TI-MIX 1983
International Symposium

April 5-8, 1983
New Orleans Hilton Hotel

Session Proceedings

Techniques & Concepts for the Nontechnical or New User

TECHNIQUES AND CONCEPTS FOR THE NONTECHNICAL OR NEW USER

TABLE OF CONTENTS

TI-MIX (Texas Instruments Mini/Microcomputer Instruments Exchange) is an organization for users of TI computers and related equipment. The purpose of TI-MIX is to promote the exchange of information between users and TI. Membership in TI-MIX is open to any person with an interest in TI computers or peripheral equipment. The international symposium provides a vehicle for direct interaction and information exchange with other users and with TI personnel. Acceptance of TI-MIX member papers for presentation at TI-MIX 1983 does not constitute an endorsement by TI-MIX or Texas Instruments Incorporated.

TI-MIX
M/S 2200
P.O. Box 2909
Austin, Texas 78769
(512) 250-7151

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

TI-MIX 1983
Techniques and Concepts for the Non-Technical or New User
Survey of 990 Languages

SPEAKERS

INTRODUCTION	**MOIZE ADNEY**	**LANGUAGES DEVELOPMENT MANAGER**
COMMERCIAL APPLICATION LANGUAGES/UTILITIES COBOL BASIC RPGII SORTMERGE	**RHONDA LACKEY**	**COBOL PROJECT TEAM MEMBER**
SCIENTIFIC APPLICATION LANGUAGES FORTRAN, PASCAL, C	**MITCH POTTER**	**PASCAL PROJECT TEAM MEMBER**
PRODUCTIVITY SOFTWARE DBMS, DD, QUERY, TIFORM, TIPE	**TIM SEIBERT**	**QUERY PROJECT TEAM MEMBER**

SPEAKERS

This is the Languages section of the TIMIX 1983 Techniques and Concepts for the Non Technical or New User.

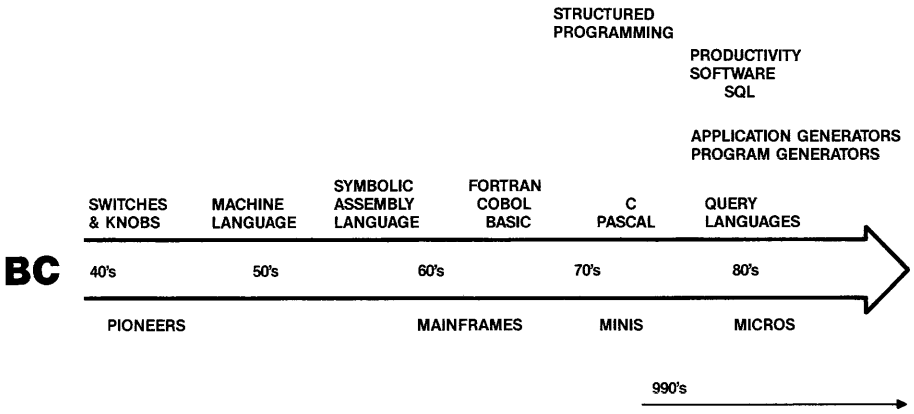
The Languages speakers are listed above along with a description of what they do back in Texas.

Rhonda Lackey is a member of the COBOL development project. Her topic will be Commercial Languages.

Mitch Potter works with Systems Languages and will discuss Pascal and C. Mitch is going to tell us about Fortran too. Fortran is probably the best known scientific application language.

Tim Seibert is a member of the QUERY 990 project team and will talk about productivity software.

CHRONOLOGY



CHRONOLOGY

Before we begin exploring the individual languages, I would like to give you some perspective on where we are today and how we got there.

Although computation, assisted by symbolism and mechanical devices, has been practiced for a long, long time - electronic digital computers have, by comparison, only recently arrived on the scene.

In something just over 40 years computers have evolved from the pioneer class, programmed largely by switches and knobs, to the situation of the 80's where everyone in the industrialized world has some daily interaction with computers large and small. Todays computers are programmed in a variety of languages - most of which are better suited for a particular, narrow set of applications than for others. This matching of the language to the application is a part of the information in the discussions which follow.

BC, by the way, is "Before Computers".

Mr. Randy Hall, our session chairman, specifically asked us to discuss symbolic assembly language and assure you that it has not, like switches and knobs, become extinct....yet.

"Assemblers" as the programs that process assembly language source statements are called, were among the first attempts to get a computer to perform some of the more tedious tasks associated with the intellectual process of writing a computer program.

SYMBOLIC ASSEMBLY LANGUAGE

LINE NUMBER	RELATIVE MEMORY LOCATION	MACHINE LANGUAGE	SOURCE STATEMENT			
0028	207C		L170			
0029	207C	9880		CB	RO,@FLAGS(R2)	IF FLAGS (I)
	207E	0026				
0030	2080	130D		JEQ	L260	=1, THEN 260
0031			*			
0032	2082	C0C2		MOV	R2,R3	PRIME=I+I+3
0033	2084	A0C2		A	R2,R3	*
0034	2086	0223		AI	R3,3	
	2088	0003				
0035			*			
0036	208A	C102		MOV	R2,R4	K=I+PRIME
0037	208C	A103	L210	A	R3,R4	*
0038	208E	0284		CI	R4,8190	IF K > S
	2090	1FFE				
0039	2092	1B03		JH	L250	THEN 250
0040	2094	D900		MOVB	RO,@FLAGS(R4)	ELSE FLAGS(K)=0
	2096	0026				
0041	2098	10F9		JMP	L210	GOTO 210, K=K+PRIME
0042	209A		L250			
0043	209A	0581		INC	R1	COUNT=COUNT+1
0044	209C		L260			
0045	209C	0582		INC	R2	I=I+1
0046	209E	0282		CI	R2,S	IF I <S
	20A0	1FFF				
0047	20A2	1AEC		JL	L170	THEN 170
0048			*			
0049	20A4	0605		DEC	R5	IF LOOP COUNT
0050	20A6	16E1		JNE	L010	<>0 THEN 10
			LABEL	OPERATION CODE	OPERAND(S)	COMMENTS

SYMBOLIC ASSEMBLY LANGUAGE

Here is an excerpt from an assembly language listing of a program that implements one of the popular benchmarks for personal computers.

Note that the assembler has translated the operation codes and operands into the corresponding machine language. Also note how the assembler is allocating storage locations to the machine language instructions and assigning values to the labels in the source code. The comments are supposed to help other programmers understand what the program is doing.

Assembly language offers maximum flexibility in program implementation and arrangement of data. Decisions to minimize program size or optimize execution speed are relatively easy to implement in assembly languages, although the result may not be easy to understand, even with the best of comments.

Fortunatley programming languages that operate at a "higher level" have evolved and are now in widespread use.

Let me now introduce Rhonda Lackey who will survey some of the first high level languages designed for commercial applications.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

COMMERCIAL LANGUAGES

- INTRODUCTION
- TERMINOLOGY
- A BASIS FOR LANGUAGE EVALUATION
- CHARACTERISTICS OF COMMERCIAL APPLICATIONS
- SURVEY OF COMMERCIAL LANGUAGES
  - BASIC
  - COBOL
  - RPG
  - SORT/MERGE

COMMERCIAL LANGUAGES

This presentation will describe a subset of the languages and tools available on the 990. These languages are those that we group as "Commercial Languages" since they are used primarily for business data processing applications.

I'll begin by defining some key terms that will be used.

Before examining specific languages, it will be useful to establish a basis for comparision and evaluation. We will look at how languages differ and what makes one preferable for a specific application.

We'll look quickly at some of the major characteristics that typify commerical applications.

We will examine each of these commercial languages in turn, moving from an overview of the history and general characteristics to focus on features and future directions specific to the TI implementation. A sample program fragment of each will be shown, not for the purpose of detailed analysis, but merely to demonstrate the general "flavor" of the language. Each language will also be individually highlighted on an overall language comparison matrix which reflects some of the key issues addressed in our "language questionaire".

TERMINOLOGY

- **PROGRAMMING LANGUAGE, HIGH-LEVEL LANGUAGE, TOOL**
- **DATA STRUCTURE, OBJECT, VARIABLE, TYPE**
- **ALGORITHM, PROCEDURE, CODE, CONTROL STRUCTURES**
- **COMPILER, INTERPRETER**
- **MAINTENANCE, (TRANS)PORTABILITY**

TERMINOLOGY

A programming language is a formal notation for describing data and the transformations we wish to perform on that data. A high-level language allows these to be described above the level of primitive operations actually available on a particular machine. This may be contrasted with a software tool which may be used either directly or via a programming language to perform some specific function or set of functions.

