ind i den foreliggende datamatiske beskrivelse af CF-SYSTEM (RCSL No. 5~-D9). Det er håbet, at denne brugerorienterede beskrivelse, hvor hovedvægten er lagt på eksempler, vil kunne motivere og lette forståelsen af den egentlige beskrivelse.

2. BAGGRUNDEN FOR KONSTRUKTION AF CF-SYSTEM

Det har med udviklingen af stadigt mere omfattende administrative edb-systemer baseret på magnetbåndslagre været karakteristisk, at man for at undgå problemer ved behandling af mange filer samtidigt, har søgt at koncentrere data i så få filer som muligt. Dette har medført, at strukturen i de forskellige poster er blevet meget kompleks og sigter på bestemte kendte anvendelser. Tilføjelser og ændringer til sådanne systemer vil ofte være særdeles vanskelige og kostbare, idet de medfører ændringer i grundlaget for de allerede kørende delopgaver. Ofte vil sådanne ændringer derfor blive løst ved "knopskydninger" eller "lapperier", der vil medføre tab af effektivitet, blandt andet fordi der vil forekomme gentagelse af de samme data i flere filer.

Med pladelagres muligheder for ligelig tilgang er det muligt at opbygge basisprogammel, der med en rimelig effektivitet kan administrere data lagret på en måde, der ikke direkte sigter på bestemte anvendelser.

Et database-system som CF-systemet - kan karakteriseres ved at sammenhænge mellem data, der udtrykker at disse data vil blive behandlet sammen i en proces, beskrives eksplicit, dvs. at der etableres filer, hvis poster sammenkæder disse data, og ikke implicit ved posternes struktur, som det er tilfældet ved de traditionelle magnetbåndsløsninger.

Den eksplicite definition af sammenhænge (relationer) har en række brugsmæssige fordele:

Ved udvidelser af systemer, der benytter CF-SYSTEM, er man sikret, at kørende delsystemer ikke påvirkes ved tilføjelse af nye delsystemer, samt at det er simpelt at udvide registre. Delsystemer med beregningsprocesser, der både forudsætter brug af data fra eksisterende filer og data fra nye filer, kan således tilpasses med begrænset arbejdsindsats og således, at der fortsat er en passende balance mellem krav til beregningsmæssig effektivitet og lagerplads (ingen eller ringe gentagelse af data). Da ændringer i eksisterende filer, oprettelse af nye indbyrdes afhængige filer samt etablering af nye sammenhænge kan foretages relativt nemt, er der mulighed for med egnede brugerprogrammer succesivt at opnå en alsidig anvendelse af de lagrede data.

En af forudsætningerne for at kunne kombinere data fra forskellige filer efter ikke forud fastlagte kriterier er, som nævnt, at filerne er lagret på medier med direkte tilgang. Disse lagringsmedier er imidlertid relativt langsomme at arbejde med i forhold til sekventielle medier og ved bevist udnyttelse af de direkte tilgangsmuligheder må man konstatere, at CF-SYSTEM er relativt langsomt.

Overvejelser vedrørende begrænsninger ved en sekventiel løsning kan være afgørende for om man kan have fordel af at benytte CF-SYSTEM. Opgaver, der beskæftiger sig med isolerede funktioner, hvor et betydeligt antal transaktioner med fordel kan behandles ved sekventielt gennemløb af filen, vil næppe kunne drage fordel af CF-SYSTEM og vil lige så godt kunne løses med konventionelle magnetbåndsløsninger. Opgaver, hvor man har behov for at kombinere et stort antal data på mange forskellige måder, er oplagte anvendelser af CF-SYSTEM.

En lang række virksomheder står idag overfor den sidste opgavetype i et forsøg på at opbygge integrerede informationssystemer, hvor der gennem oplysninger fra mange forskellige delområder foretages en koordination af de forskellige funktioner i en virksomhed. Denne koordination skal foretages på en sådan måde, at det er muligt at præsentere beslutningsinformation på tværs af de enkelte organisatoriske delområder, og således at disse delområder kan udnytte fælles information. Dette vil specielt være tilfældet med en række ledelsesmæssige opgaver, hvor oplysninger, der opsamles fra mange forskellige filer, ønskes sammenfattet til et koordineret beslutningsgrundlag.

Grunddata for det integrerede system kan benævnes virksomhedens database, der kan opfattes som en datamæssig model af virksomhedens struktur og aktiviteter. Som karakteristik for en database kan angives:

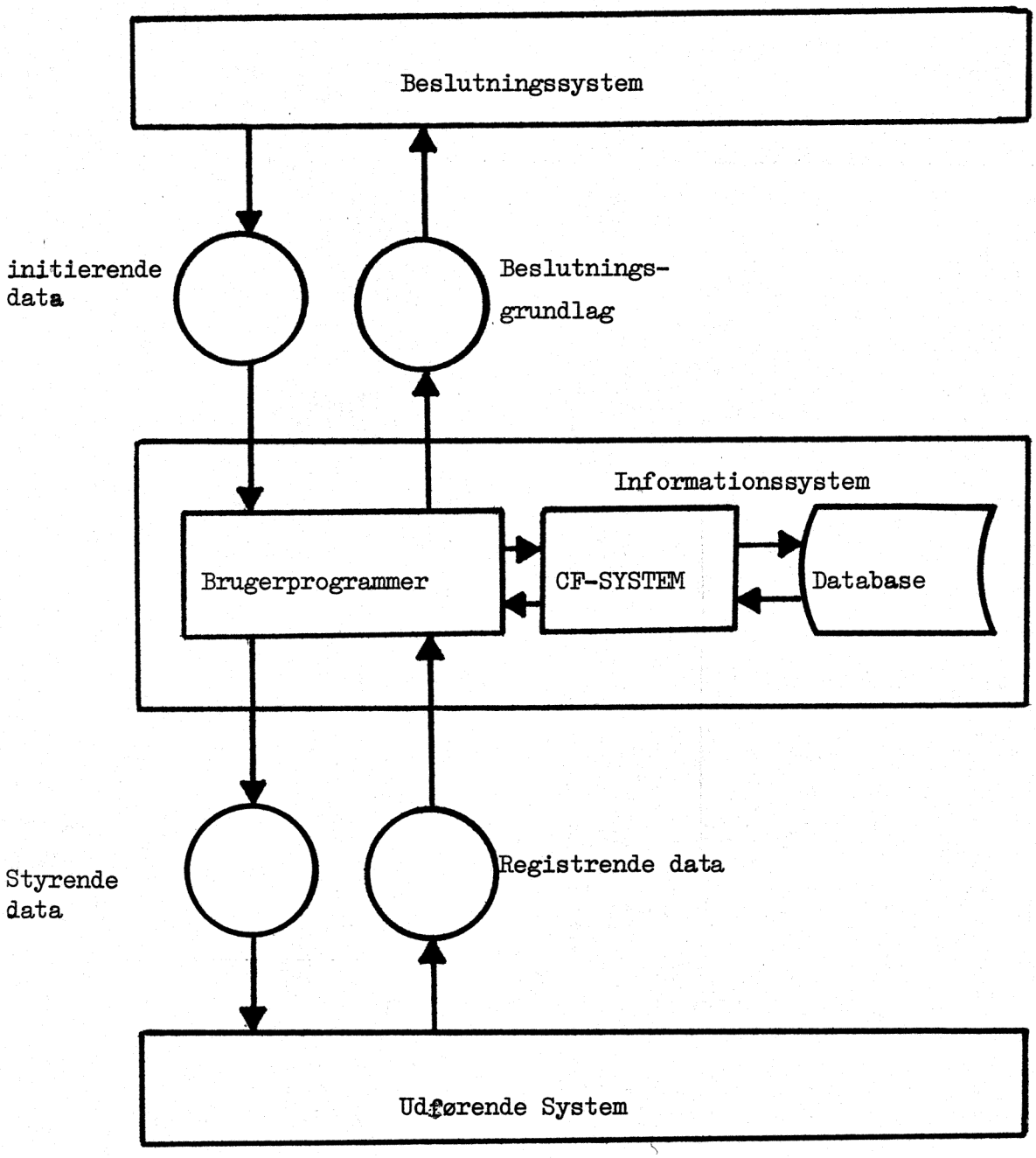
Ved en database forslås mængden af permanente filer i et datamatisk informationssystem, idet det gælder:

- At den metode og det maskinel, der benyttes ved organisering af data, giver en model af objektsystemet (virksomheden), som er hensigtsmæssig for langsigtede anvendelser.

- at etablering af og alle brugerprocessers kommunikation med en således organiseret database sker via et fælles basis-programmel-system.

Første del af definitionen fastslår altså, at udgangspunktet for konstruktion af et integreret system ikke alene skal være de problemer, man ønsker løst på kort sigt, men tillige de permanente elementer man opererer med i virksomheden. Det er væsentligt at metoden og organiseringen af den enkelte løsning skal ses i lyset af en langsigtet planlægning.

Anden del af definitionen fastslår, at etablering og kommunikation skal ske via et fælles basisprogrammel-system. På nedenstående figur ses den principielle placering af CF-SYSTEM i et informationssystem. Virksomheden kan via brugerprogrammer kommunikere med databasen, med CF-SYSTEM-procedurer som et indskudt administrerende organ og udskrive det ønskede beslutningsgrundlag til beslutningssystemet.


Beslutningssystem
initierende data
Beslutnings-grundlag
Informationssystem
Brugerprogrammer
CF-SYSTEM
Database
Styrende data
Registrende data
Udførende System

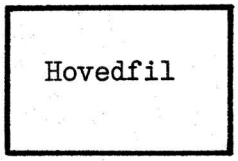
3. FILSTRUKTUR.

3.1 Hoved- og listefiler.

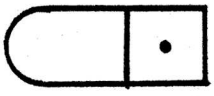
CF-SYSTEM består af en række RC 4000 algol/fortran standard procedurer, der kan etablere og behandle poster og sammenhænge mellem poster i filer på baggrundslagre med direkte tilgang og som kan varetage kommunikationen mellem brugerprogrammer og filer.

CF-SYSTEM lagrer de til et brugersystem hørende data i hovedfiler, som indeholder poster for enheder som varer, kunder, maskiner, ordrer, og lignende, altså fysiske og/eller abstrakte elementer. De forskellige elementer er knyttet sammen ved hjælp af nogle relationer. Disse relationer lagres i listefiler, hvor posterne indeholder information om sammenhænge mellem **elementer** i hovedfilerne.

