DASK - en 1. generations computer

af

H. B. Hansen

3. Del

Kørsel med Dask

Dask blev udbygget en hel del i sin levetid, og til slut var der mulighed for at programmere i Algol, og operativsystemet var ret avanceret. Jeg vil dog indskrænke mig til de første to-tre år af Dasks levetid, hvor forholdene var så primitive at det forekommer utroligt at man kunne lave nyttige programmer; men det var faktisk muligt.

Man kom meget tæt på maskinen når man kodede, og kendskab til kontrolbordets indretning var en absolut forudsætning for af få udført et program. Ved hjælp af trykknapperne kunne man sætte bestemte bits ind i registrene. Desuden skulle nøgler og omskiftere på kontrolbordet stilles i bestemte tilstande inden starten af et program. Fx kunne man starte et program på følgende måde:

1. Strimlen med programteksten blev sat i strimmellæseren.
2. Indholdet af AS blev sat til sedecimalt 7C3. Dette er 1987 decimalt, hvilket var indhopsadressen til operativsystemets indlæseprogram. Mere om det senere.
3. Indholdet af OP blev sat til 10 sedecimalt. Dette er operationskoden for et udetinget hop. Man kunne også blot aktivere nøglen HOP.
4. Nøglen START blev aktiveret. Nu blev hopordren i ASOP udført, og programteksten indlæst til ferritlageret, styret af operativsystemkommandoer der var hullet med på strimlen. Specielt måtte strimlen slutte med en systemkommando der stoppede indlæsningen.
5. Strimlen med data til programmet blev sat i strimmelæseren.
6. Indholdet af AS blev sat til den sedecimale værdi af startadressen for det indlæste program. Denne adresse måtte man selv bestemme sig til, så en del af kodningen bestod i at lægge programdele ud i lageret så de ikke overlappede hinanden. Der var med andre ord ingen loader i moderne forstand.
7. Indholdet af OP blev sat til 10 (sedecimalt).
8. Omskifteren for valg af ydre enhed for trykning blev sat til den ønskede værdi (S for skrivemaskine, P for perforator og S+P for begge dele).
9. START-nøglen blev aktiveret . Programmet startede, strimlen i læseren blev forhåbentlig ædt lidt efter lidt, og ud af perforatoren kom der en ny strimmel, som senere måtte bringes hen til flexowriteren for at blive udskrevet.

Hele biblioteket, flydende regning, kvadratrod, trigonometriske funktioner osv. var *ikke* indbygget i maskinen, men forelå også på strimler, der skulle læses ind. De originale strimler var fremstillet af en særlig kraftig papirtype, som kunne tåle at blive kopieret i flexowriteren igen og igen. Så man kopierede altså de biblioteksprogrammer, man havde brug for, med ind på sin programstrimmel, eller læste dem ind i ferritlageret en ad gangen. Meget af arbejdet foregik derfor off-line.

I begyndelsen, dvs. i 1950'erne, var det småt med administrativt personale på Regnecentralen. Der var en rengøringskone, men det var alt. De administrative rutinejobs gik derfor på omgang, hvilket ikke altid var lige hensigtsmæssigt. Jeg husker endnu Jørn Jensen (ham der senere opfandt Jensen's device i Algol, og som udviklede de tidligere omtalte kontrolprogrammer) i rollen som telefonvagt, bøjet over et stykke kodepapir, og med en sammenklappet leverpostejmad i hånden, mens omstillingsbordet ringede, summede og blinkede på livet løs. Hen ad vejen kom der dog mere personale, specielt nogle *hulledamer*,

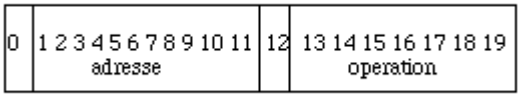
hvis opgave var at fremstille strimler efter vores manuskripter, samt at udskrive strimler på flexowriter. For at det kunne ske nogenlunde fejlfrit var det nødvendigt med lidt mere orden end kodere normalt kunne forventes at præstere, og derfor blev der fremstillet noget særligt kodepapir, som man skrev programmerne på. Arbejdsgangen, som jeg husker den fra slutningen af 50'erne, blev da noget i retning af følgende:

- **Opgavefordeling**. Denne foregik til at begynde med på et ugentligt møde, hvor alle medarbejdere samledes. Bech læste de med posten modtagne henvendelser højt, og hvis man følte sig tiltrukket af en opgave, sagde man strakt til, hvorefter man fik overdraget sagen. Det var næsten som ved en skønhedskonkurrence, hvor man skal afgive sin stemme før man har set alle deltagerne - man falder for en af damerne undervejs, men ægrer sig måske bagefter. Systemet virkede fortræffeligt og sikrede, at man fik prøvet lidt af hvert (det var Bech der bestemte præsentationsrækkefølgen). Senere skete opgavefordelingen dog efter et mere konventionelt system, hvor man fik udstukket opgaver efter ledelsens skøn.
- **Kodning**. Det var den aktivitet der førte frem til at der stod nogle ordrer på nogle kodeark. Der blev ikke skelnet mellem forskellige faser i arbejdet, såsom systemanalyse og systemkonstruktion, alt kørte sammen i én pærevælling. Man måtte selv sørge for at kontakte kunden, og alle opgaver var i princippet enmandsopgaver.
- **Hulning**. Kodemanuskriptet blev herefter hullet, af en hulledame eller af én selv.
- **Testkørsel**. Man indfandt sig i maskinrummet med alle sine strimler, og hvis der ikke var andre ved maskinen, gik man igang med at indlæse strimlerne og køre programmet. Det var naturligvis praktisk taget altid forkert. Man var heldig hvis der overhovedet kom output; enten stoppede maskinen efter få sekunders forløb, eller også gik den i en uendelig løkke, hvilket viste sig ved at lamperne på kontrolbordet blinkede i et eller andet interessant cyklisk mønster. Hvis man skruede op for højttaleren kunne man måske høre en monoton melodi. Så gjalt det om at finde fejlene - i hvert fald bare den første. Det gjorde man ved at køre på trin ved hjælp af speederen på kontrolbordet. Efter hver ordre kontrollerede man at alle registre havde det indhold, man forventede. Vanskeligheden var naturligvis at man kun kunne se registrene binært, men det blev man hurtigt skrap til. Der var også en anden mulighed, nemlig at udskrive hele lagerets og/eller registrenes indhold efter udførelse af en mistænkelig ordre. En sådan post mortem dump var sedecimal (ved en festlig lejlighed foreslog jeg at et post mortem program skulle hedde et ligsynsprogram på dansk, men det fangede aldrig rigtigt an) .
- **Fejlrettelse**. Når man havde fundet en fejl måtte man fremstille en ny strimmel hvor fejlen var rettet. Det kunne måske gøres ved at rette på selve den gamle strimmel, og til det formål havde man særlige små hulleapparater, *håndhullere*, som kunne lave ekstra huller på et udvalgt sted af en lang strimmel. Værre var det hvis der var for mange huller; så måtte man klæbe lidt blank strimmel over de gale hulkombinationer og bruge håndhulleren til at lave de rigtige. Der var sågar nogle kodere der klippede strimler over og spejsede nye stumper ind ved hjælp af tape. Der var også andre muligheder, såsom at lave en rettelsesstrimmel der så skulle indlæses efter den gale strimmel ved næste testkørsel. Denne teknik førte imidlertid let til at man fik en masse små strimler at holde rede på, en for hver rettelse, og de skulle som regel indlæses i en ganske bestemt rækkefølge. Det var skrappe krav at stille til et menneskes noget brøstfældige akkuratesse, navnlig hvis man var under tidspres (hvilket man altid var), og det gik da også tit galt. Hvor ofte har jeg ikke set kodere storme frem og tilbage mellem maskinen og flexowriteren, men et dusin strimler flyvende om halsen, mere og mere opkogte i hovedet, når deadline nærmede sig.
- **Rutinekørsel**. En rutinekørsel med et fejlfrit program var enhver koders drøm. Man læser sin strimmel ind, trykker på START, venter, river strimlen i perforatoren af, og overgiver den til en hulledame - skønt! Der blev altid fremstillet én lang strimmel med et færdigt program, så denne fase kunne faktisk overlades til en *operatør*. Sådan en blev ansat da beholdningen af færdige programmer voksede ud over en vis størrelse.