The descriptions of the representation and relationships between data are referred to as data structures. A "piece" of data is an object (or variable) whose type is that of its underlying structure.

An algorithm is a description of the steps required to transform the data (solve the problem). This description may also be referred to as a procedure or code. Control Structures are facilities provided by the language for governing the order in which solution steps are executed. Most shampoo bottles give an algorithm which uses the control structure "Repeat" (and fails to specify when to stop the cycle).

This notation is translated into machine-executable form by another program called a compiler. A compiler produces an equivalent set of instructions which may be directly executed by the target machine or may require an interpreter to perform the indicated operations. A frequent trade-off between the two is the superior execution speed of the directly-executable program versus the ease of program development in an interpretive environment. The interpretive environment typically provides nicer debugging facilities and requires less time to rebuild a program after a change has been made.

Maintenance refers to the continuing process of changing an existing program to correct problems, add new functions or improve performance. (Trans)Portability is the ease with which a program can be made to run on a different machine.

Many of the other terms used have meanings which often can be inferred from their usage in more familiar contexts.

A BASIS FOR LANGUAGE EVALUATION

- LANGUAGE/TRANSLATOR-SPECIFIC
  - Development support
    - ...Well-documented, stable translator
    - ...Tools available (e.g. debugger)
    - ...Ease of team development
    - ...Available programmer skills
  - Future directions
    - ...Maintainability
    - ...Transportability
    - ...Likelihood of continued improvements
- APPLICATION-SPECIFIC
  - Expressive power for this task
  - Existing base of similar applications
  - Performance and size requirements

A BASIS FOR LANGUAGE EVALUATION

In choosing a language for a particular application, the characteristics of both must be considered.

Strength of support for program development is critical and is very much tied to a specific language and/or translator implementation. In addition to the quality of the translator and its documentation, this includes factors such as the availablity of development tools to facilitate program editing, debugging, and test monitoring and support for combining and verifying interfaces between separately developed program components. The availability of programmers already familiar with a language can be a significant advantage.

Considerations for the future include ease of maintenance and transportability, and the probability of continued improvements in both the language and the translator (is the product "alive"?). Maintainability is a product of many factors, but in general the more readable the language, the easier it is to understand the code and quickly correct a problem without breaking something else. Transportability is promoted by language standards.

An important application-specific consideration is the relative ease with which the solution to a particular problem can be expressed in a particular language. Although an application can usually be written in more than one language, the code in one case may be significantly more clear, compact and efficient if the features of one language are better suited to the problem. An existing base of similar applications written in a particular language provides both a clue as to the language suitability and a starting point for new applications.

The performance and size (memory) requirements for both the translator and the application must also be analyzed.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

CHARACTERISTICS OF COMMERCIAL APPLICATIONS

- **HIGH-VOLUME INPUT/OUTPUT**
- **RECORD-ORIENTED PROCESSING**
- **RELATIVELY STRAIGHT-FORWARD CALCULATIONS**

CHARACTERISTICS OF COMMERCIAL APPLICATIONS

The languages I will be describing are grouped as "Commercial Languages" since they are used primarily for business data processing applications. These applications are characterized by high-volume, record-oriented input and output with relatively straight-forward processing requirements. This means that in general large files are read and written, with calculations such as maintaining totals done on a line-by-line basis. Commercial languages therefore tend to emphasize file description and flexible input and output capabilities rather than features for describing complicated algorithms. A classic example of a commercial application is a payroll program.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

BASIC

- **PROMINENT CHARACTERISTICS**
  - **Widely available**
  - **Spans both scientific and business applications**
    - **. . . Built-in mathematical functions**
    - **. . . Good string processing and screen I/O**
  - **Popular as a "first language"**
  - **Easiest to develop and maintain smaller programs**
- **TI SPECIFICS**
  - **Strong portability path (standard plus additions)**
  - **Friendly development environment**

BASIC

The BASIC programming language was developed in the early sixties at Dartmouth College and is now widely-available on micro- and mini-computers. The language has features which make it attractive in both scientific and business applications. Since it is relatively easy to learn and provides a convenient development environment (usually interpretive), it is one of the most popular languages for new computer programmers. Small programs can usually be developed very quickly. Maintenance of large BASIC programs can be more difficult than with other languages, since the language is not really designed for large programs broken into components.

The specification for TI BASIC was jointly developed in 1978 by DSG and SC Consumer Products. This accounts for a strong similarity in 990 BASIC and the BASIC available on the TI Home Computer. In addition to the minimal ANSI-standard BASIC, 990 BASIC includes many additional features such as virtual arrays, improved VDT output, and the IF-THEN-ELSE statement. Support is also provided for program creation, editing and debugging.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

SAMPLE BASIC PROGRAM

```
100 PRINT ERASE ALL
110 PRINT :: PRINT :: PRINT :: PRINT :: PRINT
120 ACCEPT "Time per 'M'ile - - - Time for 'R'ace - - - 'Q'uit: ";A$
125 IF SEG$(A$,1,1) = "Q" THEN 490
130 PRINT
140 ACCEPT " Distance (in kilometers): ":D
150 PRINT
160 IF SEG$(A$,1,1) <> "M" THEN 300
170   ! Kilometers and time input, Minutes per mile output
180   ACCEPT "Time for race - - - - Minutes: ";M
190   ACCEPT "                      Seconds: ";S
200   M = M + (S/60)
210   MI = (1000*D)*(39.37/36)/1760
220   PRINT
230   PRINT "          Miles in race: ";MI
260   S = INT(((MPM - M)*60) + 0.5)
270   PRINT "Time per mile - - - - Minutes:";M
280   PRINT "                      Seconds:";S
290   GO TO 110
300 IF SEG$(A$,1,1) <> "R" THEN 450
310   ! Time per mile and kilometers input, Time per race output
320   ACCEPT "Time per mile - - - - Minutes: ":M
330   ACCEPT "                      Seconds: ":S
340   M = M + (S/60)
350   MI = D*.62136994945   ! D * Miles per Kilometer
360   TIME = M*MI
370   M = INT(TIME)
380   S = INT(((TIME - M)*60) + 0.5)
390   PRINT
400   PRINT "Time for race - - - - Minutes:";M
410   PRINT "                      Seconds:";S
420   GO TO 110
430 !
440   ! Error
450   PRINT "Say what?"
460   GO TO 110
470 !
480   ! Exit
490   BYE
```

SAMPLE BASIC PROGRAM

This is a sample BASIC program. Notice that source statements are numbered, are fairly simple, and that program variables are not declared.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		xxx	xxx				xxx
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	xxx
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	xxx	xxx	xxx	xxx	*	xxx
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

LANGUAGE COMPARISON MATRIX

This is the first of several appearances of the language comparison matrix. Note that the columns are broken into two groups: languages and tools. The notational conventions are: "*" = excellent, "+" = very good, (blank) = average, "-" = poor, and "xxx" = not applicable.

The highlighted column for BASIC emphasizes some of the major strengths and weaknesses of the language.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

COBOL

- **PROMINENT CHARACTERISTICS**
  - **Very widely used**
  - **Highly-standardized**
  - **"English-like" syntax**
- **TI SPECIFICS**
  - **Strong portability path**
  - **Interpretive, debugger available**
  - **Promising future directions**
    - **...COBOL Program Generator**
    - **...Good chance for future upgrades**

COBOL

COBOL (COmmon Business Oriented Language) is the most prevalent language for business data processing applications. Originally developed in the early 1960's, part of it's success has come from widespread use in government installations. One of the most colorful advocates of COBOL is Captain Grace Hopper of the U. S. Navy, a key contributer to the original language definition. A highly-developed language standard exists. The standard has evolved over a period of many years and is a crucial factor in the popularity and portablility of the language.

COBOL is characterized by English-like syntax intended to make the code "self-documenting". Statements like "SUBTRACT WITHHOLDING FROM GROSS-PAY GIVING NET-PAY" make the code fairly easy to read and understand. The data representations available are intented to facilitate the decimal computations common in business applications.

990 COBOL is very similar to RMCOBOL, a widely-available product. The compiler is interpretive and a debugger is provided.