En hovedfil er organiseret index-sekventielt. Postlængden kan væreriabel, og alle referencer til poster i filen sker ved brug af posternes nøgle. Ved en nøgle vil vi forstå en entydig identifikation af et element. I det følgende vil vi benytte følgende symbol for en hovedfil

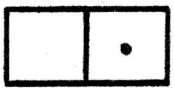


og følgende symbol for en post i en hovedfil

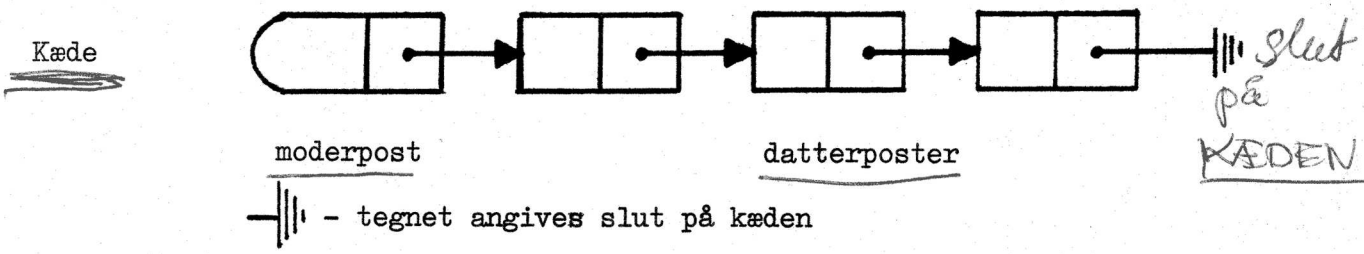


I en listefil kan postlængden være variabel, og reference til en post i en listefil kan kun ske via andre poster. I det følgende vil vi benytte følgende symbol for en listefil:

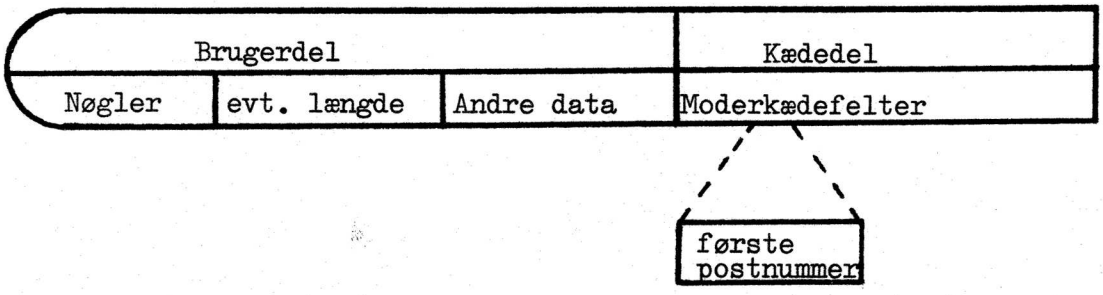
og følgende symbol for en post i en listefil :



Hvis posterne i en listefil er sammenhægtet i kæder vil vi anvende følgende symbol for en kæde:



Hovedfilposter består af en brugerdel og en kædedel. Brugerdelen indeholder brugerens data (bl. a. nøgler) samt en evt. længdeangivelse af brugerdelen. Kædedelen indeholder lige så mange moderkædefelter, som der er defineret kæder til. Et moderkædefelt indeholder postnummer for første post i den tilhørende kæde mellem poster i en tilknyttet listefil. Der kan være tilknyttet flere listefiler.

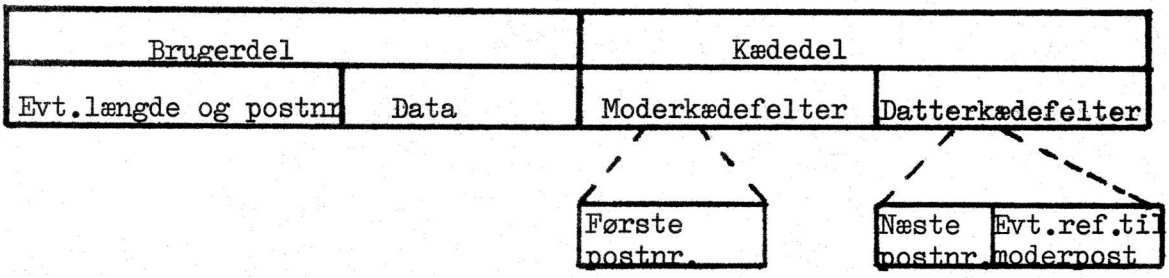


Struktur for en hovedfilpost.

Hvis posten ikke indeholder postnummer (først postlængde), er posten udelukkende identificeret ved sin placering i filen ! (åbenbart !?)

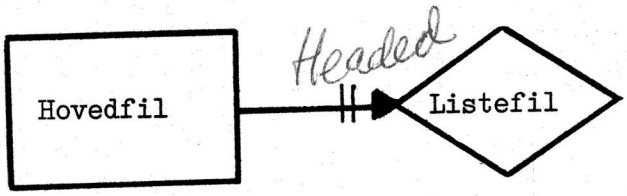
Listefilposter består ligeledes af en brugerdel og en kædedel. Brugerdelen indeholder brugerens data, og ved variabel postlængde tillige længdeangivelse og postnummer. Kædedelen indeholder henvisning til næste post i kæden. Hvis en post er første post i en kæde vil henvisningen til næste post være anbragt i et moderkædefelt ellers i et datterkædefelt.

Et moderkædefelt indeholder postnummer for første post i den tilhørende kæde mellem poster i en tilknyttet listefil. Der kan være tilknyttet flere listefiler. Et datterkædefelt indeholder postnummer for næste listefilpost i den aktuelle kæde (evt. markering af slut på kæde). Datterkædefelter kan evt. også indeholde en reference til moderposten for den aktuelle kæde. Der kan være flere moderkædefelter og datterkædefelter. Ligger moderposter i en hovedfil er referencen nøglen for den post i hovedfilen, hvortil listefilposten er kædet. Ligger moderposten i en listefil er referencen postnummer for den post i listefilen, hvortil listefilposten er kædet.

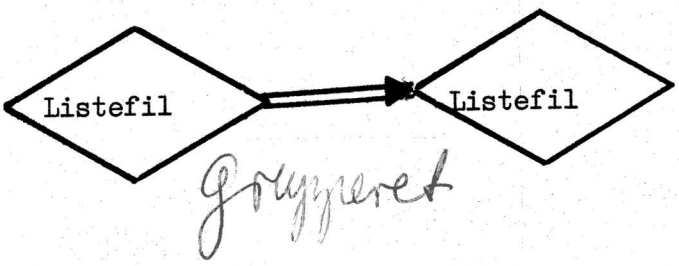


Struktur for Listefilpost

Indeholder datterkædefelter i en listefil en reference til moderposten for den aktuelle kæde, siges posten at være headed. Denne egenskab kan grafisk vises ved en dobbeltstreg over forbindelseslinien mellem de to filer i filstrukturen.



Såfremt man i overvejende grad har anbragt posterne fysisk sekventielt i en kæde, siges kæden at være grupperet. Dette vises grafisk med en dobbelt pil i kæderetningen.



Søgetiden fra post til post i en kæde er mindre i en grupperet kæde end i en kæde, der ikke er grupperet, hvorfor man normalt vil gruppere de mest benyttede kæder. Det skal fremhæves, at det kun er muligt at gruppere een kæde pr. listefil.

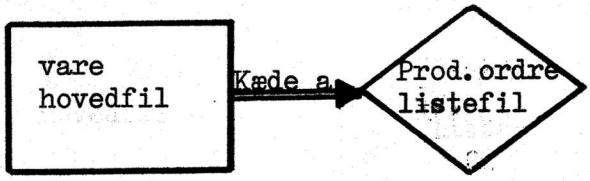
(Post)
En kæde kan aldrig godt være både Headed og grupperet.

3.2 Brug af CF-SYSTEM

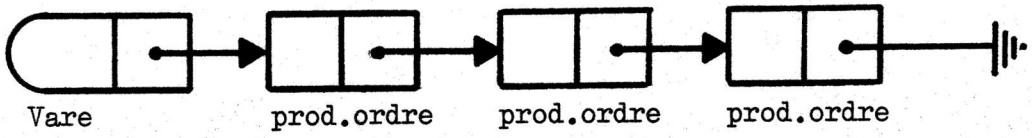
Listestruktur.

Man kan benytte en listefil til at afbilde begivenheder eller suplerende oplysninger vedrørende et angivet element. Betragter man f. eks. de til en vare knyttede produktionsordrer, vil disse kunne afbildes på denne måde, idet produktionsordrerne indeholder oplysninger om mængder og tidspunkter. En sådan brug af CF-SYSTEM benævnes en listestruktur og benyttes altså til at afbilde variable egenskaber ved et element. Kædedelen indeholder kun en henvisning til næste post i kæden, eller en slutmarkering.

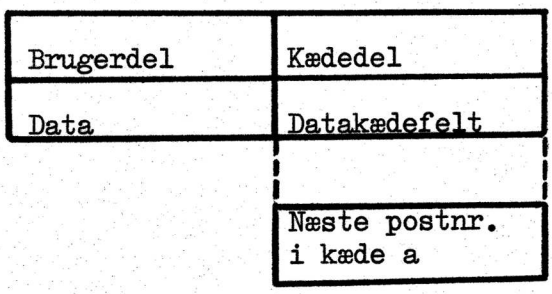
Filstruktur



Kædestruktur



Poststruktur (listefil)

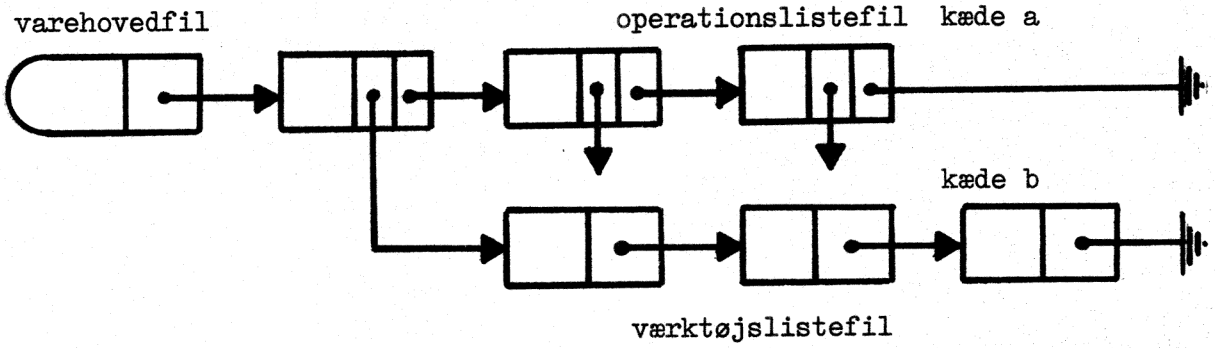


Et specialtilfælde af en listestruktur fremkommer når man sammenkæder to listefiler. Betragter man f. eks. de til en resulterende vare benyttede operationer, kan man være interesseret i at anbringe de til en operation benyttede værktøjer i en særlig listefil, der er sammenkædet med operationsfilen. På omstående figur er de resulterende varer anbragt i hovedfil, operationerne og værktøjerne i hver sin listefil. Listefilen, der indeholder operationerne er her forsynet med et moderkædefelt og et datterkædefelt.

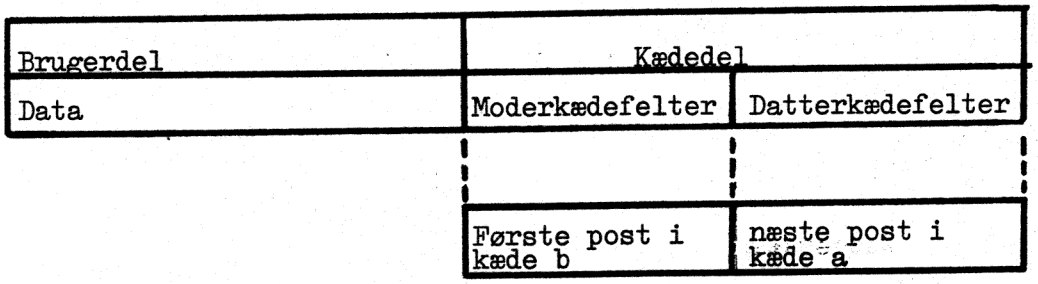
Kæde a indeholder operationer per resulterende vare, kæde b værktøjer per operation.