Denne kørselsform hedder *open shop*. Den kan praktiseres hvis belastningen på maskinen ikke er for stor. Efter kort tids forløb skete der imidlertid det at der næsten altid stod en og kørte ved maskinen, når man kom med sine strimler. Så satte man sig ud i køkkenet og fik en kop kaffe. Det var ikke særlig rationelt, så det blev hurtigt nødvendigt at styre adgangen til maskinen. Det skete ved at man reserverede tid på samme måde som man reserverer tid på en tennisbane, ved hjælp af afkrydsning af et tidsrum på en fælles kalender. Det er første skridt på vejen mod *closed shop*. Næste skridt skete da der kom magnetbåndsstationer på Dask. De var ømfindtlige overfor støvpartikler o.lign., så en skønne dag var der sat et skilt op på døren ind til dagligstuen: Rygning forbudt. Dette forbud bevirkede at operatøren fik mere at lave; mange kunne ikke forestille sig hvordan man skulle kunne køre uden at ryge. Så, i 1959, udkom "Dask Kontrolprogrammer", en ret enestående debugger kodet af Jørn Jensen, og det gjorde det muligt ved hjælp af særlige indkøringsblanketter at overlade også testkørslerne til operatøren. Og en dag var ordet "Rygning" på skiltet til dagligstuen erstattet af ordet "Adgang". Så havde man *closed shop*. Man sender sit programmanuskript og sine data til hulning og korrekturlæsning, udfærdiger en testblanket og sender det hele til kørsel. Nogen tid efter får man resultatet tilbage, hvorefter man laver rettelser og en ny testblanket - og så fremdeles. Koderen kommer aldrig selv i kontakt med maskinen. Det er ikke nær så sjovt som open shop, men man kan jo sige at vi selv havde programmeret os frem til denne sørgelige tingenes tilstand. Det skal dog siges at closed shop aldrig blev realiseret 100% på Dask, det skete først da Regnecentralen fik en stor amerikansk 2. generations maskine (CDC1604).

Ordrekoden for Dask

Formatet af en ordre så således ud i Dask:



Positionerne i en celle i Dask var nummereret fra venstre mod højre. En Dask-ordre fyldte en hac, altså 20 positioner. Den bestod af tre dele: (1) adressen, 11 bit, position 1-11, (2) operationen, 7 bit, position 13-19 og (3) indeksmærkningen, 2 bit, position 0 og 12.

De 11 adressebits var netop tilstrækkelige til at adressere de 2048 hac i ferritlageret, og Dask havde altså ingen basisregistre eller andre nymodens opfindelser til effektivisering af adresseringen. Indicering var faktisk en ny opfindelse; fx havde Besk ingen indeksregistre. De 2 indeksbit gav mulighed for fire indeksregistre: IRA, IRB, IRC og IRD. IRA var at fiktivt register der altid indeholdt nul, så det betød "ingen indicering". Indicering foregik ved at indholdet af det nævnte indeksregister blev lagt til adressen før operationens udførelse. De 7 operationsbits gav mulighed for 128 forskellige operationer. Som noget ganske nyt på den tid var disse operationer ordnet i et bestemt logisk system, så man ikke skulle huske dem alle sammen udenad.

Operationerne blev nummereret sedecimalt, med to sedecimale cifre. Operationerne med numrene 00 til 1F var *grundoperationerne*. Til disse kunne man lægge 20 og/eller 40 (ligeledes sedecimalt) hvorved der fremkom nogle varianter af grundoperationerne. Princippet kan lettest forklares med et eksempel:

Operationskoden 01 betød subtraktion. En ordre med adresse 236, indeksmærke A og operationskode 00 betød: træk indholdet af hec 236 fra indholdet af AR, og gem resultatet i AR. Hvis man lægger 20 til operationskoden fås en ny operationskode: 21, der betød at adressen 236 nu skulle opfattes som adressen på en hac, og virkningen blev så at indholdet af hac 236 blev trukket fra venste halvdel af AR (bemærk at

det er venstre halvdel - Dask regnede med ægte brøker. Hvis adressen havde være ulige var der også blevet opereret på venstre halvdel af AR). Addition af 40 til operationskoden betød "nulstil AR inden operationen", så operationen 41 var "hent hec 236 med modsat fortegn". Endelig kunne man lægge både 20 og 40 til grundoperationen, hvorved man fik kombineret de to varianter; 61 betyder derfor "hent hac 236 med modsat fortegn".

Sådan virkede variationstype 1. Det havde jo været let hvis der ikke var flere variationstyper, men så let var det desværre ikke. Der var ialt 14 variationstyper, men det er dog lettere at huske 32 grundoperationer og 14 variationstyper end 128 operationer.

En ordre blev noteret med en decimal adresse, et indeksmærke (A, B, C eller D) og en sedecimal operation. Ordren

236 A 01

var således den grundordre der er omtalt i eksemplet ovenfor. Denne notationsform ligger tæt op ad ordrenes interne repræsentation, og er efter moderne forhold yderst simpel, men den var dog mere avanceret end det fx var tilfældet for Besk, hvor en ordre skulle angives sedecimalt helt igennem, mens man altså på Dask kunne angive adresserne decimalt.

