Datapoint

Rudk ? rådgivning

Niels-BO - forsikringsselskabs-opgave (Fredericia?) for RC

hjælpe "pigerne" i de resterende måneder?

Leasing-selskab. , Vekseler , RC-annonce for os.
udlånelse til landmand

tekstbehandling.

Skov regnskab.

Application - areas :

1. Intelligent Data Entry
2. Multi - Processing System (DATASHARE)
3. Business Languages
4. Communications Software

---

ad. 1. Intelligent Data Entry

DATAFORM or DATABUS

DATABUS ← offers more processing capability

DATAFORM ← contains a fast, simple-to-use form generation and powerful input edit facilities

OPERATING SYSTEMS :	Diskette Operating System
	Cassette " "
PROGRAMMING LANGUAGES :	DATAFORM
	DATABUS
COMMUNICATIONS :	DATAPOLL
	Emulators

---

ad. 2. Multi - Processing Business Systems - DATASHARE

OPERATING SYSTEMS :	Disk Operating System
	DOS Utilities
PROGRAMMING LANGUAGES :	DATASHARE
COMMUNICATIONS :	DATAPOLL
	Emulators

ad. 3. General Purpose Business Computing

Many Datapoint computers are used as local, in-house, production computing facilities.

OPERATING SYSTEMS: ALL

PROGRAMMING LANGUAGES: DATABUS
(RPG II, BASIC, SCRIBE)

---

ad. 4 Data communications

COMMUNCATIONS: DATAPOLL
Emulators

UTILITIES PACKAGES	Debugging tools
	Math subroutines
	Data transfer / print
	Recovery and backup
	Sort / Index / Reformat
	Program generation

OPERATING SYSTEMS

With Datapoint computers an operating system forms the foundation of a program generation system and also becomes the master program under which the final applications program will operate.

In many cases, application programs will use sections of the operating system for subroutines.

Datapoint offers five operating systems - one for each major storage media.

DISK OPERATING SYSTEMS – are symbolically oriented and designed to completely remove any requirement that the programmer or operator know any disk file parameters about any data file or program other than its name.

File location, expansion, contraction and random/sequential and index/sequential (ISAM) access facilities are automatically provided by the Datapoint DOS's.

Editors, assemblers and program generation packages are handled by a DOS.

DOS require a 16 K machine.

Cartridge Disk Operating System	DOS.A
Mass Storage Disk –"– –"–	DOS.B
Diskette –"– –"– (media: Cassette)	DOS.C
Diskette 1100 Operating System (media: Diskette)	DOS.C

DOS Utilities:

BACKUP = Disk Copy and De-Fragmentation Program

CHAIN = Program Chaining Command

BLOKEDIT = Block Text Editor

DOSLIST = Text File Lister

CRCFIX = Cyclic Redundancy Check Fixer

DDUMP = Visual Dump Program

SOBO = Source and Object Cassette Output Command

BOOTMAKE = Disk Bootstrap Generator

INIT 9370 = Mass Storage Disk Initializer

FILES = Sorted File Description List Command

SYSLEEP = DOS Pack Sleep

MIN / MOUT = Cassette Input / Output Utilities

SORT = Disk-based Sorting Facility

User specify:

1. The name of the data file to be located and sorted.
2. The character positions (i.e., the location or key) of the sort field and, if desired, the positions of additional fields to be subsequently recordered.
3. The name of the resulting output data file.

In addition to the above, the following optional specifications can be designated:

1. Output format as either indexed or sequential
2. Ascending or descending collating sequences
3. Required portions of the records to be placed in the new file.
4. Hardcopy output of the new file.
5. Sorting of primary and secondary records
6. Conditional parts
7. Insertion of constant data
8. Tag files, i.e., one with pointers or tags to designate several sequences of the same file.

TAPE = Magnetic Tape Utility Routine

REPAIR = DOS Pack Repair Program

INDEXED SEQUENTIAL FILE ROUTINES (ISAM)

REFORMAT = ISAM Index File Reformatter

INDEX = ISAM Index File Generation

CASSETTE TAPE OPERATING SYSTEM

CTOS = Cassette Tape Operating System

IN - input a program file and assign a name to it.