Filstruktur

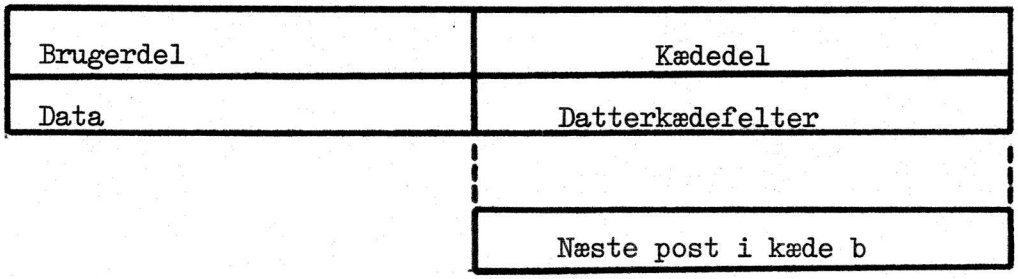
Kædestruktur



Poststruktur for operationslistefil



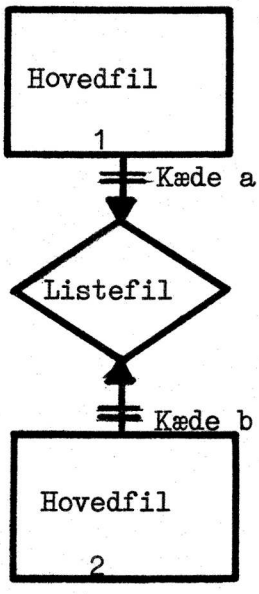
Poststruktur for værktøjslistefil



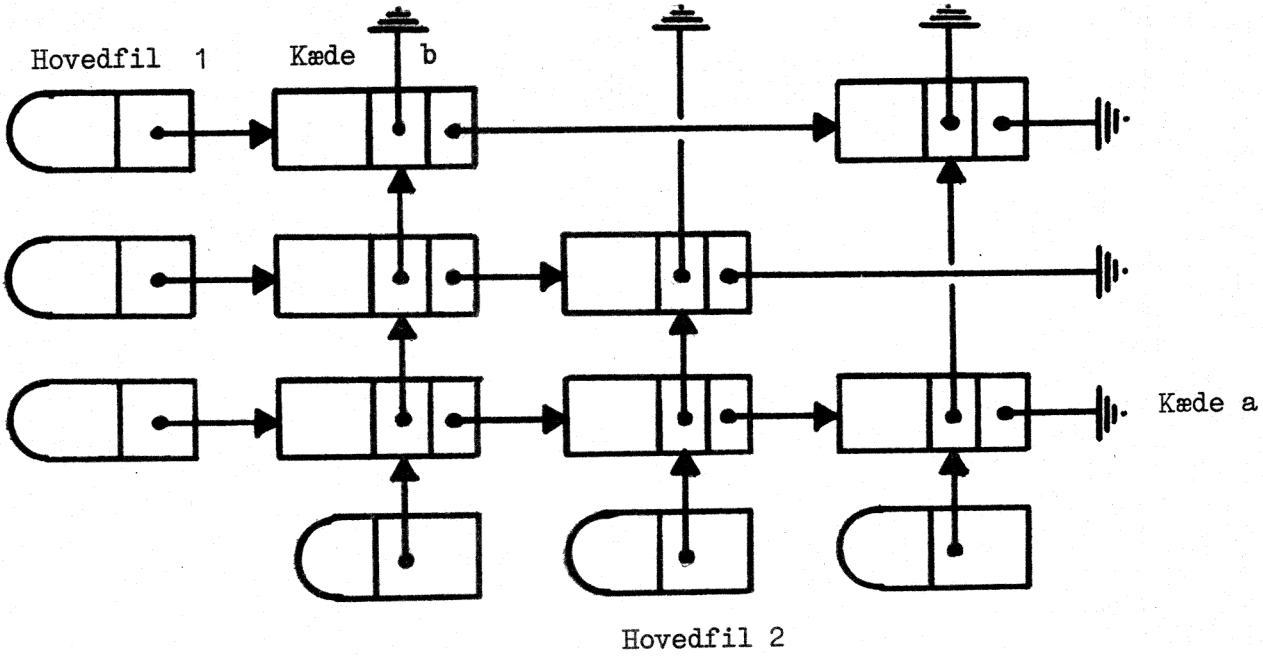
Forbindelsesstruktur.

Ønsker man at sammenkæde posterne i to forskellige hovedfiler, dannes en forbindelseslistefil med to datterkædefelter per post, eet for hver hovedfil. Ved denne sammenkædning er det muligt at referere til listefil-posterne via begge hovedfilerne. Kommer man til en kæde mellem poster i listefilen via den ene hovedfil, f.eks. kæde a på nedenstående figur, kan man ved at gennemløbe disse poster finde nøglerne for posterne i den anden hovedfil i den til denne fil hørende reference i datterkædefeltet. Den samme fremgangsmåde kan også anvendes for den anden hovedfil.

Filstruktur

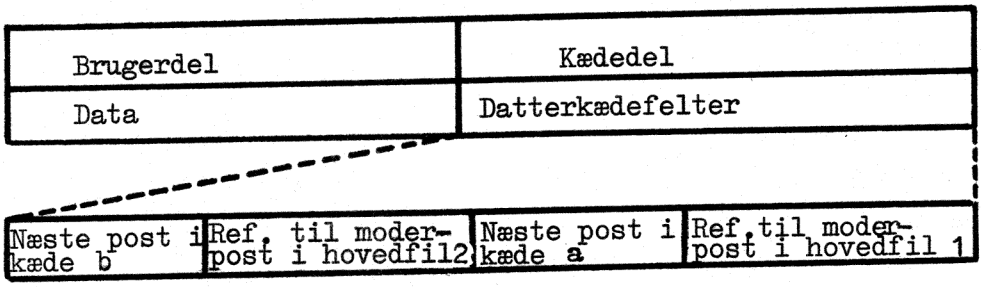


Kædestruktur



Højre kædefelt hører til kæde a, venstre kædefelt hører til kæde b.

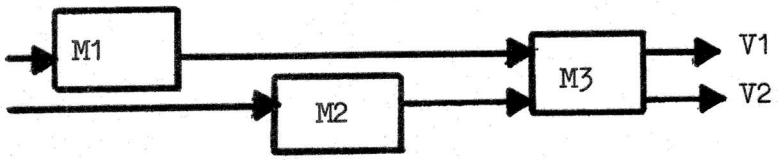
Poststruktur i listefil



Som eksempel på anvendelse af en forbindelseslistefil kan følgende nævnes: For at kunne fremstille beslutningsgrundlag for produktionsprocessen ønsker en virksomhed at anbringe oplysninger om sine resulterende varer, sine maskiner samt sammenhængene mellem disse i sin database. Sammenhængene skal udtrykkes som hvilke maskiner en færdigvare må passere samt hvilke færdigvarer, der passerer en maskine.

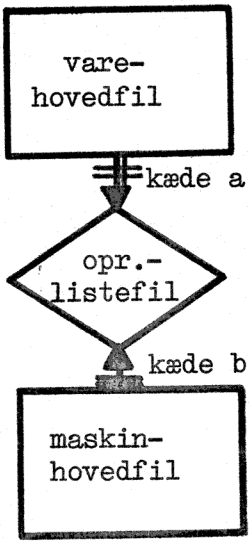
Vi går ud fra, at virksomheden skal fremstille de to varer V1 og V2 på maskinerne M1, M2 og M3. Som det ses på nedenstående figur må V1 bearbejdes på M1 og M3, medens V2 bearbejdes på M2 og M3.

Fysisk struktur:



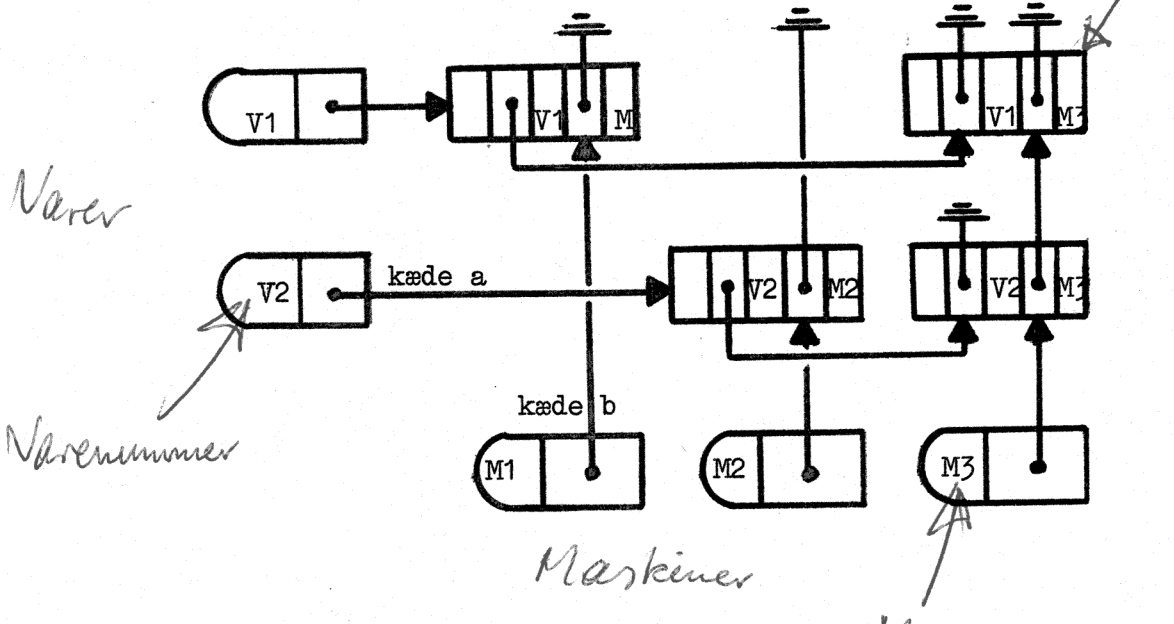
Varer og maskiner anbringes i hver sin hovedfil, medens forbindelsen mellem disse, som er operationer, anbringes i en forbindelseslistefil, der sammenknytter de to hovedfiler. På omstående figurer ses filstruktur og kædestruktur. Det fremgår, at en hovedfilpost i brugerdelen har varenummer/maskinnummer som nøgle. I listefilens kædedel er der to datterkædefelter. Referencer til de to hovedfiler er angivet som vare- og maskinnummer. De venstre kædefelter hører til kæder, der indeholder indeholder information om, hvilke maskiner en vare må passere (kæde a). De højre kædefelter hører til kæder, der beskriver hvilke varer, der passerer en maskine (kæde b).

Filstruktur

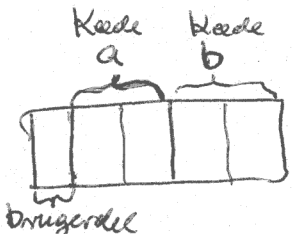


Kædestruktur

Hvert listefil-element er en operation



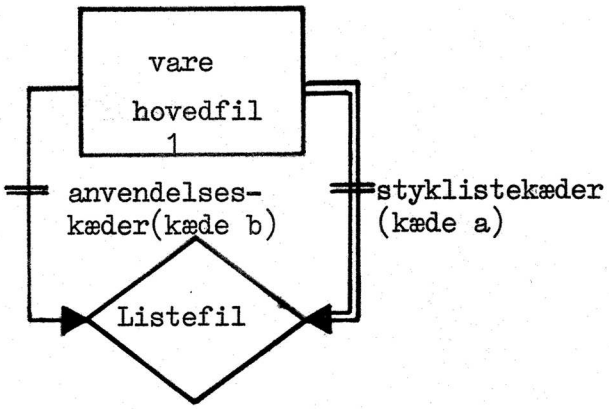
Listeelement:



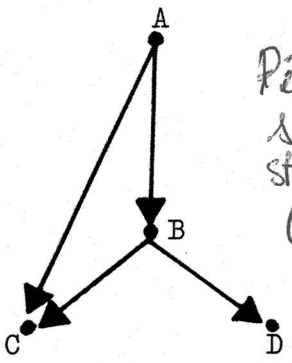
Netværksstruktur

Hvis man ønsker at sammenkæde posterne i een og samme hovedfil, f.eks. for at beskrive nedbrydning af varestrukturer, benyttes den samme teknik, idet de to datterkædefelter i listefilen refererer til samme hovedfil. Hvis der anvendes styklistekæder og anvendelseskæder kan man afbilde filstruktur, kædestruktur og poststruktur som vist på de følgende figurer. På figurerne er a-kæder styklistekæder og b-kæder anvendelseskæder.