The future is attractive for COBOL users. Plans include a COBOL Program Generator and the large user base promotes continued commitment to product improvement.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

SAMPLE COBOL PROGRAM

```
IDENTIFICATION DIVISION.
PROGRAM-ID. COMPRESS.
*     This program deletes comment lines from a COBOL program
*     and displays summary tallies.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. TI-990.
OBJECT-COMPUTER. TI-990.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT SOURCE-FILE ASSIGN TO INPUT "SOURCE".
     SELECT COMPRESSED-FILE ASSIGN TO OUTPUT "OUTPUT".
DATA DIVISION.
FILE SECTION.
FD SOURCE-FILE.
01 IN-REC.
     03 FILLER PIC X(6).
     03 INDICATOR PIC X.
     03 FILLER PIC X(73).
FD COMPRESSED-FILE.
01 OUT-REC PIC X(80).
WORKING-STORAGE SECTION.
77 LINE-COUNT PIC 9(5) VALUE 0.
77 COMMENT-COUNT PIC 9(5) VALUE 0.
77 COMMENT-PERCENTAGE PIC 999V99 USAGE COMPUTATIONAL.
77 PERCENTAGE-DISPLAY PIC 999.99 USAGE DISPLAY.
77 HOLD-THAT-SCREEN PIC X.
PROCEDURE DIVISION.
INITIALIZATION
     OPEN INPUT SOURCE-FILE.
     OPEN OUTPUT COMPRESSED-FILE.
IO-LOOP.
     ADD 1 TO LINE-COUNT.
     IF INDICATOR = "*"
       ADD 1 TO COMMENT-COUNT
     ELSE
       WRITE OUT-REC FROM IN-REC.
     GO TO IO-LOOP.
FINISH-UP.
     COMPUTE COMMENT-PERCENTAGE ROUNDED =
       COMMENT-COUNT * 100 / LINE-COUNT.
     MOVE COMMENT-PERCENTAGE TO PERCENTAGE-DISPLAY.
     DISPLAY "Total Lines = " ERASE, LINE-COUNT.
     DISPLAY "Number Comment Lines = ", COMMENT-COUNT,
             " = ", PERCENTAGE-DISPLAY, "%".
     ACCEPT HOLD-THAT-SCREEN.
```

SAMPLE COBOL PROGRAM

This a sample COBOL program. Notice the extensive declarations and the English-like sentence structure of the procedural code.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

LANGUAGE COMPARISON MATRIX

The highlighted column for COBOL in this table emphasizes some of the major strengths and weaknesses of the language.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

RPG

- **PROMINENT CHARACTERISTICS**
  - **Features specifically tailored for Report Generation**
  - **Cryptic input format is also powerful and compact**
- **TI SPECIFICS**
  - **Some menu assistance in specification entry**
  - **Recent and future directions**
    - **...Available on DNOS**
    - **...Limited development and support**

RPG

RPG (Report Program Generator) provides the capability for generating reports given a compact description of the content and layout of the reports. Originally developed by IBM in the mid- 1960's, the language features are specifically intended to facilitate the specification of report properties such as pagination conditions, summary calculations, and print positions. The cryptic input format can be considered as both a postive and negative feature. Reports can often be described and generated very quickly by an RPG expert, but readability is limited.

Menus available on the 990 provide some assistance in preparing Report Specifications.

RPG is now available on both the DX10 and DNOS operating systems, although future development and support are limited.

SAMPLE RPG SPECIFICATION

RPG II OUTPUT SPECIFICATIONS

PROGRAM
PROGRAMMER | DATE

PROGRAM IDENTIFICATION 75 76 77 78 79 80

Page (1-2)	Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) / OR-AND (14-16)	Fetch (F) / ADD (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-25)	And (26-28)	And (29-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position (40-43)	P/B/L/R (44)	Constant or Edit Word (45-70)
	Ø1Ø	O	PRTFIL	H			2	Ø6		1P								
	Ø2Ø	O		OR						OA								
	Ø3Ø	O														12		'SALES REPORT'
	Ø4Ø	O											UDATE	Y		21		
	Ø5Ø	O														27		'PAGE'
	Ø6Ø	O											PAGE	Z		32		
	Ø7Ø	O		H			2			1P								
	Ø8Ø	O		OR						OA								
	Ø9Ø	O														4		'SALE'
	1ØØ	O														25		'AMOUNT'
	11Ø	O		D		1				1Ø								
	12Ø	O									L1		ITEM			4		
	13Ø	O											AMT	3		24		
	14Ø	O		T		2	1			L1								
	15Ø	O											SUBTOT	3	B	24		
	16Ø	O														27		'*'
	17Ø	O		T		2				LR								
	18Ø	O											TOTAL	3		24		
	19Ø	O														28		'**'
	2ØØ	O	DSKFIL	D	A	D	D			77								

COMMAS	PRINT ZERO BALANCES	NO SIGN	CR	-	
YES	YES	1	A	J	X = REMOVE PLUS SIGN
YES	NO	2	B	K	Y = DATE FIELD EDIT
NO	YES	3	C	L	Z = ZERO SUPPRESS
NO	NO	4	D	M	

NUMBER OF SHEETS PER PAD MAY VARY SLIGHTLY.
(B)137950(1/2)

SAMPLE RPG SPECIFICATION

RPG programs are written on "specification forms" which separate the descriptions of input, calculations, and output. This is an example of an output specification for a sales report.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

LANGUAGE COMPARISON MATRIX

The highlighted column for RPG in this table emphasizes some of the major strengths and weaknesses of the language.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

SORT/MERGE

- **PROMINENT CHARACTERISTICS**
  - **More a tool than a classical language**
  - **Multiple usage contexts**
    - **...Stand-alone**
    - **...Subroutines called from application program**
  - **Cryptic interface**
- **TI SPECIFICS**
  - **New release in 1983**
    - **..."Friendly" front-end**
    - **...Sequence-preserving sort**

SORT/MERGE

The SORT/MERGE package provides the capability to produce files which have been re-ordered or combined based on "key" values. In addition, other features such as tallies and reports are available. Although SORT/MERGE is not considered to be a language in the usual sense of the term, it loosely fits our definition of a mechanism for describing an algorithm for input transformation. The package can be used in a stand-alone fashion or (more frequently) linked with an application program and called as a subroutine. These programs (usually written in COBOL) can receive records for processing in sorted order.

The current interface to SORT/MERGE consists of a series of cryptic fixed-format control lines. During 1983, a more natural interface will be provided, although the existing input format will be supported for compatibility.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

SAMPLE SORT/MERGE CONTROL FILES

CURRENT FORMAT

```
00000HS0RTR  42A     42  12000
00001DOS@PROC@
00002DA00420256
00003DWEDS01
00004DIS@PROC@
00008DA0042     400
00010FNC     1  42
/*
```

PROPOSED ALTERNATIVE FORMAT

```
Field CCID contains integer from 1 to 8
Field NAME contains characters from 10 to 50
Sort using file NAMELIST
     on ascending CCID ascending NAME
     return file NAMESOUT
```

SAMPLE SORT/MERGE CONTROL FILES

Sample SORT control files are shown here, the first using the current input format, and the second in the new and improved format. The new front-end will actually allow a command file consisting of only the word "SORT" which would sort a default input file based on the value of entire lines. Although the existing input format will still be supported, this new format provides notation to cover all current SORT/MERGE options.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

LANGUAGE COMPARISON MATRIX

The highlighted column for SORT/MERGE in this table emphasizes some of the major strengths and weaknesses of the tool.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

PSYCHOLOGICAL VARIATION IN PROGRAMMERS BY LANGUAGE

PREFERRED LANGUAGE	LANGUAGE-DEPENDENT SOCIAL ADJUSTABILITY INDEX [LDSAI] (*)
BASIC	0.68 = Star Trek fanatic
C	0.97 = expediency over idealism
COBOL	0.80 = extroverted, verbose
FORTRAN	0.18 = dresses funny
PASCAL	0.44 = self-righteous
RPG	0.02 = introverted, brusque

(*) Expressed as normalized ratio of deviation.

PSYCHOLOGICAL VARIATION IN PROGRAMMERS BY LANGUAGE

The management of computer programming is not a well-understood process. Researchers continually attempt to answer questions such as:

In what ways can we measure programmer aptitude?

How do we deal with the programmers we have?