DELETE - delete the specified file from the catalog and from the CTOS tape.

CHOP -

SYMBOLIC - input a source file and assign a name to it

REPLACE - replace the specified file with a new object file

INSERT - place a new object file on the CTOS tape in front of the cataloged file.

OUT - copy the specified file from the CTOS tape onto another cassette in object file format.

LGO - write a loader followed by the specified file(s) from the CTOS tape

APPEND - append an object file to the cataloged file.

ATTACH - attach the source file to the object file in front deck.

ATO - " " object " " " " " " " "

AUTO - to set [illegible]

MANUAL - clears the auto-load entry.

RUN - load and execute specified file.

SREPLACE - replace the specified file with a new source file

SINSERT - place a new source file on the CTOS tape in front of the cataloged file.

In addition, there are tape manipulation commands: REWIND, BACKSPACE, FAD, (file advance), and RAD (record advance). The LIST command will read and display any tape on the video screen.

CTOSPGS = Assembly Program Generation System

CTOS UTILITIES:

LISTER = Cassette General Purpose Lister

DUMP = Memory to Cassette Tape Dump

DEBUG = Boot Block Resident Debugging Routine

CORDMP

FIX

RCOPY = Expanded Cassette Tape Duplicating Program

COPY " " " "

MAGNETIC TAPE OPERATING SYSTEM

MTOSGEN

IN

DELETE

CHOP

PROGRAMMING LANGUAGES

DATABUS, DATASHARE and RPG II are most effective in a business data processing environment as their structure easily accommodates files, records and text-oriented data.

BASIC, however, is strongly oriented toward formula solving and complex calculations.

DATAPOLL is a high-level communications package.

SCRIBE is a text-processing language to an already impressive list.

DATApoint BUSiness language

DATABUS is a family of high level programming languages designed especially for any Datapoint processor and its peripherals.

DATABUS 1, 2, 3, 4, 5, 6, 7

DATABUS for Diskette 1100

DATAFORM Forms oriented, straight forward data entry applications are best handled by a data entry language with the same characteristics

Level I is the forms generation facility of DATAFORM. Forms may be generated and utilized for data transcription to cassette or diskette media with absolutely no programming required.

Level II provides an extension of the error checking capability of DATAFORM

DATAFORM 2 is the cassette based version of DATAFORM

DATAFORM 1100 is diskette based. It operates under DOS.C

Disk Based DATAFORM. It's generally used with the cartridge disk.

DATASHARE - permits the simultaneous execution of independent DATASHARE-programs, each dealing with its own remote Datapoint CRT terminal.

INSTRUCTIONS

Directive Instructions

FORM

DIM

INIT

Control Instructions

GOTO

CALL

RETURN

STOP

CHAIN

TRAP

TRAPCLR

BRANCH

ACALL

Character String Handling Instructions

CMATCH

CMOVE

MATCH

MOVE

APPEND

RESET

BUMP

TYPE

EXTEND

CLEAR

LOAD

STORE

Numeric String Variable Arithmetic Instructions

ADD

SUB

MULT

DIV

MOVE

COMPARE

LOAD

STORE

Keyboard, C.R.T., Printer Input/Output Instructions

KEYIN

DISPLAY

PRINT

BEEP

CLICK

DSENSE

KSENSE

Directive Instructions

Ernst Abildtrup (12/3)

Variable Definition

Numeric String Variables are defined with the FORM instruction

```
EMRATE   FORM 4.2          (9999.99 to -999.99)
```

max 22 chas.

startværdi er 0

```
XAMT     FORM " 382.4  "   (9999.999 to -999.999)
```

startværdi er 382.400

Character String Variables.

Character strings are defined with either the dimension instruktion DIM or the initialization instruction INIT

```
ANAM     DIM 24        (reserverer plads til 24 tegn)
```

max 127 chas.

startværdi = 24 spaces
formpointer = 0
logical length = 0
physical length = 24

```
TITLE    INIT "PAYROLL PROGRAM"
```

startværdi = PAYROLL PROGRAM
formpointer = 1
logical length = physical length = 15

note: The actual amount of physical space reserved is three bytes greater than the number specified in the DIM or quoted in the INIT instruction

- TITLE occupies 18 bytes in memory, 15 of which hold characters.