Operationsliste for Dask

Tabellen viser hvilke maskinoperationer der var i Dask. Der er én række i tabellen for hver grundoperation. Første søjle indeholder grundoperationernes sedecimale værdier; varianttyperne fremgår af sidste søjle i tabellen. Et udvalg af de vigtigste varianttyper er beskrevet efter selve tabellen. De to midterste søjler i tabellen giver hhv. en kort betegnelse for hver operation og en beskrivelse af operationens virkning i en Pascallignende notation. I søjlen med operationernes virkning er der benyttet følgende betegnelser:

- AR: akkumulatorregisteret. Et indeks på AR udpeger en bestemt position i AR; fx er AR_0 fortegnspositionen. Et interval af indices udpeger et felt i AR; fx betyder $AR_{1\text{-}11}$ adressepositionerne i AR's venstre halvdel, mens $AR_{21\text{-}31}$ er adressepositionerne i AR's højre halvdel.
- MR: multiplikatorregisteret. Indicering er analog til indicering for AR.
- KR: Kontrolregisteret
- m: den resulterende adresse i ordren, dvs. indholdet af AS-registeret plus indholdet af det indeksregister, der er angivet i ordren.
- memory[m]: en celle i ferritlageret. Ferritlageret opfattes som et array memory[0..2047]. Indicering er analog til indicering for AR.

Operation	Navn	Virkning	Type
00	Addér	AR := AR + memory[m]	1
01	Subtrahér	AR := AR - memory[m]	1
02	Addér numerisk	AR := AR + |memory[m]|	1
03	Subtrahér numerisk	AR := AR - |memory[m]|	1
04	Addér og læs til MR	AR := MR := AR + memory[m]	1

05	Subtrahér og læs til MR	AR := MR := AR - memory[m]	1
06	Addér og læs til celle	AR := memory[m] := AR + memory[m]	2
07	MR til AR	MR := AR	3
08	AR til celle	memory[m] := AR	1
09	$AR_{adresse}$ til celle	$memory_{0-11}$ [m] := AR_{0-11}	1
0A	Multiplicér	AR := MR*memory[m]	4
0B	Dividér	MR := AR/memory[m]; AR := rest	4
0C	Venstreskift	skift AR m positioner til venstre	5
0D	Aritmetisk højreskift	skift AR m positioner til h¯jre, AR_0 uÊndret	5
0E	Normalisér	skift AR til venstre indtil AR_0 <> AR_1	5
0F	Logisk højreskift	skift AR m positioner til h¯jre, AR_0 = 0	5
10	Hop	Næste ordre fås i celle m	6
11	Hop på fortegn	Næste ordre fås evt. i m, betinget af AR_0	7
12	Hop på spild eller ikke-spild	Næste ordre fås evt. i m, betinget af spildsituation	8
13	Hop på indeks	Næste ordre fås evt. i m, betinget af IR	9
14	IR til celle	$memory_{1-11}$ [m] := IR	9
15	Adresse til IR	IR := m	9
16	Indekshop	Næste ordre fås i m, IRD := KR	10
17	Hop til syet lager	Næste ordre fås i m i det syede lager,IRD := KR	10
18	Indført d. 18. juli 1960, i stedet for AR := tilfældigt tal.	$memory_{0-31}$ [m] := AR_{0-31}; $memory_{32-39}$[m] := 0	
19	Læs fra strimmel	Læs 10 sedecimale cifre til AR og memory[m]	11
1A	Tryk ciffer	Tryk AR_{36-39}	12
1B	Tryk tegn	Tryk tegn afh. af m	12
1C	Vælg ydre enhed	Ydre enhed nr. m kobles til CPU	13
1D	Læs fra valgt ydre enhed	En blok læses til memory[m] og frem	14
1E	Søg blok på valgt bånd	AR := AR + memory[m]; Blok nr. AR_{0-19} kobles til CPU	1
1F	Skriv på valgt ydre enhed	En blok skrives fra memory[m] og frem	14

Nogle udvalgte variationstyper

Type 1: Grundformen opererer på helceller. Grundform+20 opererer på halvceller. Grundform+40 opererer på helceller, men nulstiller AR inden operationen. Grundform+60 opererer på halvceller, men nulstiller AR inden operationen.

Type 2: Grundformen og grundform+20 virker som beskrevet i tabellen, på hhv. helceller og halvceller. Grundform+40 bevirker at adressedelene i memory[m], dvs. position 0-11 og 20-31, bliver forøget med 2.