Is matching programmer and language as important
as matching application and language?

The Language-Dependent Social Adjustability Index (or "LDSAI" for short) seems to provide a useful device both for recognizing programmer potential and communicating with those who have it.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

SCIENTIFIC AND SYSTEMS LANGUAGES

- **INTRODUCTION**
- **CHARACTERISTICS OF SCIENTIFIC APPLICATIONS**
- **CHARACTERISTICS OF SYSTEMS IMPLEMENTATIONS**
- **SURVEY OF SCIENTIFIC AND SYSTEMS LANGUAGES**
  - **FORTRAN**
  - **PASCAL**
  - **C**

SCIENTIFIC AND SYSTEMS LANGUAGES

This portion of our talk will consist of a very brief overview and comparison of three high level languages designed for scientific applications and systems implementation programming. These languages are FORTRAN, Pascal, and C. FORTRAN and Pascal are available on the Texas Instruments 990 computer systems. C is not currently implemented on the 990, but has so much potential it is included for reference.

Before we get into the specifics of these three languages we will first look at some of the characteristics of scientific applications and systems implementations.

We will then look at some of the features of each of the languages that make them suitable for their particular applications. As in the commercial language part of our presentation, I will show you some simple programs, not to teach you the detailed syntax, but simply to give you a feel for the language. I will also talk briefly about the history of the language, and about some specific Texas Instruments features.

CHARACTERISTICS OF SCIENTIFIC APPLICATIONS

- LARGE NUMBER OF NUMERICAL CALCULATIONS
- EXECUTION SPEED AND EFFICIENCY ARE CRITICAL
- PROGRAMMING DONE BY ENGINEERS AND MATHEMATICIANS

CHARACTERISTICS OF SCIENTIFIC APPLICATIONS

The most significant characteristic of scientific applications is the large number of numerical calculations usually involved. These are quite often done in real or floating point arithmetic. Because of this emphasis on calculations or number crunching as we call it, a compiler which produces code that is efficient and runs fast is critical. This obviously rules out using an interpretive language such as BASIC or COBOL for these types of applications because interpretive languages execute much slower than compiled languages. Another characteristic of these applications is the programming is quite often done by engineers or mathematicians. Therefore using a language that engineers commonly know would be a big asset because it would reduce the learning curve element of writing programs. A typical example of a scientific application is computing the trajectory of a missile.

CHARACTERISTICS OF SYSTEMS IMPLEMENTATIONS

- COMPLEX ALGORITHMS
- LOW LEVEL INTERFACES ARE NECESSARY
- BIT LEVEL MANIPULATION REQUIRED
- EXECUTION SPEED AND EFFICIENCY ARE CRITICAL
- LITTLE EMPHASIS PLACED ON I/O
- PROGRAMMING DONE BY COMPUTER SCIENTISTS

CHARACTERISTICS OF SYSTEMS IMPLEMENTATIONS

Systems implementations involve the most difficult kind of programming. The programs are typically so large and the algorithms so complex that one person can usually understand only pieces of the total implementation. This makes a type of programming called structured programming a necessity. Structured programming is a modern technique of breaking up a program into small pieces with well defined inputs and outputs, and is considered the best way of writing complex programs. Languages suitable for systems implementations should encourage the use of structured programming techniques. Because parts of systems implementations need to be written in assembly language for one reason or another, the high level language you use needs to be able to easily interface with these low level routines. Many assembly language routines can be eliminated if the high level language has facilities for bit level manipulation of data. Often pieces of a systems implementation need to operate under extreme speed or space constraints. Therefore, as with scientific applications, a language that produces code that is efficient and runs fast is critical. A sophisticated I/O capability is one of the least important features required by a systems implementation language. Often systems implementation programs are written by people with a strong computer science background. Some typical pieces of a systems implementation are operating systems, compilers, and communication software.

FORTRAN

- PROMINENT CHARACTERISTICS
  - Designed and implemented in late 1950's
  - First accepted high level language
  - Scientific applications
  - Efficient execution
  - Well known
  - Easily learned
  - Libraries
  - Simple syntax weak in control structures
- TI SPECIFICS
  - Migration path between FORTRAN 66 and FORTRAN 78
  - Excellent quality code produced
  - Extra debug features

FORTRAN

FORTRAN, designed and implemented in the late 1950's was the first widely used high level programming language. Before this time all serious programming was done in assembly language because no available high level language could produce efficient code. FORTRAN is widely used for scientific and numeric computing. This is true in part because its primary design goal is one of execution efficiency. The language is well known because it has been around for such a long time and is a required language in most engineering schools across the country. This feature has two important consequences. Engineers writing new scientific FORTRAN applications will in many cases already be familiar with the language, therefore little time needs to be spent on this learning process. In addition many routines to do commonly needed tasks have already been written and are readily available. These types of routines are refered software libraries. For the programmer who does not already know FORTRAN, the language is easily learned due to its simplicity. The disadvantage of this is although the syntax of FORTRAN is sufficient for most scientific applications, the fact that it is weak in control structures hinders structured programming.

The most modern version of FORTRAN is called FORTRAN 78. An extremely good feature of the Texas Instruments FORTRAN compiler is it allows you to combine programs written in an earlier version of FORTRAN called FORTRAN 66 with the modern version. This means that you can still conveniently use all those old FORTRAN programs you may have while taking advantage of newer features. The Texas Instruments FORTRAN compiler also produces excellent quality code. Therefore, programs compiled with TI FORTRAN should run as fast and as efficiently as possible. The Texas Instruments FORTRAN also provides some debug capabilities not normally found in a compiler of this kind.

TI-MIX 1983
Techniques and Concepts for the Non-Technical or New User
Survey of 990 Languages

SAMPLE FORTRAN PROGRAM

```
      PROGRAM RTRANG
C
C CHECK FOR RIGHT TRIANGLE
C
      READ (1,6)A,B,C
      WRITE(2,7)A,B,C
      IF(ABS(A*A + B*B - C*C) - .1) 20, 5, 5
      IF(ABS(B*B + C*C - A*A - .1) 20,10,10
      IF(ABS(C*C + A*A - B*B) - .1) 20,15,15
      WRITE(2,8)
      GO TO 1
      WRITE(2,9)
      FORMAT(3F10.5)
      FORMAT (1HO,3F10.5)
      FORMAT(29H THIS IS NOT A RIGHT TRIANGLE)
      FORMAT(25H THIS IS A RIGHT TRIANGLE)
      END
```

SAMPLE FORTRAN PROGRAM

Here is a sample FORTRAN program. Notice that some of the source statements have labels and that there are GO TO statements jumping to these labels. This is an indication that the language does not encourage the use of structured programming techniques.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+ /2	+	+
LEARNING CURVE	*						+		+	+ /2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+ /3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+ /4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

PASCAL

- PROMINENT CHARACTERISTICS
  - 1968 offshoot of ALGOL
  - General purpose language
  - Systems applications
  - Block structured control statements
  - User defined types
  - Encourages structured programming
  - Verbose
  - Strongly typed
  - Well known
- TI SPECIFICS
  - Enables low level interfacing
  - Bit level manipulation
  - Possible to override strong typing

PASCAL

Pascal was designed and implemented in 1968 by Niklaus Wirth as an offshoot of Algol 60, another high level language. Pascal is a general purpose language suitable for a wide varity of applications. It is used heavily by university computer science departments because its powerful structures lend themselves to the teaching of structured programming concepts. Pascal subsets are often used as systems implementation languages. For example, several of Texas Instruments' compilers, much of our communication software, and many of our operating system utilities are written in a stripped down version of Pascal that produces very high quality code. Pascal's most notable features are block structured control statements and user defined types. Although this makes Pascal more difficult to learn than FORTRAN, these features were designed specifically to encourage the use of modern structured programming techniques. Pascal is a verbose language whose syntax uses many english sentence constructions. It also encourages the use of long descriptive variable names. These two features combined cause programs in Pascal to be self documenting. Why, even managers can understand Pascal programs! This will save you money in the long run. Pascal is a strongly typed language. This means that there are many restrictions placed on operations between objects of different types. This is actually a double edged sword. It keeps the programmer from making mistakes in manipulating objects. These kinds of mistakes can be very time consuming to track down. However, it can be overly restrictive in the sense of hindering the knowledgable programmer from performing legitimate operations on objects of unlike types. Like FORTRAN but to a lesser degree, Pascal is a well known and commonly used language. This gives users of Pascal the benefit of available software library packages containing useful functions and a high probability that their programmers will already be familiar with the language.