Control Instructions

GOTO

```
GOTO  CALC
GOTO  CALC  IF OVER
GOTO  CALC  IF NOT OVER
```

$$\begin{array}{ll} \uparrow & \uparrow \\ \text{label} & \text{conditions:} \end{array}$$

$$\left.\begin{array}{l} \text{OVER} \\ \text{LESS} \\ \left.\begin{array}{l}\text{EQUAL} \\ \text{ZERO}\end{array}\right\} \text{ens} \\ \text{EOS} \end{array}\right\} \text{The conditions result from previously executed instructions.}$$

CALL

(sammenlign procedurekald. Op til 8 niveauer inden RETURN)

```
CALL FORMAT
CALL XCOMP IF LESS
```

RETURN

```
RETURN
```
(sammenlign <u>end procedure</u>)

```
RETURN IF EQUAL
```

STOP

STOP causes the program to terminate and return to the MASTER Program.

```
STOP
STOP IF OVER
```

CHAIN enables the user to fetch and run another program on the interpreter system tape.

```
NXTPGM   INIT   "020"
                  :
                  :
             CHAIN NXTPGM
```

"020" ← octal file number of the desired program (file 020 = file 16 decimal)

CHAIN NXTPGM causes file 020 on the interpretive tape to be loaded into memory and run.

TRAP does not take action at the time it is executed but specifies that a transfer of control should occur later if a specified event occurs.

```
TRAP   EMSG   IF   EOT2
```

specifies that control should be transferred to EMSG if the end-of-tape - is encountered on cassette deck two (front deck)

The transfer is like the GOTO.

Once the trap location is set, transfer will continue to occur to that location until the trap is reset with another TRAP statement or cleared with the TRAPCLR instruction

The events that may be specified are:

EOT(n)	- End-of-tape mark on indicated device
RFAIL(n)	- Read failure on indicated device (parity error, bad file mark)
FORM(n)	- String data read into numeric field from indicated device
	n = 1 or 2 (1 = cassette deck 1)
CFAIL	- Specified file number of CHAIN instruction not valid.

TRAPCLR

TRAPCLR RFAIL1 clears the trap location set for the read failure

BRANCH transfers control to a statement specified by an index.

```
BRANCH  N  OF  START, CALC, POINT
                 ↑
                n=1     2     3
```

↑ if $N \leqq 0$ or $N >$ the number of variables, control continues with the following statement.

Note that the index is truncated to the no decimal places before it is used (e.g. $1.7 = 1$)

ACALL allows the user to call assembly language subprograms to be executed outside of the interpreter.

The assembly language programs should not overlay any of the interpreter or the Databus user area which calls it, unless the program reloads the interpreter or user program before returning, in which case the user program should be restarted.

ACALL 0200000 calls a subprogram starting at location 20000 octal.

The location to be called may be decimal or octal, but must be a numeric literal.

The last statement in the subprogram executed should be a RET to return to the interpreter to resume execution of the Databus program.

TYPE sets the EQUAL and ZERO condition if the string is of valid numeric format (only leading minus, one decimal point, and digits or spaces).

```
TYPE   ALPHA
```

EXTEND increments the formpointer, stores a space in the position under the new formpointer, and sets the logical length to point where the new formpointer points if the new logical length would not point to the ETX at the end of the character string.

```
EXTEND  BUFF
```

CLEAR causes the operand's logical length and formpointer to be zero.

```
CLEAR NBUFF
```

LOAD performs a MOVE from the character string pointed to by the index numeric operand, the second operand, to the first character string specified

```
LOAD  AVAR  FROM  N  OF  NAME,TITLE,HEDING
```

STORE performs a MOVE from the first character string specified to a character string in a list specified by an index numeric operand given as the second operand.

```
STORE  Y  INTO  NUM  OF  ITEM,ENTRY,ALINK,LIST
```