Filstruktur

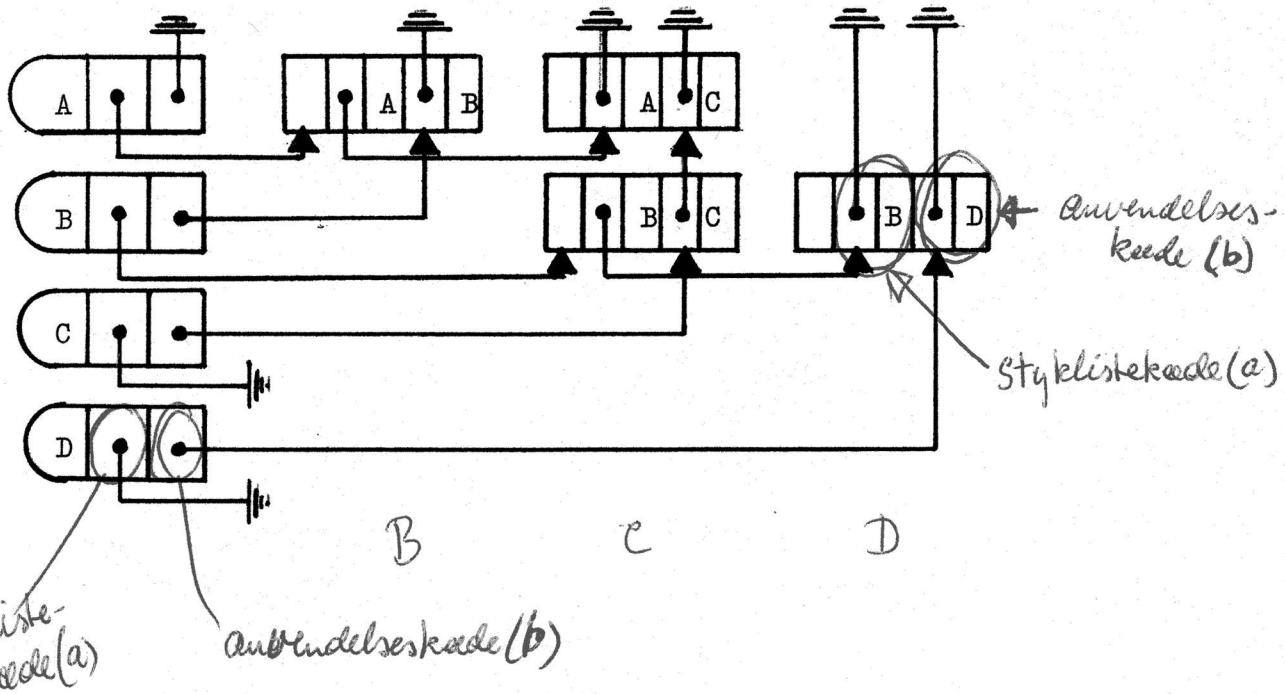


Varenetværk

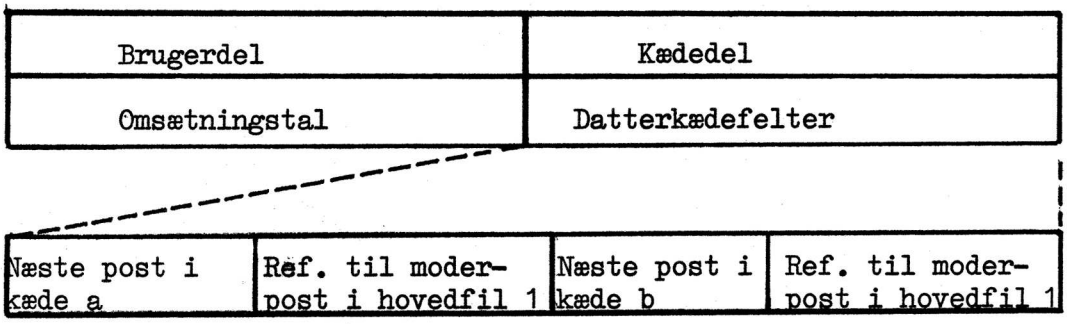


Pilene peger som ved styklistekæden (dvs. vare A har en stykliste best. af Bog C)

Kædestruktur



Poststruktur(listefil)



Hvis vi forestiller os, at varenetværket er som vist på forrige side, vil kædestrukturen få det viste udseende. De kæder, der benytter venstre kædefelt afbilder styklistekæder, hvor der er foretaget en nedbrydning et niveau ned. De kæder, der benytter højre kædefelt afbilder anvendelseskæder med anvendelser et niveau op. Kommer man til en styklistekæde mellem poster i listefilen via et varenummer i hovedfilen, kan man ved at gennemløbe disse poster finde nøglerne for posterne i de indgående varer et niveau ned. Tilsvarende gælder for en anvendelseskæde. Det ses, at posterne i listefilen afbilder pilene i varenetværket. Brugerdelen i posterne kan anvendes til lagring af omsætningstal.

Konklusion

Vi har i det foregående gennemgået en række simple grundlæggende filstrukturer. En database, som skal udgøre grunddata for et større integreret system, vil normalt være baseret på en langt mere kompleks filstruktur. Omstående er vist en database til integreret produktions- og økonomistyring. Databasen er taget med for at vise hvorledes en kompleks filstruktur kan opbygges af de grundlæggende filstrukturer.

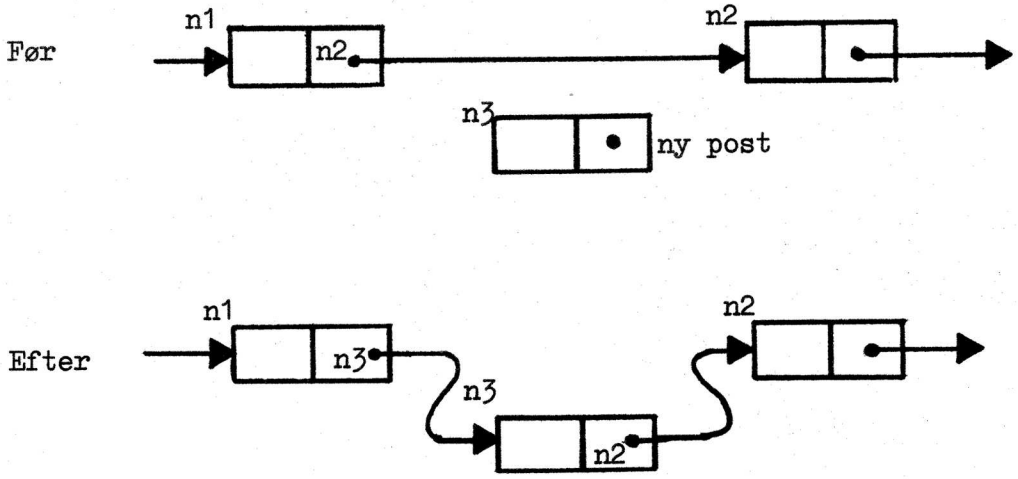
4. CF-PROCEDURER

Der eksisterer en række procedurer i CF-SYSTEM. Ved brug af disse procedurer, er det bl. a. muligt at indsætte, forbinde, opsøge og slette poster. I det følgende vil disse fire funktioner blive nærmere beskrevet samtidig med at de vigtigste procedurer nævnes.

Indsættelse af poster

Indsættelse af poster i en hovedfil foretages med proceduren: insert_m (zm, record). Her er zm den pågældende hovedfil og record den pågældende post.

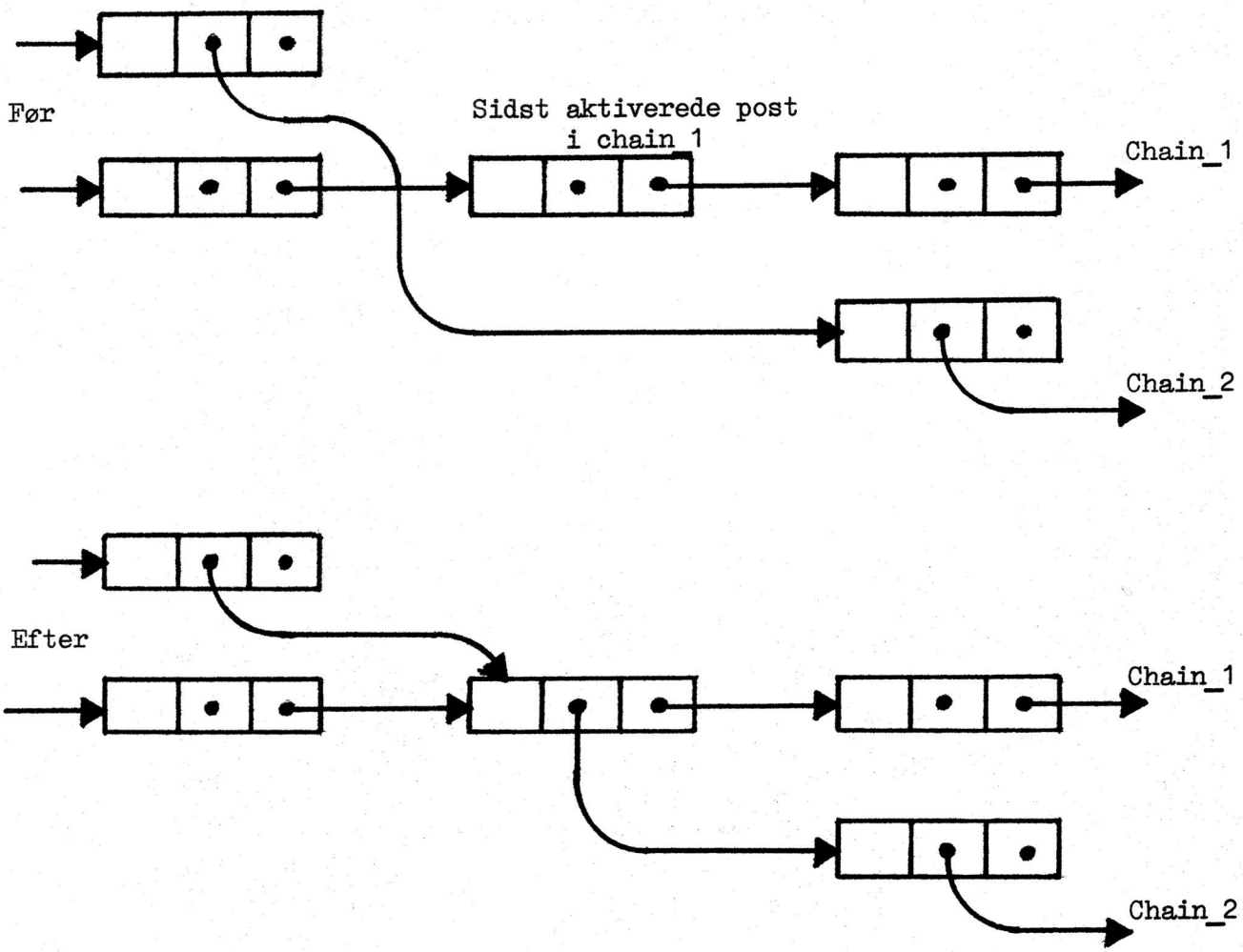
Indsættelse af poster i en listefil foretages med proceduren: insert_l (zl, chain, icmode, record). Her er zl den pågældende listefil, chain den kæde, posten skal indsættes i, icmode den plads i kæden, hvor posten skal anbringes, record den pågældende post. For at få en mere indgående forståelse af, hvorledes proceduren arbejder, vil vi betragte følgende figur med kædestruktur før og efter indsættelse.



Indsættelse af en ny post i en kæde mellem post n1 og n2 foretages således: Først finder CF-SYSTEM et frit postnummer til den nye post. Dette foregår efter en grupperingsstrategi, idet CF-SYSTEM forsøger at finde et postnummer, således at den nye post bliver placeret i samme fysiske blok som n1. Herefter indsættes posten i kæden ved at postnummeret i post n1's kædefelt overføres til den nye post n3's kædefelt, hvorefter postnummeret i post n1's kædefelt ændres til postnummeret for den nye post n3.