Texas Instruments Pascal enables you to interface with low level routines and do bit level manipulation of data, which is required for systems implementation programming. It also provides a way of overriding the strong type checking feature of the language.

TI-MIX 1983
Techniques and Concepts for the Non-Technical or New User
Survey of 990 Languages

SAMPLE PASCAL PROGRAM

```
PROGRAM SHIFT;

(* PURPOSE: Shift lines 4 spaces to right and number *)

CONST = 4;
        MaxColumn = 80;
        MinColumn = 1;

VAR     Buff: PACKED ARRAY [1..80] OF CHAR;
        LineNum: INTEGER;
        Stat: INTEGER;

BEGIN
  RESET(INPUT);
  REWRITE(OUTPUT);
  LineNum := 1;
  WHILE NOT EOF(INPUT) DO
    BEGIN
      READLN(Buff);
          Buff[Column + NumberOfSpaces] := Buff[Column];
      ENCODE(Buff, 1, Stat, LineNum: NumberOfSpaces);
      WRITELN(Buff);
      LineNum := LineNum + 1;
    END;
END.
```

SAMPLE PASCAL PROGRAM

Here is a sample PASCAL program. Notice how many of the constructions use words in the way they are used in sentences, For example, "FOR" the current column being equal to the maximum column minus the number of spaces to shift "DOWNTO" the minimum column, "DO" this line of code. Also notice the descriptive names like "NumberOfSpaces".

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		xxx	xxx				xxx
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	xxx
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	xxx	xxx	xxx	xxx	*	xxx
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

C

- **PROMINENT CHARACTERISTICS**
  - **Early 1970's as part of UNIX**
  - **General purpose language**
  - **Systems applications**
  - **Similar to PASCAL**
  - **Loosely typed**
  - **Source level optimization**
  - **Bit level manipulation**
  - **Concise**
  - **Designed for the expert user**

C

C was designed and implemented about the same time as Pascal by Dennis Ritchie for use on and development of the UNIX operating system. Most of UNIX and the tools that run on it are written in C. C, like Pascal, is a general purpose language suitable for many different kinds of applications. Unlike Pascal, it is not a teaching language heavily used by computer science departments, but instead was conceived specifically as a systems implementation language. In other words, it is more utilitarian in nature. C has many of the same features as Pascal which encourage the use of structured programming. In contrast to Pascal, it is not a strongly typed language, which gives the programmer more flexibility in manipulating objects. In addition C has features which allow more control over the machine code being produced at the source level, and some features to do bit level operations. In these respects C is more of a bridge between an assembly language and a high level language, thus its excellent suitability for systems implementations. Another interesting characteristic of C is its emphasis on concision, which is very much in line with the UNIX philosophy of tools designed for the expert user rather than the novice. This feature allows the experienced C programmer to develop applications more quickly than with a verbose language such as Pascal at the expense of some program readability. C is not as well known as Pascal but it is currently receiving much attention due in part to its association with the UNIX operating system.

TI-MIX 1983
Techniques and Concepts for the Non-Technical or New User
Survey of 990 Languages

SAMPLE C PROGRAM

```
#define   YES 1
#define   NO  1

main ( )

  /* PURPOSE: Count lines, words, and chars in input */

     int c, nl, nw, nc, inword;

     inword = NO;
     while ((c = getchar()) != EOF) {
          ++nc;
          if (c == '/n')
               ++nl;
          if (c ==' ' || c == '/n' || c == '/t')
               inword = NO;
          else if (inword == NO) {
               inword = YES;
               ++nw;
          }
     }
     printf("%d, %d, %d/n", nl, nw, nc);
```

SAMPLE C PROGRAM

Here is a sample C program. Notice how it much more cryptic than Pascal. These symbols are equivalent to the "BEGIN" and "END" that were in the Pascal program. Also notice some of the conciseness the language gives you such as setting all three of these variables to zero in one statement.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

SUMMARY

- **HIGH LEVEL VERSUS LOW LEVEL**
  - **Greater ease of development**
  - **Greater maintainability**
  - **Greater portability**
  - **Generally acceptable efficiency and execution speed**

SUMMARY

In summary I have tried to hit on some of the highlights and primary uses of three important high level languages. In going over these features you should begin to see some of the advantages of using one of these languages over an assembly language, even for an application where execution speed and efficiency are important considerations. You need to keep in mind when choosing a language for an application, the importance of ease of development, maintainability, and portability which translate into dollars saved during the life cycle of a program. All three languages are vastly superior in these areas than an assembly language while maintaining an acceptable level of speed and efficiency.

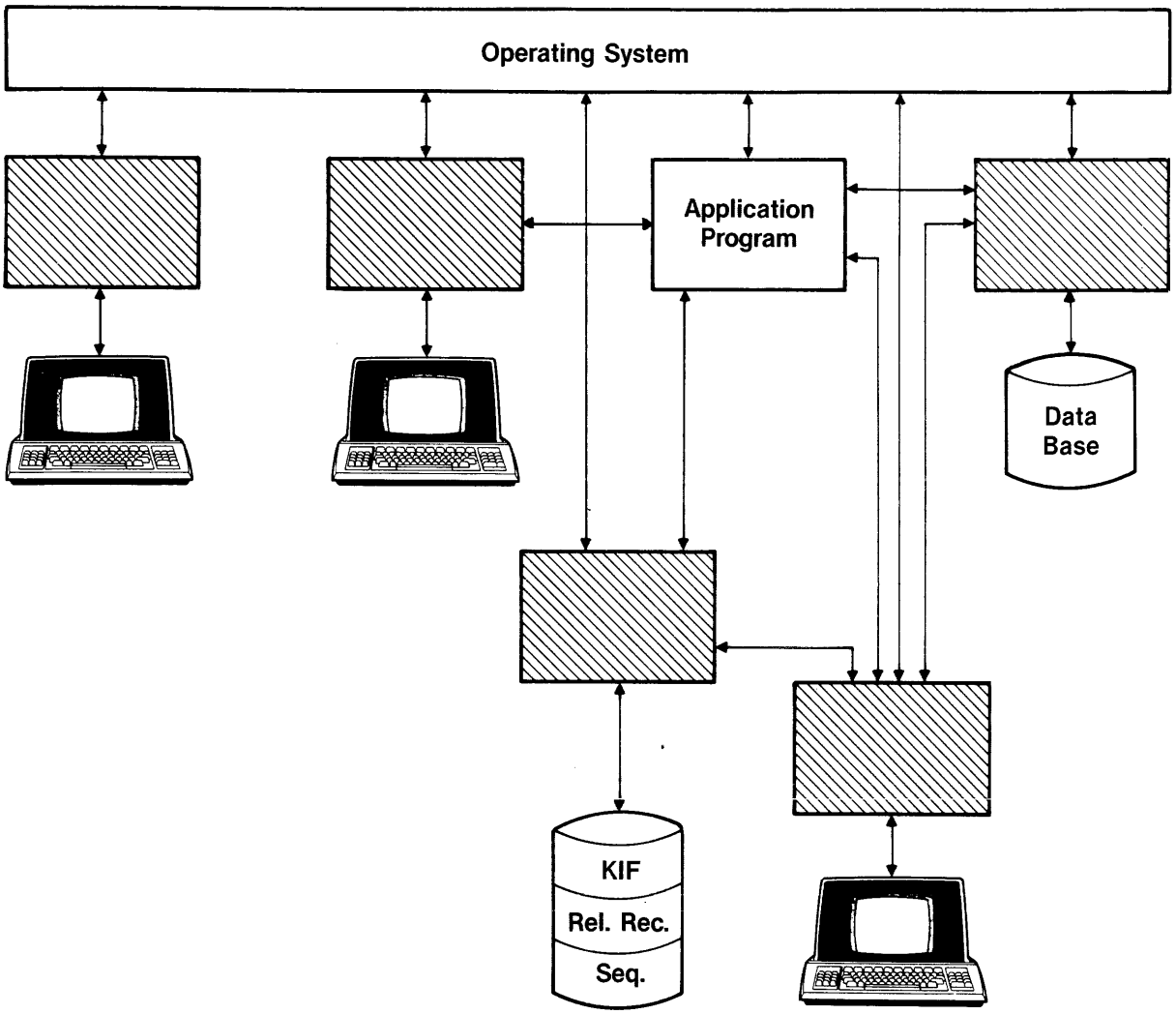
PRODUCTIVITY SOFTWARE

Greater Productivity Not More Programmers

* Greater Productivity - Not More Programmers

This section of the presentation will show how five products can bring greater productivity to a company's data processing department.