Character String Handling Instructions

13/3 79

CMATCH compares two characters, one taken from each of the source and destination operands.

```
CMATCH   XDATA TO YDATA
CMATCH   Y,X
CMATCH   "A", DOG
CMATCH   DOG TO "B"
CMATCH   CAT, 0101
```

CMOVE moves a character from the source operand to under the formpointer in the destination string.

```
CMOVE  XDATA, YDATA
CMOVE  "A" TO CAT
CMOVE  X, Y
CMOVE  0101 TO STRING
```

MATCH compares two character strings starting at the formpointer of each and stopping when the end of either string is reached.

If either formpointer is zero, the MATCH operation will result in only clearing the LESS and EQUAL flags and setting the EOS flag.
Otherwise, the "length" of each string is calculated to be LENGTH - FORMPOINTER + 1 and the LESS flag is set if the destination string length is less than that of the source string.
The two strings are then compared on a character-for-character basis for the number of characters equal to the lesser of the two lengths.
If all the characters match, the EQUAL flag is set.
If they do not match, the LESS flag's meaning is changed to indicate whether the numeric value of the destination character (in the character pair) is less than the numeric value of the source character (LESS flag set) or vice versa (LESS flag reset)

Source	Destination	Result
ABCDE	ABCD	EQUAL, LESS
ABC	Z	NOT EQUAL, NOT LESS
ZZZ	AAA	LESS, NOT EQUAL
ABC	ABC	EQUAL, NOT LESS
ABCD	ABCDE	EQUAL, NOT LESS

```
MATCH   A TO B
MATCH   STR1, STR2
```

13/3 79

Numeric String Variable Arithmetic Instructions

Following each numeric string variable arithmetic instruction, the condition flags, OVER, LESS, and ZERO (EQUAL) are set to indicate the results of the operation.

OVER indicates that the result of an operation is too large to fit in the space allocated for the variable (a result is still given with truncation to the left and rounding to the right, however).

LESS indicates that the content of the second variable is negative following the execution of the instruction (or would have been in the case of COMPARE)

ZERO (EQUAL) indicates that the value of the second variable is zero following the execution of the instruction.

Whenever overflow occurs, the higher valued digits that do not fit the variable are lost. For example, a variable is defined:

```
NBR42    FORM  2.2
```

and a result of 4234.67 is generated for that variable, NBR42 will contain only 34.67.

Whenever an operation produces lower order digits than a variable was defined for, the result is rounded up. A variable with the FORM 3.1 would contain:

```
46.2  for  46.213
812.5  for  812.483
3.7  for  3.666
3.9  for  3.850
```

Note that if an OVER occurs during an ADD, SUB, or COMPARE of two strings of different physical lengths, the result and the LESS condition flag may not be correct.

<u>ADD</u> causes the content of variable one to be added to the content of variable two.

```
ADD  X TO  Y
ADD  DOG, CAT
```

<u>SUB</u> causes the content of variable one to be subtracted from the content of variable two.

```
SUB  RX350  FROM  TOTAL
SUB  Z, TOTAL
```

Cassette Tape Input / Output Instructions

READ

WRITE

REWIND

BKSP

PREPARE

WEOF

14/3 79

<u>LOAD</u> selects an operand out of the list based on the index operand. It then performs a MOVE operation from the contents of the selected variable into the first operand.

```
LOAD CAT FROM N OF FACT, MULT, SPACE
```

<u>STORE</u> selects an operand out of the list based on the index operand. It then performs a MOVE operation from the contents of the first operand <u>into the selected variable</u>.

```
STORE X INTO NUM OF VAW, SUB, TOT
```

14/3 79

<u>MULT</u> causes the content of variable two to be multiplied by the content of variable one.

Note that the sum of the number of characters in the two operands must be less than 32.

```
MULT DICK BY HARRY
MULT W,Z
```

<u>DIV</u> causes the content of variable two to be divided by the content of variable one.

Note that the number of characters in the dividend <u>plus</u> the number of characters in the divisor plus two times the number of characters after the decimal point in the divisor must be less than 32.