Grundform+60 virker analogt med grundform+40, men kun på en halvcelle (pos. 0-11 eller pos. 20-31, afhængigt af om m er lige eller ulige).

Type 3: operation 07 og 27 virker begge som beskrevet i tabellen. 47 og 67 har begge følgende virkning: AR := AR and MR, idet konjunktionen udføres på alle 40 bits i parallel.

Type 4: Grundform og grundform+20 virker på den korte akkumulator, hhv. helcelle og halvcelle. Grundform+40 og grundform+60 virker på den lange akkumulator: ARMR, hhv. helcelle og halvcelle.

Type 5: Samme som type 4, men da der ikke er nogen lageracces skelnes ikke mellem helceller og halvceller.

Type 6, 7, 8 og 10: Grundform og grundform+40 giver ikke stop før hop, mens grundform+20 og grundform+60 bevirker stop før hop. 10 og 30 nulstiller ikke AR før hop, mens 50 og 70 nulstiller AR før hop. 11 og 31 er hop på plus, mens 51 og 71 er hop på minus. 12 og 32 er hop på spild, mens 52 og 72 er hop på ikke-spild. 16 og 36 er ubetingede hop, mens 56 og 76 er hop betinget af omskifter på kontrolbordet.

Type 9: Grundformen er en blindordre (IRA). De tre varianter vedrører hhv. IRB, IRC og IRD.

Type 12: for 1A og 3A ordrens vedkommende trykkes AR_{36-39} på det valgte outputmedie, og for 5A og 7A ordrens vedkommende er det AR_{33-39} der trykkes (altså et valg mellem et 16-tegns alfabet og et 128-tegns alfabet). For 1B og varianter er det adressen modulo 16 hhv. 128 der trykkes.

Dette er et ordrerepertoire man genfinder i forskellige skikkelser i mange maskiner, og jeg vil senere vise eksempler på ordrernes anvendelse i små programmer. Her blot et par bemærkninger om nogle af ordrerne.

Grundoperationen 07 har, som alle grundoperationer, tre varianter, men det er jo svært at variere en flytning mellem to registre på fire måder. Derfor udnyttes kun to af mulighederne (i variationstype 3). Den anden af mulighederne er den eneste logiske dyadiske operation i Dask, nemlig logisk konjunktion af AR og MR (alle 40 bits i parrallel). Man kunne synes at det var lidt småt, men det er fordi man tænkte på logisk konjunktion som en ren maskeoperation der skulle anvendes i forbindelse med logiske skift, grundoperation 0C eller 0F. Man kan forøvrigt selv programmere logisk disjunktion, nemlig som *aritmetisk* addition minus *logisk* konjunktion. Dette trick viser noget, der er karakteristisk for den tids programmering: der var ingen typer; alt var blot bits i celler, og dem kunne man operere på efter forgodtbefindende.

Grundoperationen 16 var hop til et underprogram. Indholdet af KR, når ordren udføres, er den adresse ordren kommer fra, og dette indhold overføres til IRD, dvs. returadressen står i IRD. Et returhop bliver derfor nemt at programmere:

1 D 10

Til gengæld er det jo svært at kode rekursivitet, men det havde man slet ikke tænkt på i 1957. Varianten 56 betød "returhop afængigt af omskifter på kontrolbordet". Hvis omskifteren var slået fra virkede denne ordre som en blindordre. Dette kunne som tidligere nævnt bruges til testformål.

De to grundoperationer 17 og 18 blev ikke brugt til at begynde med, men blev senere udnyttet. På et tidspunkt kom der en generator for tilfældige tal på Dask, og ordren 0 A 18 bevirkede da at der kom 40 tilfældige bits i AR. Meget smart, men det viste sig dog hurtigt at anvendelse af en generator for tilfældige tal havde visse ulemper. Generatoren udnyttede variationer i den kosmiske baggrundsstråling til at generere bittene, hvilket vel må siges at være ret tilfældigt, i hvert fald uden for menneskelig styring. Men det betød at en testkørsel ikke kunne reproduceres efter rettelse af en fejl, og det blev vanskeligt at overbevise sig om at fejlen faktisk var rettet. Derfor blev generatoren senere koblet fra igen, og man

klarede sig med pseudotilfældige tal, som det også er almindeligt nu om dage. Generatoren for tilfældige tal blev dog ikke skrottet, men blev til en ydre enhed i lighed med fx mangnetbånd.

17- ordren blev til det tidligere omtalte returhop ind i det syede lager.