PRODUCTIVITY AIDS



* Productivity Aids - Centered around the application

These five products work with Texas Instruments operating systems. They work closely with the applicaton program.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

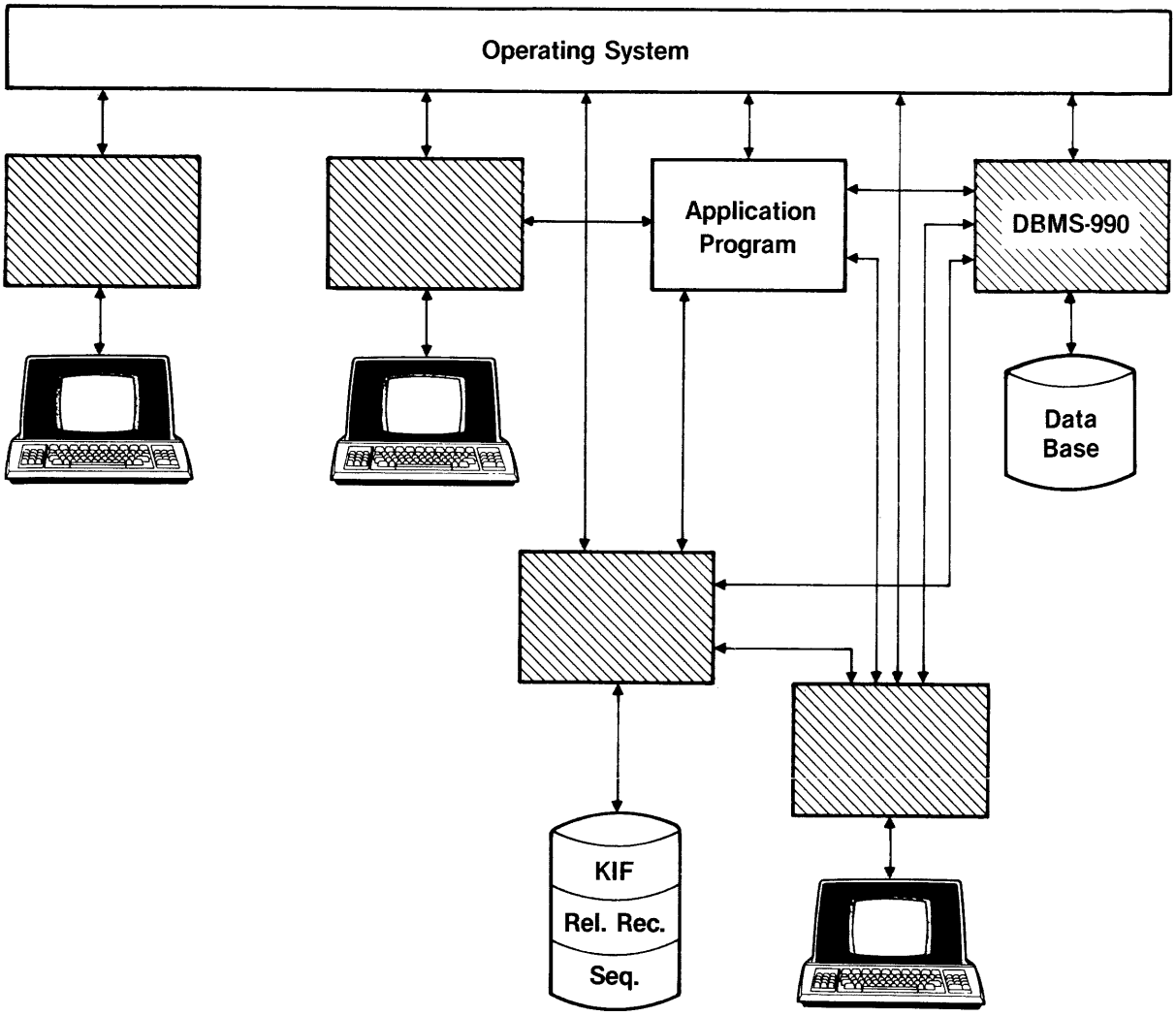
TECHNIQUES AND CONCEPTS FOR NONTECHNICAL USER

- DATA MANAGEMENT
  - Problems concerning a DP manager
  - DBMS-990
  - Query-990
  - DD-990
- PRODUCTIVITY BOOSTERS
  - TIFORM
  - TIPE

* Techniques and Concepts for Nontechnical User

This section of this presentation will cover two different areas. First I will discuss data management. I will describe some of the problems that face a data processing manager. Then I will propose some solutions using Texas Instruments products. These products are DBMS-990, Query-990, and Data Dictionary-990. Next, I will cover two products that can greatly improve the productivity in a given department. The products are TIFORM, the TI screen management package, and TIPE, the TI word processor.

PRODUCTIVITY AIDS



* Productivity Aids - DBMS-990

The first product to be discussed is DBMS-990. This product will be referenced several times throughout the presentation.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

DATA MANAGEMENT

- **PROBLEMS FACING A DATA PROCESSING MANAGER**
  - **Amount of data, currently and in future**
  - **Amount of time spent retrieving information from data in its current form (eg. searching through file cabinets)**
  - **Number of times data will be inserted, deleted, or updated**
  - **Amount of redundancy in data in its current form (eg. the number of times employees names and addresses are printed on various documents)**
  - **Necessity for security measures**
  - **Necessity for some transaction rollback function**

* Data Management - Problems facing a Data Processing Manager

There are several problems invloving data management that a manager ought to be concerned with. The amount of data he or she must keep track of is a major problem. If a business is doing particularly well, the sheer quantity of data may increase dramatically. Another big problem might be the amount of time spent extracting data in its current format. Data redundancy can also be a big problem. For example, an employee's name and address may appear on every document that involves that particular employee or his dependants. If the employee changes his or her address, each of the documents would need to be modified. This also brings up a similar problem : data verification. If there is a great deal of redundancy, it would be difficult to insure that all the data is accurate. Security can also a major worry. There may be some files containing sensitive information not for common access. There is also a major need for keeping track of all the transactions that are performed on the data. This is useful in the event that you need to "un-do" an operation. These are all important problems a manager may have to face.

DATA MANAGEMENT

- DBMS-990 ALLOWS A USER TO:
  - Define logical construct to base the data organization
  - Easily insert data to allow the data base to grow with the quantity of data
  - Produce reports on the data, quickly and accurately using various utilities and application programs
  - Reduce data redundancy
  - Insure data accuracy
  - Implement security measures
  - Records successful updates to DBMS files
    - ...Useful in transaction rollback
    - ...Used for backups in the event of a system crash

* Data Management - DBMS-990 Allows a User to ...

A data base management system is a system for consistent organization of data. With DBMS-990, users can organize, store, retrieve, and modify their data by means of a logical construct. The user can group together related information, give the group a meaningful name, and reference all the information in that group through that name. DBMS-990 allows the user to select the pieces of information he or she wants to group together. As stated earlier, the amount of data you need to keep track of can be a major problem, as is the possibliity for growth in the future. A data base management system can grow easily with the quantity of data. It is also easy to see that the cost of converting to a data base system will increase as the quantity of data increases. The amount of time you spend searching for data and putting it in some form that makes sense can be noticeably reduced using DBMS-990. There are powerful utilities that interface with DBMS-990, most notably Query-990. I have already shown how Query-990 may be used to produce a report in a few minutes that may have taken hours using non-DBMS methods. As for the amount of redundancy existing in your data, DBMS-990 may be used to set up links between related information instead of having to duplicate the information. If we go back to our first example, the employee's name and address need only appear once instead of appearing many times on many documents. Links could be established to all other necessary information. The problem of data verification disappears also, since the fact that the data appears once insures that any updates to the data will update all occurrences. As for security, DBMS-990 offers a good set of security features. Certain files can be made to require certain passwords to allow access. It may take one password to insert data, and a different one to delete data. These specifications can be made by the user. DBMS-990 has an automatic transaction logging capability. This is useful in the event of machine failure. For example, if a bank was supposed to withdraw one hundred dollars from your savings account and deposit it into your checking account, and there was a machine failure at some time during the transaction, the logging facility will tell exactly which parts of the transaction were successfully completed. If the part that withdrew the money was completed, but not the deposit, then you certainly wouldn't want to have a second one hundred dollars withdrawn from your savings account. Transaction logging can insure that this doesn't happen.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