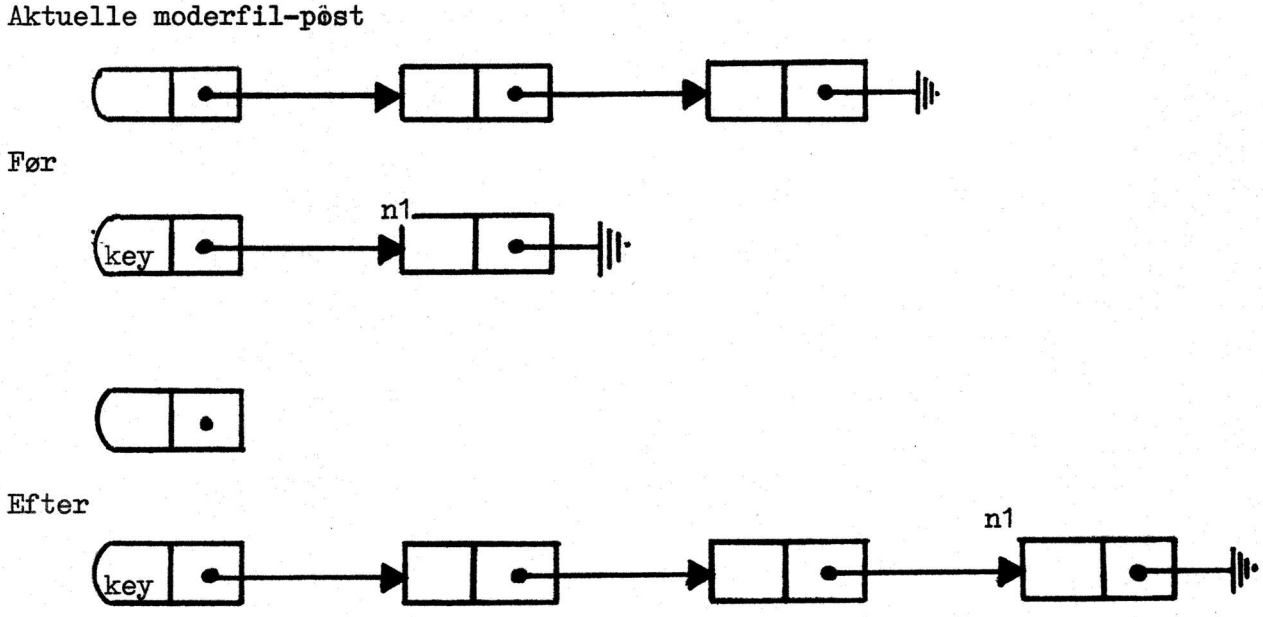
Forbindelse af poster

Når en post skal tilhøre flere kæder, må den indsættes i disse via proceduren : connect (zl, chain_1, chain_2, icmode). Her er zl den listefil, i hvilken den pågældende post er placeret, chain_1 den kæde, hvis sidst aktiverede post nu også skal tilhøre en anden kæde, nemlig chain_2, icmode den plads i chain_2, hvor sidst aktiverede post i chain_1 skal anbringes.



connect-proceduren har principielt samme funktion som insert_1-proceduren. Den væsentligste forskel ligger i angivelsen af postnummeret for den post, der ønskes indsat. I insert_1-proceduren vælges postnummeret efter en grupperingsstrategi, hvorved man forsøger at placere den nye post i samme fysiske blok, som den foregående post i kæden (posterne siges at være grupperet). I connect-proceduren er postnummeret for den post, der skal indsættes i chain_2 allerede fastlagt ved postens placering i chain_1 (posterne kan ikke grupperes).

Såfremt man ønsker at flytte en hel kæde fra en moderpost til en anden post i samme fil, anvendes proceduren : move_chain (z, chain, key). Her angiver z moderfilen, som altså enten kan være en hovedfil eller en listefil. Chain angiver den pågældende kæde, der skal flyttes, og key identificerer den post, hvortil kæden skal flyttes. Såfremt der i forvejen eksisterer kædeposter til den nye moderpost, vil de flyttede kædeposter blive anbragt foran disse. Nedenfor ses en figur af denne funktion.



Opsøgning af poster

Under udnyttelsen af databasen har man behov for at kunne opsøge poster i hoved- og listefiler. Opsøgning af en post i en hovedfil med specificeret nøgle kan foretages med proceduren: get_m (zm, key). Her angiver zm hovedfilen og key den specificerede nøgle. Proceduren : next_m (zm) opsøger den næste post i hovedfilen.

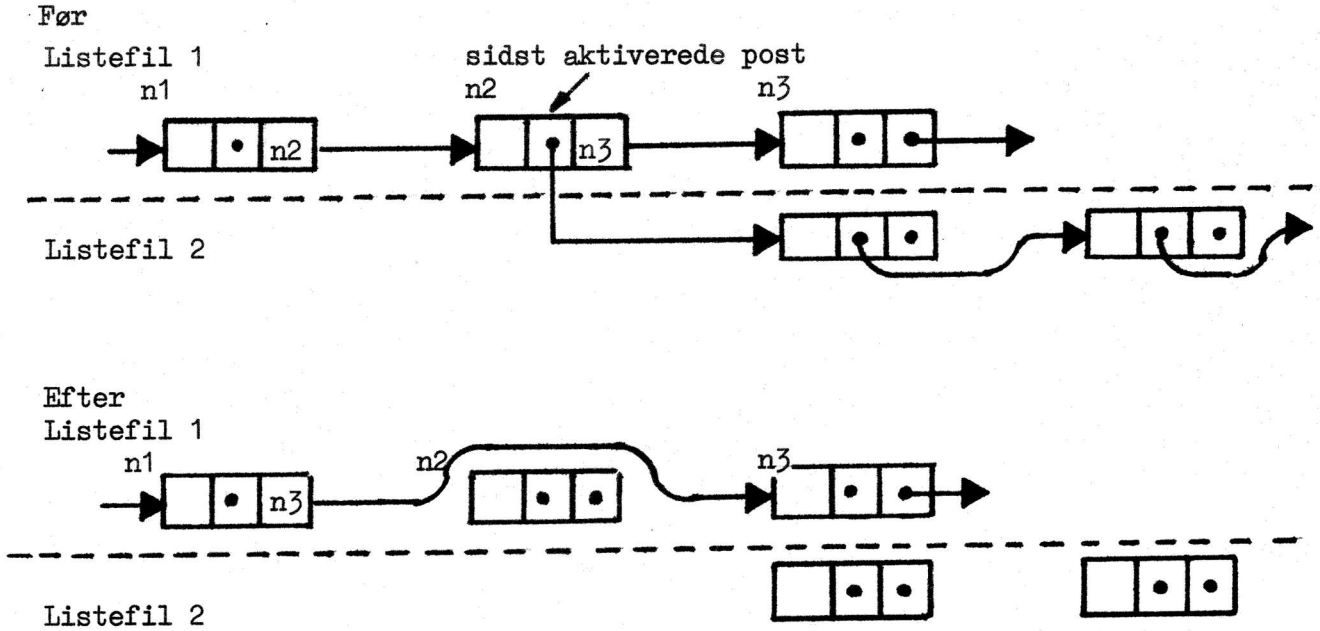
Opsøgning af en post i en listefil med specificeret postnummer kan foretages med proceduren: get_numb_l (zl, rec_no). Her angiver zl listefilen og rec_no postnummeret. Opsøgning af en post i en listefil efter dens plads i kæden, kan foretages med proceduren: get_l (zl, chain, gmode). Her angiver zl listefilen, chain den pågældende kæde og gmode posten i kæden. gmode=1 giver første post, gmode=2 giver næste post efter den sidst aktiverede post i kæden og gmode=3 sidst aktiverede post i kæden.

Rækkefølgen for poster i en kæde er givet ved den rækkefølge, de er sat ind i.

Sletning af poster

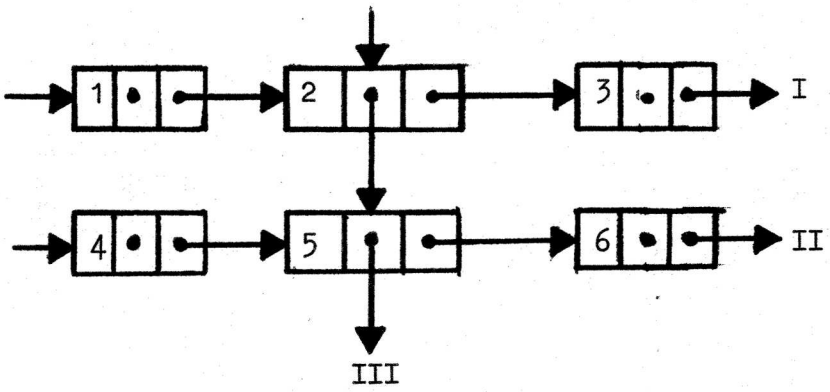
Når oplysninger i databasen er blevet forældet eller måske oprettet fejlagtigt, er der behov for at kunne slette dem igen. Hvis man ønsker at slette en post i en hovedfil og alle poster i kæder, der har hovedfilposten som moderpost, kan man anvende proceduren : delete_m (zm). Her angiver zm den pågældende hovedfil. Den hovedfilpost, der slettes, er den sidst aktiverede post i den pågældende fil. Delete_m er en "alvorlig" procedure, da de pågældende poster kan indgå i kæder til poster i andre hovedfiler.

Hvis man ønsker at slette en post i en listefil og alle poster i kæder, der har listefilposten som moderpost, kan man anvende proceduren: delete_l (zl, chain). Her angiver zl den pågældende listefil, chain den kæde, som posten er aktiveret af. Den post i listefilen, der slettes, er den sidst aktiverede post i den pågældende fil. For at få en mere indgående forståelse af, hvorledes proceduren arbejder, vil vi betragte følgende figur med kædestruktur før og efter en sletning.



Hvis listefilpost n2 kun indgår i en enkelt datterkæde slettes n2 ved at ændre postnummeret i post n1's kædefelt til postnummer n3.

Da CF-SYSTEM anvender énvejs kædning er det ikke muligt at fjerne en post endeligt, hvis den indgår i flere kæder. Dette skyldes, at CF-SYSTEM ikke kender foregående post i alle de kæder, som posten indgår i, og derfor ikke kan lede hanvisningerne uden om den nedlagte post. Derfor "dødsmærkes" en listefilpost, når den ønskes slettet. Når de til den pågældende post hørende kæder gennemløbes i anden anledning, etableres de nye henvisninger og posten bliver fysisk slettet, når den sidste kæde gennemløbes. Hvis man i nedenstående listefil dødsmærker post 5 under gennemløb af kæde II, vil alle henvisninger være retableret, når kæde III er gennemløbet.



Hvis man ønsker at slette alle poster i en kæde og alle poster i kæder, der har moderposter i den specificerede kæde, kan man anvende proceduren: delete_chain (z, chain). Her angiver z den fil, som indeholder moderposten for den specificerede kæde, benævnt chain. I modsætning til delete_m og delete_1 slettes selve moderposten altså ikke. Det må bemærkes, at en moderpost ikke kan slettes, uden at det får følger for "efterkommerne".

5. KONSTRUKTION AF DATABASE

Konstruktion af en database kan baseres på 2 niveauer:

I. Opstilling af en virksomhedsmodel

II. Transformation af model til database

5.1 Opstilling af en virksomhedsmodel

1) Primær afgrænsning af det administrative område.

2) Analyse af det administrative problem med henblik på at lave en almengyldig datamæssig beskrivelse af virksomheden, der kan anvendes på lang sigt.

   Denne beskrivelse kan foretages ved:

   a) Udskille de fysiske elementer, som udgør konkrete fænomener.

   b) Udskille sammenhænge mellem de fysiske elementer, som udgør abstrakte fænomener.