Division by zero results in the OVER condition being set and the destination variable not being changed.

If the quotient cannot be represented fully in the destination variable format, the quotient will be rounded to the number of places in the destination variable if the divisor has at least one digit place after the decimal point.
If there are no digit places after the decimal point in the divisor, the quotient will be truncated (rounded down) to the number of places in the destination variable.

```
DIV SFACT INTO XRSLT
DIV X3,HOURS
```

<u>MOVE</u> causes the content of variable one to replace the content of variable two

```
MOVE FIRST TO SECOND
MOVE A,B
```

<u>COMPARE</u> sets the condition flags exactly as if SUB instruction had occurred.

```
COMPARE XFRM TO YFRM
COMPARE RING, DING
```

Care should be used in defining variables to be compared.
Comparison of variables in which the length of the first variable

15/3 79

Keyboard, C.R.T., Printer Input / Output Instructions

These statements move data between the program variables and the keyboard, screen, or printer.

KEYIN causes data to be entered into either character or numeric strings from the keyboard.

Other than variable names, the KEYIN instruction may contain quoted items and list controls.
Quoted items are simply displayed as they are shown in the statement.
The list controls begin with an asterisk and allow such functions as cursor positioning and screen erasure.

```
          list control    quoted item
          ⏞⏞⏞⏞⏞⏞⏞⏞⏞⏞    ⏞⏞⏞⏞⏞⏞⏞⏞⏞⏞⏞
KEYIN     *P1:1, *EF, "NAME: ", NAME, *H35, "ACNT NR:  ":

          ACTNR,  " ADDRESS: ", STREET, *P10:Y, CITY:
                                              ↑
          *HX, *V4, "ZIP: ", ZIP;          Variable
            ↑
          variable
```

DISPLAY follows the same rules as the KEYIN <u>except</u> that when a variable name is encountered in the list following the instruction, the <u>variable's contents</u> <u>are displayed</u> <u>instead of keyed in.</u>

```
DISPLAY    *P5:1, "RATE: ", RATE:

           *H5, *V2, "AMOUNT: ", AMNT
```

15/3 79

PRINT causes the contents of variables in the list to be printed in a fashion similar to the way DISPLAY causes the contents of variables to be displayed.

The list controls are much the same as DISPLAY except that cursor positioning cannot be used, column tabulation is provided: *<n> causes tabulation to column <n> unless that column has been passed.

*F causes an advance to the top of the next form.

*L causes a line feed to be printed.

*C causes a carriage return to be printed.

```
PRINT   *20,"TRANSACTION SUMMARY",*C,*L:
        PNAME,*C,*L,*10,RATE,*20,HOURS,*30:
        AMNT,*L
```

BEEP causes the machine to produce an audible tone.

```
BEEP
```

CLICK causes the machine to produce an audible click

```
CLICK
```

DSENSE tests the DISPLAY key sense switch.

If the DISPLAY key has been depressed, then the EQUAL condition flag is set.

If the DISPLAY key is not depressed, then the EQUAL condition flag is reset.

```
DSENSE
```

__MOVE__ transfers the contents of the source string, starting from under the formpointer, into the destination string.

```
MOVE  STRING TO STRING
MOVE  A,B
MOVE  STRING TO NUMBER
MOVE  NUMBER, STRING
```

__APPEND__ appends the source string to the destination string.

```
APPEND  SOURCE TO DEST
APPEND  NAME,BUFF
```

__RESET__ changes the value of the formpointer of the source string to the value indicated by the second operand.

```
RESET  XDATA TO 5
RESET  Y
RESET  Z TO NUMBER
RESET  Z TO STRING
```

__BUMP__ increments or decrements the formpointer if the result will be within the string (between 1 and the logical length).

```
BUMP  CAT
BUMP  CAT BY 2
BUMP  CAT,-1
```

__ENDSET__ causes the operand's formpointer to point where its logical length point.

```
ENDSET  PNAME
```

__LENSET__ causes the operand's logical length to point where its formpointer points.

```
LENSET  QNAME
```