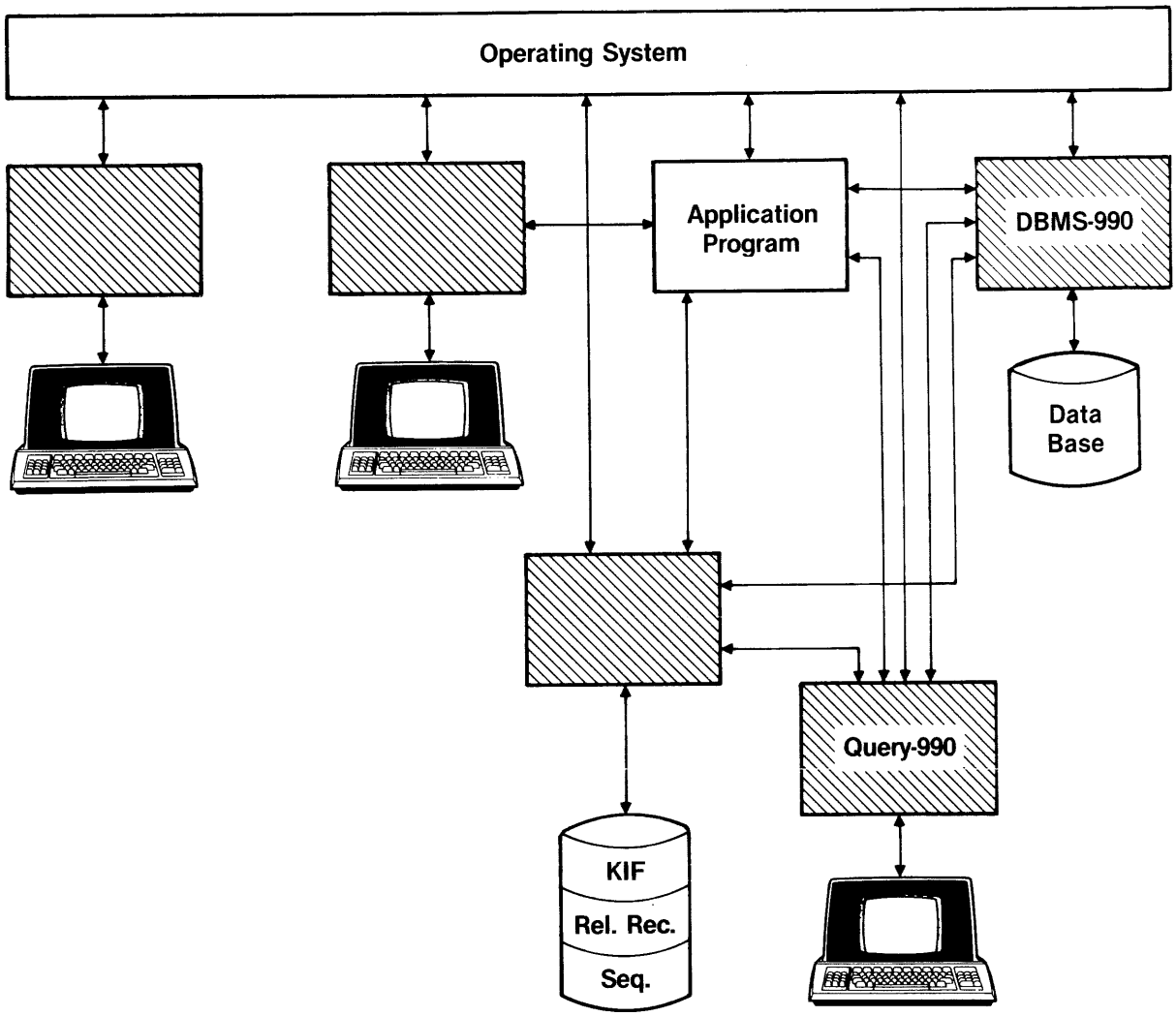
	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

* Language Comparison Matrix - DBMS-990

This is the same language comparison chart from the presentation on scientific languages. It will be shown after each of the products is presented. We can see that it is not particularly fast in execution, but DBMS-990 is rated highly in the business applications category. It is also strongly rated in the data management-data access category. This is because these were explicit design intentions.

PRODUCTIVITY AIDS



* Productivity Aids - Query-990

Our next product is Query-990.

DATA MANAGEMENT

- QUERY-990
  - Powerful retrieval system
    - . . . Non-procedural language
    - . . . English-like syntax
    - . . . Insert, delete, and update a file
    - . . . Report formatting capabilities
  - Two modes of operation
    - . . . Interactive
    - . . . Batch
  - Guided Query
    - . . . Beginning Query-990 user
    - . . . User-friendly format
    - . . . Extensive help facilities

* Data Management - Query-990

Query-990 is a powerful retrieval system. The major design objective was to produce a tool for extraction of data from DBMS-990 files. In a later release, the facility to retrieve data from conventional files was added. Conventional files that Query-990 can access are key indexed files, relative record files, and sequential files created by DD-990, the Texas Instruments Data Dictionary. Query-990 is a non-procedural language. Its English-like syntax makes Query-990 easily understood by programmers and non-programmers alike. With proper preparation, queries like "LIST EMPLOYEE-NAME EMPLOYEE-SALES FROM SALES-INFO-FILE WHERE COMMISSION > 5000 SORTED BY JOB-SALARY" can be run, producing results that can be understood by almost anyone. Query-990 allows a user to modify files. A user can insert data into a file, delete data, or change existing data. Query-990 also has report formating capabilities to allow a user to produce output including totals and averages of data in the report. Query-990 has two modes of operation : an interactive mode, and a batch mode. In the interactive mode, also referred to as the stand alone mode, the user can build a query statement, execute the query, save the query statement in a file for execution at a later time, and save the output from his query in a file. In interactive mode, the user may edit a query statement from a previous session, and re-execute the new query statement. In the batch mode, Query-990 may be called from an application program written in COBOL, Pascal, FORTRAN, or BASIC. Query-990 also has a utility that is designed for the first time user, called Guided Query. Guided Query provides a user friendly format with concise directions during the session, and extensive help facilities, making it an excellent learning tool.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C – (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		×××	×××				×××
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	×××
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	×××	×××	×××	×××	*	×××
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

* Language Comparison Matrix - Query-990

In looking at Query-990 in the comparison matrix, we see that Query gets rated highly in the learning curve category. This is because of the Guided Query utility. It is also rated highly in the maintainability category, as well as the business applications and data management and access categories.

```
LIST 01 FROM CUST

TERESA LOWEREY      2163151223    3    24    80
JOHN TORTERELLA     2166545656    5    24    79
ALICE BERTRAND      2165456471   12     4    80
RENALDO NOCON       5123454655    9    20    75
KIM BARNES          5126543322    1    24    80
ROBERT MACDOWELL    4196545658    5    31    78
NANCY MONACHINO     5124887544    8    19    81
JEFFERY THOMAS      2163535411   10     8    80
FRANCIS DIGBY       4196564454    6    14    79
```