5.2 Transformationen af model til database under hensyntagen tilstedeværende programmel.

1) En foreløbig database opnås ved at anbringe fysiske elementer i hovedfiler og sammenhænge i listefiler.

2) Komprimering og justering.

Som er konsekvens af den anvendte fremgangsmåde ved opstillingen af virksomhedsmodellen og ved transformationen fra model til foreløbig database, vil databasen sandsynligvis bestå af et større antal filer. For at kunne arbejde med et praktisk gennemførligt system (begrænset ferritlagerplads bl.a.) må vi nedbringe antallet af filer. Man kunne formulere denne opgave som at minimere antallet af direkte sammenkoblede filer under hensyntagen til et minimalt tab af faciliteter i informationssøgningen.

Konsekvenserne ved fjernelse af en fil er principielt følgende:

Hvad tabes ?

- tab af direkte tilgangsmulighed ved, at et fysisk element lagres som relation eller egenskab, dvs. der må foretages en søgning via andre fysiske elementer.

- relationer imellem to fysiske elementer mangler, dvs. der må foretages søgning via et tredie fysisk element, der er forbundet til begge.

- forbindelsesstukturer må eventuelt ændres til listestrukturer eller netværksstrukturer.

Hvad vindes ?

Til hver fil, der anvendes af et givet program, er der knyttet et ferritlagerområde, en zone med plads til buffere og styrende data, bl.a. kædebeskrivelser. For hver fjernet hovedfil vindes normalt 3000 - 4000 bytes, og for hver fjernet listefil 2000 - 3000 bytes.

6. EKSEMPEL PÅ KONSTRUKTION OG ANVENDELSE AF EN DATABASE.

6.1 Bestemmelse af fysiske elementer og relationer

På grundlag af en beskrivelse af en virksomhed og dens administrative opgaver foreligger der et grundlag for konstruktion af en database.

De fysiske elementer, der er udskilt under beskrivelsen, er sammensat af en nøgle, der identificerer det fysiske element og en række tilknyttede egenskaber. Under beskrivelsen vil der desuden være udskilt en række relationer mellem de fysiske elementer.

Ved at anbringe de udskilte fysiske elementer i hovedfiler og relationerne i listefiler når vi frem til følgende filer:

Hovedfiler:

Varefil:

Nøgle : Varenummer
Egenskaber : Varebeskrivelse
Lagerbeholdning

Kundefil:

Nøgle : Kundenummer
Egenskaber : Kundebeskrivelse

Leverandørfil:

Nøgle : Leverandørnummer
Egenskaber : Leverandørbeskrivelse

Operationsstedfil:

Nøgle : Operationstednummer
Egenskaber : Operationsstedbeskrivelse

Afdelingsfil:

Nøgle : Afdelingsnummer
Egenskaber : Afdelingsbeskrivelse

Listefiler:

Varestrukturfil

Kvantum hvormed en vare indgår i en anden vare

Vare/Kundefil

Kvantum af en vare til en kunde
Leveringstermin
Leveringsmåde

Vare/lev. fil

Kvantum af en vare fra en leverandør
Bestillingstermin

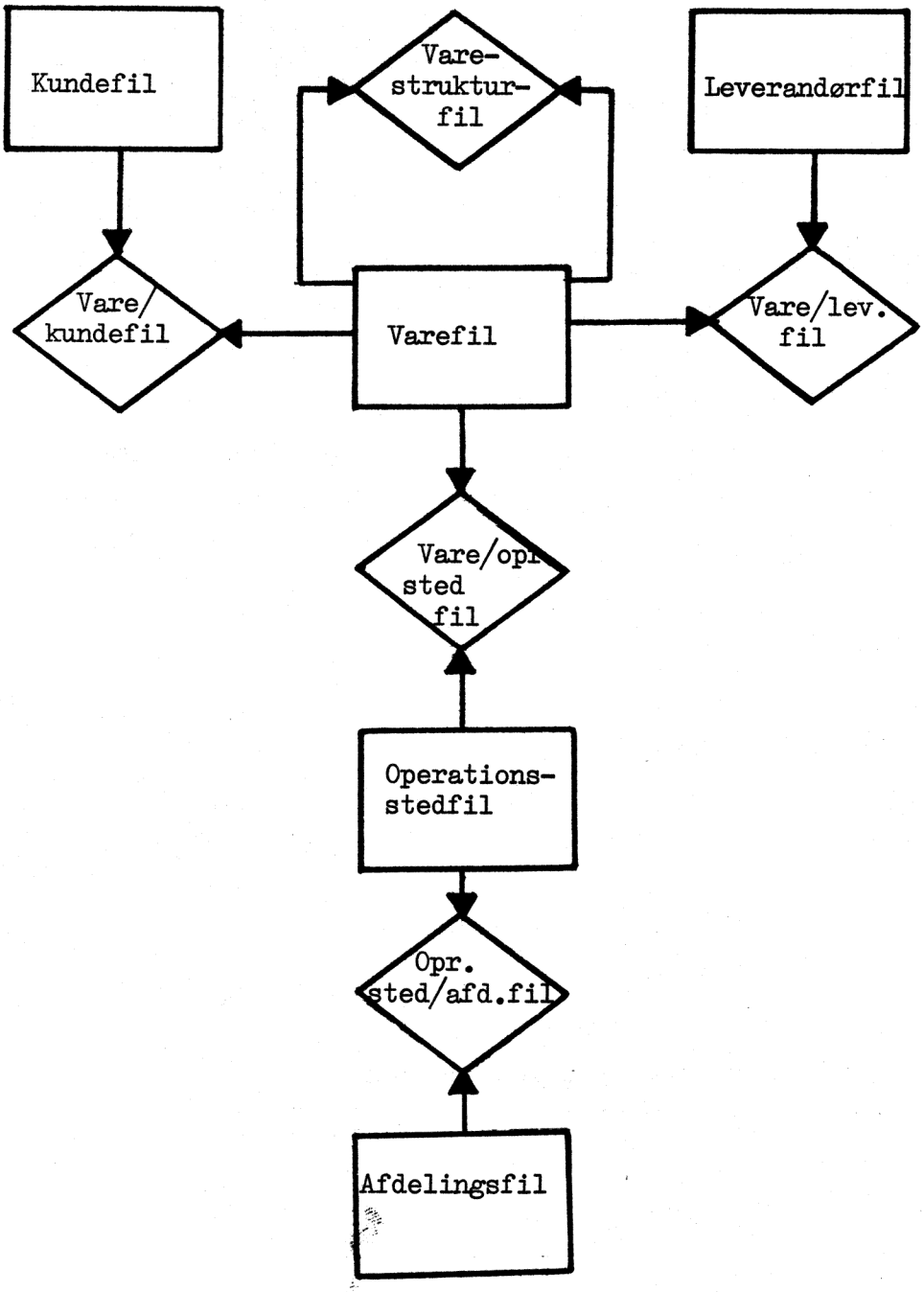
Vare/operationsstedfil

Operationsnummer
Tidsforbrug

Operationssted/afdelingsfil

Operationssted i en afdeling

Foreløbig database



b) Frekvens : Benyttes denne relation med høj eller lav frekvens.

eks. Benyttes relationen ved hver kørsel eller er det sjældent.

c) Selektivitet : Er efterspørgslen stærkt selektiv (omfattende nogle få entiteter og relationer) eller svag selektiv (omfattende hele hovedfilen eller veldefinerede grupper deraf sammen med alle relationer).

eks. Sammenstilles kun nogle få råvarer og leverandører eller sammenstilles alle råvarer med de respektive leverandører.

Såfremt der i en given situation er

- kraftig struktur
- Høj frekvens
- Svag selektivitet

er det hensigtsmæssigt med en forbindelsesstruktur mellem de to hovedfiler.

Hvis man trods fjernelse af en forbindelseslistefil mellem to hovedfiler alligevel har behov for en vis relation kan denne i stedet tilføjes som egenskaber i de pågældende hovedfiler.

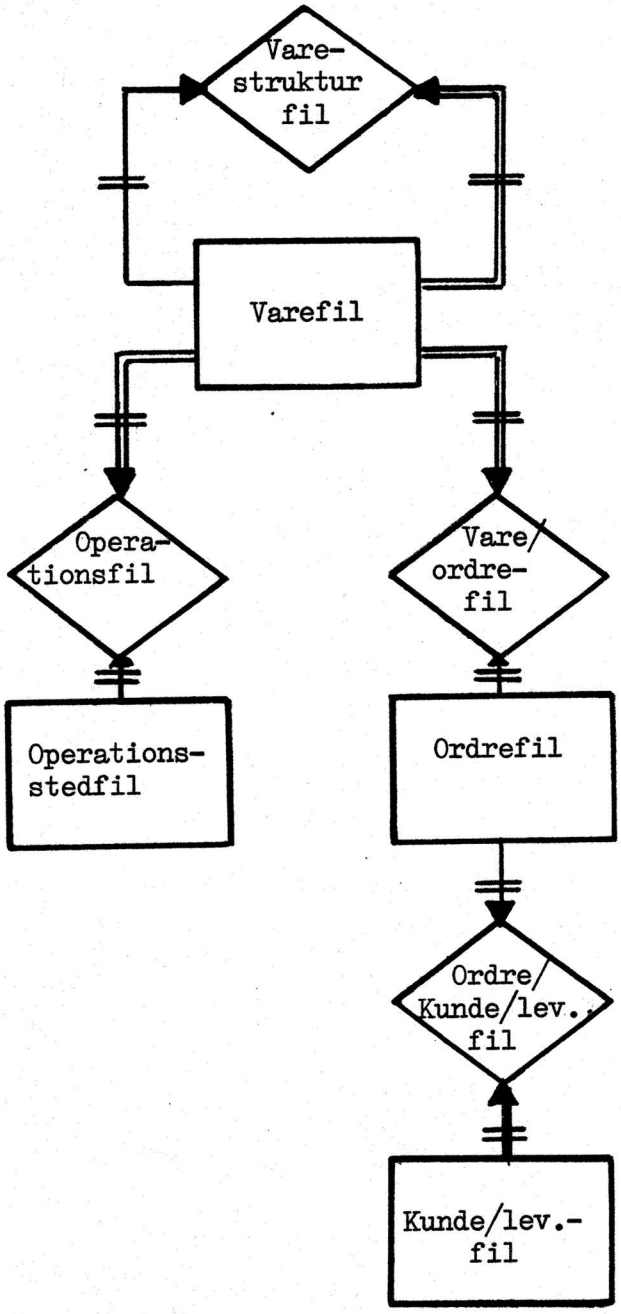
Indsættelse af en hovedfil.

Såfremt man har behov for at have direkte adgang til en størrelse, som er anbragt som relation i en listefil, kan man indsætte en hovedfil med den aktuelle størrelse som nøgle. Hvis man har behov for at kunne finde en ordre direkte via ordrenummeret kan man oprette en ordrefil mellem varefil og kunde/lev.fil. Denne fil vil også kunne benyttes til produktionsordrer.

Endelig database

Efter de anvendte komprimeringsoperationer og den anvendte justeringsoperation vil databasen få omstående udseende. Antallet af filer er reduceret fra 10 til 8 og den foreløbige database er justeret, så den kan opfylde de stillede krav.

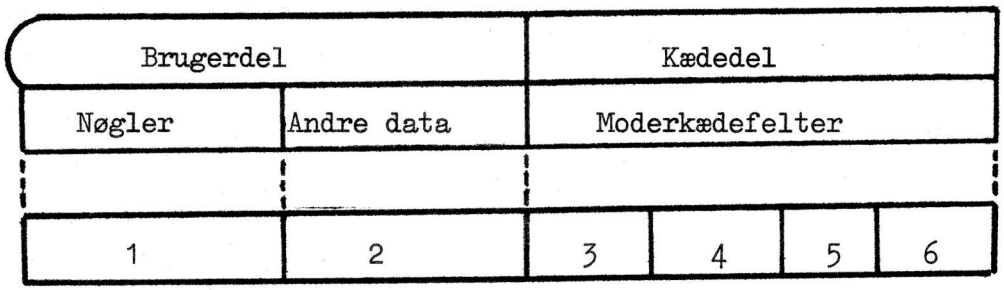
Endelig database



6.3 Indhold i filposter.

I det følgende vil vi for hver fil i databasen beskrive indholdet i brugerdel og kædedel.

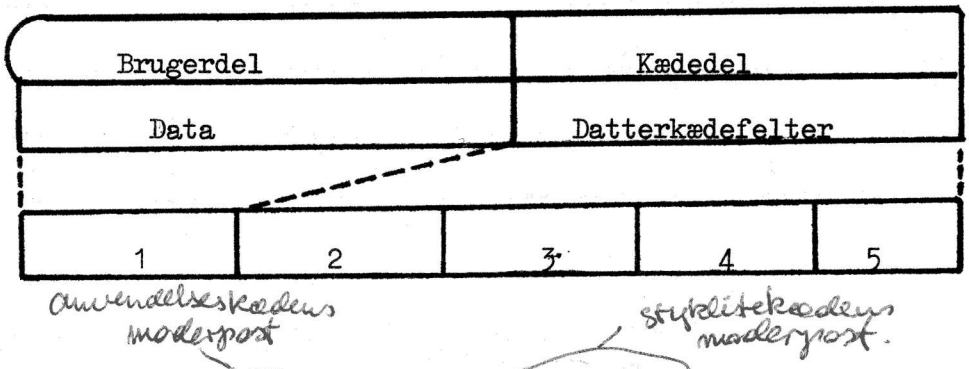
Varepost.



1. Varenummer.
2. Varebeskrivelse
   Lagerbeholdning.
3. Første postnummer i kæde med indgående varer (styklistekæde).
4. Første postnummer i kæde med varer, hvor denne vare indgår (anvendelseskæde).
5. Første postnummer i kæde med indgående operationer.
6. Første postnummer i kæde med ordrer, hvor denne vare indgår.

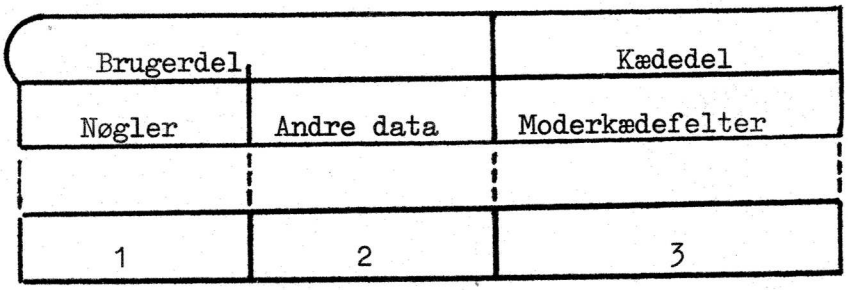
Varestrukturpost.

SMd s. 3,11



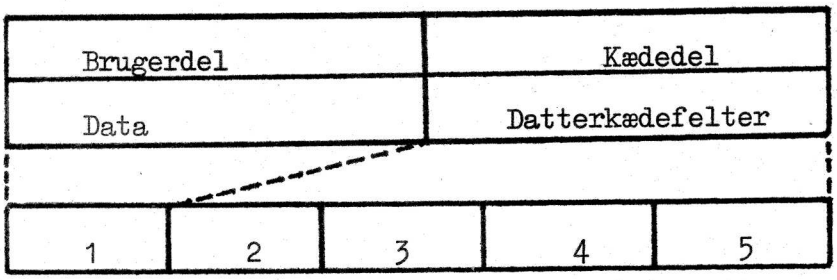
1. Kvantum, hvormed varen indgår i en anden vare.
2. Næste postnummer i styklistekæden.
3. Varenummer for styklistekædens moderpost i varefilen.
4. Næste postnummer i anvendelseskæden.
5. Varenummer for anvendelseskædens moderpost i varefilen.

Kunde/leverandørpost:



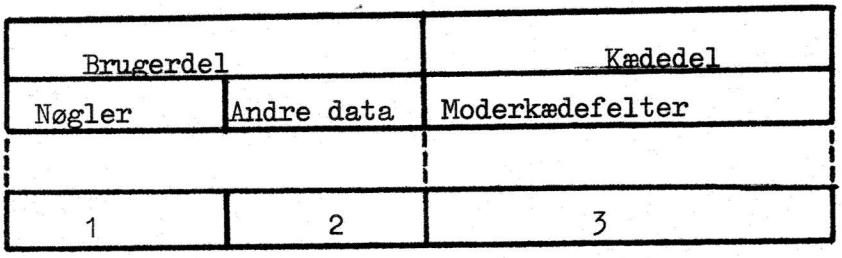
1. Kunde/leverandørnummer.
2. Beskrivelse af kunde/leverandør.
3. Første post i kæde med ordre for en kunde/lev.

Ordre/Kunde/lev. post.



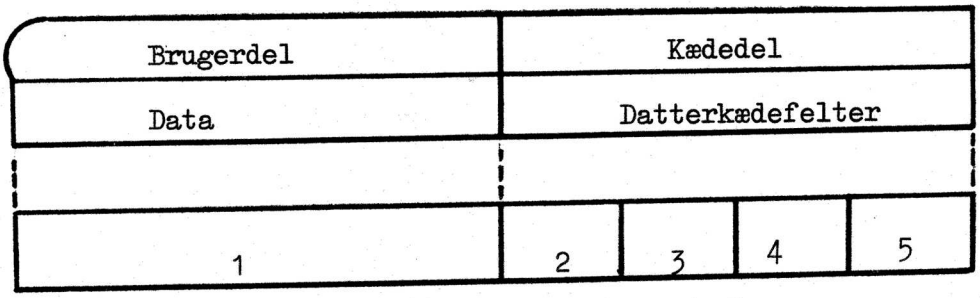
1. Blank
2. Næste post i kæde med ordre for en kunde/lev.
3. Kundenr./lev.nr. for moderposten i kunde/lev. fil
4. Slut på kæden. ← Fordi der til en ordre er knyttet én og kun én kunde/leverandør
5. Ordrenr. for moderposten i ordrefilen

Operationsstedpost.



1. Operationsstednummer.
2. Operationsstedbeskrivelse
   Afdelingsnummer
   Afdelingsbeskrivelse
3. Første post i kæde med varer til dette operationssted.

Vare/operationsstedpost.



1. Operationsnummer
   Operationsbeskrivelse
   Tidsforbrug
2. Næste post i kæde med operationssteder for vare.
3. Varenummer for operationsstedkædens resulterende vare (moderpost i varefilen).
4. Næste post i kæde med varer der fremstilles på operationsstedet.
5. Operationsstednr. for moderpost i operationsstedfilen.

6.4 Udskrift af outputbilag.

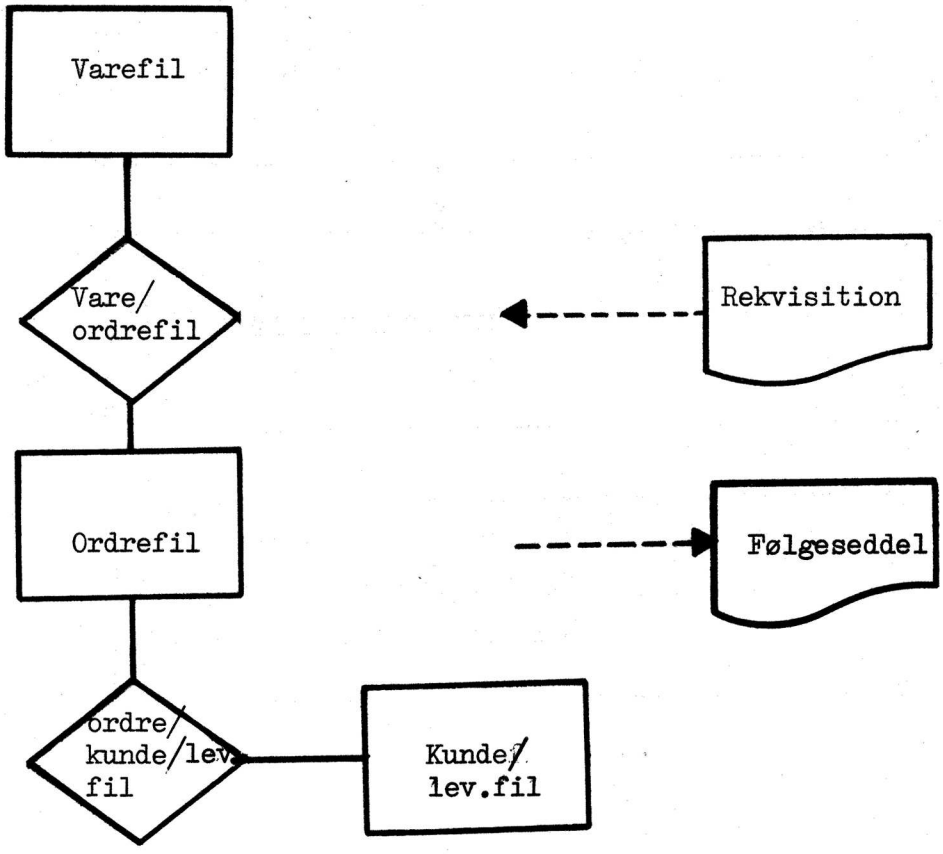
I det følgende ses hvilke dele af databasen, der er i brug ved udskrift af nedenstående følgeseddel.

FØLGESEDDEL

5) Kundenr. : xxxxx
6) Kundebeskrivelse: xx - xx
1) Ordrenr. : xxxxx
2) Ordrebeskrivelse: xx - xx
3) Lev. termin: xxxx
4) Leveringsmåde: xx - xx

7) Varenr. xxxxxx
9)Varebeskrivelse: xx - xx
8) Kvantum: xx

Database



APPENDIX

A. BEGRÆNSNINGER, TIDER OG PLADSBEHOV.

I dette afsnit diskuteres forbruget af de tre ressourcer: ferritlager, køretid og baggrundslager.

Formålet med diskussionen er udelukkende at bibringe læseren et skøn over omkostningerne ved brug af CF-SYSTEM; der forekommer dog en del detaljer, som det kan anbefales førstegangslæseren at gå let hen over.

CF-programmellet sætter ingen grænser for antal af filer og kæder i en filkonfiguration.

Hovedfilerne tillader frit valg af antal, type og position af nøglefelter, og af bucket- og blok-størrelser, medens der for listefiler er en begrænsning i største bloklængde (8 segmenter = 4096 bytes = 6144 iso-characterer) og dermed af største postlængde.

Begrænsningerne vil snarest ligge i hardware, d.v.s. baggrundslagerets størrelse og hurtighed, samt ferritlagerets størrelse.

I de følgende beregninger tages RC 4000's standard pladelager RC 433 som udgangspunkt. Pladelageret er delt op i 8000 segmenter, hver af længden 512 bytes (1 byte = 12 bit).

Ved en læsning eller skrivning transporteres et helt antal segmenter som en blok til eller fra RC 4000's ferritlager.

På et pladelager kan filer oprettes, nedlægges, forlænges og forkortes frit inden for pladelagerets samlede størrelse, dog således at der for pladelageret vil være defineret et segment-antal, hvoraf en filstørrelse skal være et multiplum (f. eks. 20 eller 40 segmenter).

Af betydning for transporttiden er inddelingen af pladelageret i 200 cylindre hver på 40 segmenter, idet pladelagerets magnethoveder skal placeres i den rette cylinder før en transport kan foregå.

Tilgangstiden til en blok inden for den aktuelle cylinder er 13 millisekunder (msec) i gennemsnit, medens et cylinder-skift tager mellem 33 og 150 msec.

A.1 Ferritlager-forbrug.

I et algol- eller fortran-program er ferritlageret delt mellem data og program, således at data-området indeholder de variable og input-output-buffere, som er erklæret i en given programdel, og program-området indeholder et variabelt antal programsegmenter. I program-området skal der helst være plads til de program-segmenter, der anvendes i den indre løkke i programdelen, idet hyppige læsninger af program-segmenter fra baggrundslager ellers vil forøge køretiden.

Man må altså i forbindelse med CF-SYSTEM regne med et ferritlager-forbrug svarende dels til de hyppigst anvendte CF-kodesegmenter, dels til de anvendte data-områder.

Ved læsning af poster og opdatering af felter inden for eksisterende poster anvendes maksimalt 6 CF-kodesegmenter (3100 bytes), hvis der derudover også oprettes og nedlægges poster, samt indsættes poster i kæder, anvendes op til 15 CF-kodesegmenter (7700 bytes).

Til hver fil, som anvendes af et givet program, er der knyttet et ferritlager-område, en zone, med plads til buffere og styrende data bl.a. kædebeskrivelser. Der skal ikke her gives detaljerede dimensioneringsregler for de to fil-typer, blot skal gennemgås fire eksempler på henholdsvis store og små, hovedfiler og listefiler.

Resultaterne i form af ferritlager-forbrug, nævnes her, så man evt. kan overspringe eksemplerne.

Stor hovedfil : 9.0 segmenter = 4600 bytes
Lille hovedfil : 4.8 segmenter = 2500 bytes
Stor listefil : 6.5 segmenter = 3300 bytes
Lille listefil : 3.3 segmenter = 1700 bytes

A.1.1 Stor hovedfil.

bloklængde : 2 segmenter = 1024 bytes
bucketstørrelse : 40 segmenter = 1 cylinder
samlet nøglelængde: 12 bytes
filstørrelse : 4000 segmenter = 1/2 pladelager
5 nøgler, variabel postlængde og 5 tilknyttede kæder.

ferritlager-forbrug =

filhoved + buckettabel + bloktabel + 2 * blok =
418 + 1600 + 512 + 2 * 1024 =

4578 bytes = 9.0 segmenter

Hvis der ikke indsættes nye poster i filen i det pågældende program, kan den ene blok spares, hvorved ferritlager-forbruget reduceres til 7 segmenter.

A.1.2 Lille hovedfil.

bloklængde : 1 segment = 512 bytes
bucketstørrelse : 40 segmenter = 1 cylinder
samlet nøglelængde : 8 bytes
filstørrelse : 2000 segmenter = 1/4 pladelager
5 nøgler, variabel postlængde og 5 tilknyttede kæder.

ferritlager-forbrug =

filhoved + buckettabel + bloktabel + 2 * blok =
304 + 600 + 512 + 2 * 512 =

2440 bytes = 4.8 segmenter

Dog 3.8 segmenter, hvis indsættelse af poster ikke forekommer i programmet.

A.1.3 Stor listefil.

bloklængde : 2 segmenter = 1024 bytes
filstørrelse : 4000 segmenter = 1/2 pladelager
5 indgående kæder hver med samlet nøglelængde 12 bytes for moder-filen.

A.2.1 Læsning af vilkårlig hovedfil-post.

Hovedfil som i A.1.1.
En hovedfil-post fremfindes ved tre søgninger.

a. søgning i bucket-tabel, herved findes den bucket, hvori posten ligger. Bucket-tabellen ligger permanent i lageret, så der er ingen transportomkostninger.

   (en bucket indeholder et fast antal blokke og en indholdsfortegnelse over disse, bloktabellen).

b. søgning i bloktabel, herved findes den blok, hvori posten ligger. Bloktabellen må læses ned i lageret, før søgningen kan foregå.

c. Søgning i blok, herved findes den ønskede post. Blokken må læses ned i lageret, før søgningen kan foregå.

   tidsforbrug =
   bloktabel + blok =
   (100 + 1 * 6.2) + (13 + 2 * 6.2) = 132 msec

Dette må tages som en gennemsnitstid, der gælder, hvis hele pladelageret anvendes i tilfældige spring.
Når blokken læses langt hurtigere end bloktabellen, skyldes det, at pladelageret efter læsning af bloktabellen er positioneret til den rette cylinder. Dette forudsætter, at bucket-størrelsen er valgt til cylinder-størrelsen, 40 segmenter, eller en ægte brøk heraf, f.eks. 20 segmenter, og at hovedfilens bucket-grænser er synkroniseret med pladelagerets cylinder-grænser. Hvis dette ikke er opfyldt, må man regne med ca. 50 msec ekstra.

Valg af bucket-størrelse har også indflydelse på ferritlager-forbruget, idet bucket-tabellen fylder proportionalt med antallet af buckets, og bloktabellen fylder proportionalt med antallet af blokke i en bucket, dog i spring på hele segmenter.
(proportionalitets-faktoren er : 4 + samlet_nøgle_længde (bytes)).

Ved en eventuel tilbageskrivning af posten spares transporten af bloktabellen, og transporten vil være hurtig (13 + 2 * 6.2 msec), hvis pladelageret ikke har været rørt i mellemtiden.

Nedlæggelse af poster og simple indsættelser kræver tilbageskrivning af både bloktabel og blok, medens komplicerede indsættelser, hvor der ikke forud er plads i den rette blok, kan kræve en tid, som er flere gange så stor.

A.2.2 Læsning af listefil-post.

Listefil som i eks. A.2.3.

Ud fra postens nummer beregnes den direkte adresse på den blok, som indeholder posten.

tidsforbrug =
bloktransport =
100 + 2 * 6.2 msec = 112 msec

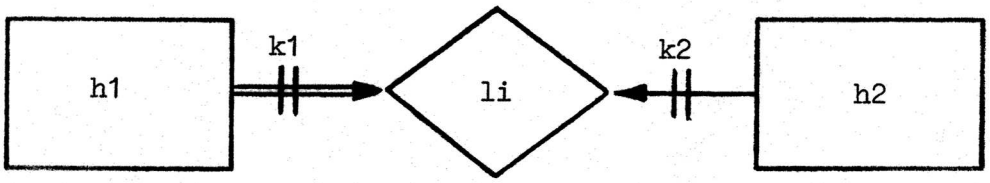
Hvis pladelageret har været anvendt til en anden transport efter læsningen, koster tilbageskrivning af posten det samme, ellers kun 13 + 2 * 6.2 msec = 25 msec. Nedlæggelse eller indsættelse af en post kræver adgang til og opdatering af foregående post, som eventuelt kan ligge i en anden blok (eller i moder-filen).

Denne blok vil normalt stadig være tilstede i lageret, hvis kørslen foregår med en zone med plads til to blokke.

Som nævnt i afsnit 4. bliver en post, som indgår i flere kæder, nedlagt i flere tempi, efterhånden som de kæder, hvortil den er forbundet, bliver gennemløbet. Dette medfører, at man i programmer, som blot gennemløber kæder, vil have fordel af to blokke i lageret, hvis der forekommer mange døde. Besparelsen for hver død post vil være op til én bloktransport i gennemsnit.

Det kan dog også være en ulempe at have flere blokke i lageret, fordi en opdateret blok ikke bliver skrevet tilbage, før der er læst andre blokke ned i de øvrige blokbuffere, hvorved pladelageret sandsynligvis har skiftet cylinder.

A.2.3 Forbindelses-struktur.



Filer som i A.1.1 og A.1.3

Der læses en vilkårlig post i h1, derefter gennemløbes den tilhørende k1-kæde, som tænkes at indeholde 10 li-poster. For hver li-post læses den til k2-kæden hørende moder-post i h2.

tidsforbrug =
læsning af h1-post + 10 * læsning af li-post + 10 * læsning af h2-post =
132 + (1 til 10) * 112 + 10 * 132 msec =
1.6 sec til 2.6 sec

Her gælder det mindste tal i det tilfælde, at k1-kæden er grupperet fuldkomment, d.v.s. at alle 10 li-poster er anbragt i den samme blok. I praksis må man regne med en dårligere gruppering, især hvis filen har været meget fuld, og posterne i kæden ikke er blevet oprettet samtidig.

Ved den tilsvarende proces i den modsatte retning (fra h2) vil tidsforbruget sandsynligvis være det maksimale, idet k2 ikke er grupperet.
(gruppering er kun mulig for én gruppe af kæder).

A.3 Baggrundslager-forbrug.

Pladelagerets størrelse og antallet af pladelagre ved den enkelte installation sætter en grænse for hvor stort system, det er muligt at oprette.

Grænsen for den enkelte fil sættes af pladelagerstørrelsen (8000 segmenter = 4.096.000 bytes for pladelageret RC 433), men det er tilladt at anvende flere pladelagre til en filkonfiguration, og det er i princippet bedst kun af have én fil på hvert pladelager, idet man herved nedsætter samlet antal og varighed af cylinderskift.

Forbruget af baggrundslager kan for den enkelte fil deles i fire dele:

1. aktive poster.
2. døde poster
3. filadministration.
4. uudnyttede blokrester og ledig plads til indsættelser.

Heraf må i hvert fald 2. og 4. betragtes som spild i forhold til lagring på magnetbånd.
Noget af dette spild indvindes formentlig igen, fordi CF-SYSTEM eliminerer redundans, d.v.s. gentagelse af den samme oplysning i flere filer.

A.3.1 Aktive poster.

Posterne indeholder dels bruger-data, herunder nøgle- og længde-felter, og dels kædefelter.

En hovedfil-post indeholder kun moder-kædefelter (se afsnit 4.), ét for hver pil, der udgår fra filen i den grafiske fremstilling af filkonfigurationen. Hvert moder-kædefelt fylder 2 bytes. En listefil-post har tilsvarende ét datter-kædefelt for hver indgående pil i den grafiske fremstilling, men kan også have moder-kædefelter.

Et datter-kædefelt indeholder ud over 2 bytes med henvisning til næste post i kæden evt. også et felt, som refererer til moderposten.
Med en listefil som moderfil er dette felt også 2 bytes, med en hovedfil er det hovedfil-postens samlede nøglelængde, som fås ved at lægge hovedfil-postens nøglefelter i forlængelse af hinanden.

A.3.2 Døde poster.

Disse poster kan kun opstå i listefiler, og skyldes den anvendte kædeteknik, énvejs kædning (se afsnit 4).

En post kan kun nedlægges under gennemløb af én af de kæder, hvori posten er ophængt. For denne kædes vedkommende kender CF-SYSTEM den foregående post og kan lede dennes henvisning uden om den nedlagte. Hvis posten er tilknyttet flere kæder end denne ene, bliver posten i første omgang dødsmærket, og først endeligt fjernet, når den sidste tilsluttede kæde bliver gennemløbet.

Man bør ved planlægningen af en filkonfiguration undgå kæder, som man ingen direkte anvendelse har for (d.v.s. anvendelser, hvor kæden gennemløbes), idet man risikerer, at døde poster bliver ophobet i sådanne kæder.

A.3.3 Fil-administration.

Denne består for begge fil-typer dels af et filhoved, som bl.a. indeholder kæde- og nøgle-specifikationer, dels af en overordnet tabel, der beskriver hele filen, og endeligt af tabeller fordelt i hele filen på bucket- eller blok-niveau.

Disse data udgør skønsmæssigt 5 til 10 % af baggrundslager-forbruget.

A.3.4 Uudnyttede blokrester og ledig plads til indsættelser.

Da poster ikke kan overskride de faste blokgrænser, må i gennemsnit en plads svarende til en halv postlængde i hver blok være umulig at udnytte.

spild = 100/(2*blokningsfaktor) %,

hvor blokningsfaktoren er det gennemsnitlige antal poster, der er plads til i en blok. f.eks. giver 5 poster pr. blok et spild på 10 %.

Ud over dette bør der afsættes ekstra plads til at smidiggøre indsættelse af nye poster.