TI-MIX 1983

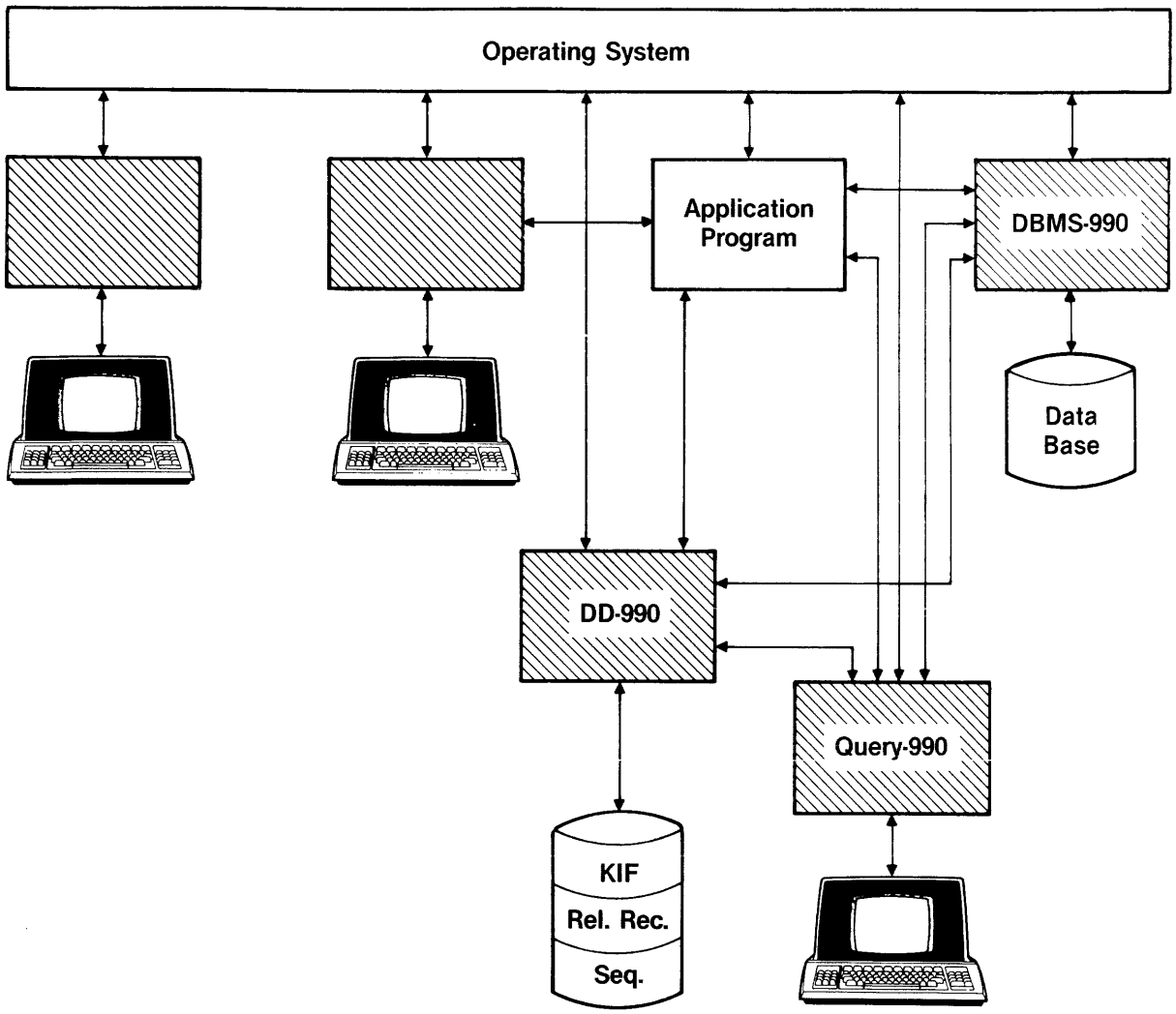
PRODUCTIVITY BOOSTERS

```
program TESTQUERY;

type

    CONTROL_BLOCK = record
        PSWD  : packed array [1. . 4] of char;
        FNCT;
        STAT  : packed array [1. . 2] of char;
        DBFL,
        LOC1,
        LOC2,
        KYID  : packed array [1. . 4] of char;
        KEYV  : packed array [1. . 6] of char;
        TERM  : integer;
    end;

    LINE_LIST = record
        LL :   packed array [1. . 50] of char;
        TERM  : integer;
    end;

    DATA_AREA = record
        NAME : packed array [1. . 20] of char;
        PHON : packed array [1. . 10] of char;
        ADMO,
        ADDA,
        ADYR,
        TERM  : integer;
    end;

var

    CB : CONTROL_BLOCK;
    DA : DATA_AREA;
    LLIST : LINE_LIST;

    BUFFER : packed array [1. . 80] of char;
    S : integer;
    ENDLINE : boolean;

procedure DBMSYS ( var CONT_BLOCK : CONTROL_BLOCK; var CBE : integer;
                   var L_LIST : LINE_LIST; var LLE : integer;
                   var D_AREA : DATA_AREA; var DAE : integer); external;

BEGIN (* TESTQUERY *)

  rewrite(OUTPUT);

  CB.PSWD := 'DBMS';
  CB.FNCT := 'OF';
  CB.KYID := 'EXCL';
  CB.DBFL := 'CUST';

  DBMSYS(CB,CB. TERM,CB. TERM; :LINE_LIST,CB. TERM,
                    CB. TERM; :DATA_AREA,CB. TERM);

  if CB. STAT <> '**' then
    begin
      for I := 1 to 80 do BUFFER(I) := ' ';
      encode(BUFFER,1,S,'Open Error : ');
      encode(BUFFER,14,S,CB. STAT);
      write(OUTPUT,BUFFER);
    end;

  ENDLINE := false;

  CB.LOC1 := '*****';
  CB.LOC2 := '*****';

  encode(LLIST.LL,1,S, 'LINE = 01');
  encode(LLIST.LL,8,S, '');
  encode(LLIST.LL,09,S, 'NAME');
  encode(LLIST.LL,13,S, 'PHON');
  encode(LLIST.LL,17,S, 'ADMO');
  encode(LLIST.LL,21,S, 'ADDA');
  encode(LLIST.LL,25,S, 'ADYR');
  encode(LLIST.LL,29,S, '****RLSE');

  while not ENDLINE do
    begin

      DBMSYS(CB,CB. TERM,LLIST,LLIST. TERM,DA,DA. TERM);

      encode(BUFFER,1,S,DA. NAME);
      encode(BUFFER,23,&,DA. PHON);
      encode(BUFFER, 35,S,DA. ADMO);
      encode(BUFFER,41,S,DA. ADDA);
      encode(BUFFER,45,S,DA. ADYR);

      write(OUTPUT,BUFFER);
      if CB.LOC2 = '****' then ENDLINE := true;

    end;

  CB.FNCT := 'CF';

  DBMSYS(CB,CB. TERM, CB. TERM; :LINE_LIST,CB. TERM,
                    CB. TERM; :DATA_AREA,CB. TERM);

  END.
```

* Data Management - Query-990 Example vs. Application Program With DBMS Call

To show a small example of how Query-990 may increase productivity, I have two illustrations. The first one is probably the shortest query statement that can be written, one line in length. The output from this query is also dislayed. The second illustration is a one hundred line Pascal program using a call to DBMS-990 to extract data from a DBMS file. Both segments of code produce the same output. This example is not meant to indicate that there is always a 1:100 ratio of size of Query-990 statements to application program size. It is only an example of probably the simplest query statement and the application program needed to produce the same results.

PRODUCTIVITY AIDS



* Productivity Aids - DD-990

Data Dictionary, called DD-990, is the next product.

DATA MANAGEMENT

- **DD-990**
  - **Provides centralized interface to all data definition**
  - **Two modes of data definition**
    - **...Interactive : Interactive Data Librarian**
    - **...Batch : Batch Data Librarian**
  - **Allows complete data access to all file types**
    - **...DBMS-990**
    - **...Key indexed files**
    - **...Relative record files**
    - **...Sequential files**
  - **Extensive utilities**
    - **...FD section of a COBOL program from file definition**
    - **...Cross reference on a particular field defined in the dictionary**

* Data Management - DD-990

DD-990 is the Texas Instruments data dictionary. It provides a centralized interface to all the data definitions in an environment. DD-990 has four categories of entities that it recognizes. They are fields, groups, files, and programs. DD-990 keeps track of which files and which programs reference which fields and groups. This is extremely important in the event that a particular field needs to be alterred. Using a cross reference utility, you can obtain a listing containing all the occurrences of a particular field, or of all fields. There are two methods of making definitions in DD-990. You may interactively insert the definitions, using the Interactive Data Librarian, in which you are prompted for the necessary information needed at a particular time. Or you may make definitions in a batch mode, using the Batch Data Librarian. In either case, data definitions may be made for DBMS-990 files, or any of the three conventionl file types : key indexed files, relative record files, or sequential files. DD-990 has extensive facilities to generate reports on the information stored in the library, that is the collection of data definitions and their relationships. A good example of a powerful utility is the cross reference utility. With it, a user can get a listing containing every use of a certain field, group, or file used in any context in the dictionary. Another powerful and useful utility is the GCB utility. With it, a user can generate the FD section of a COBOL program for any file in the dictionary.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

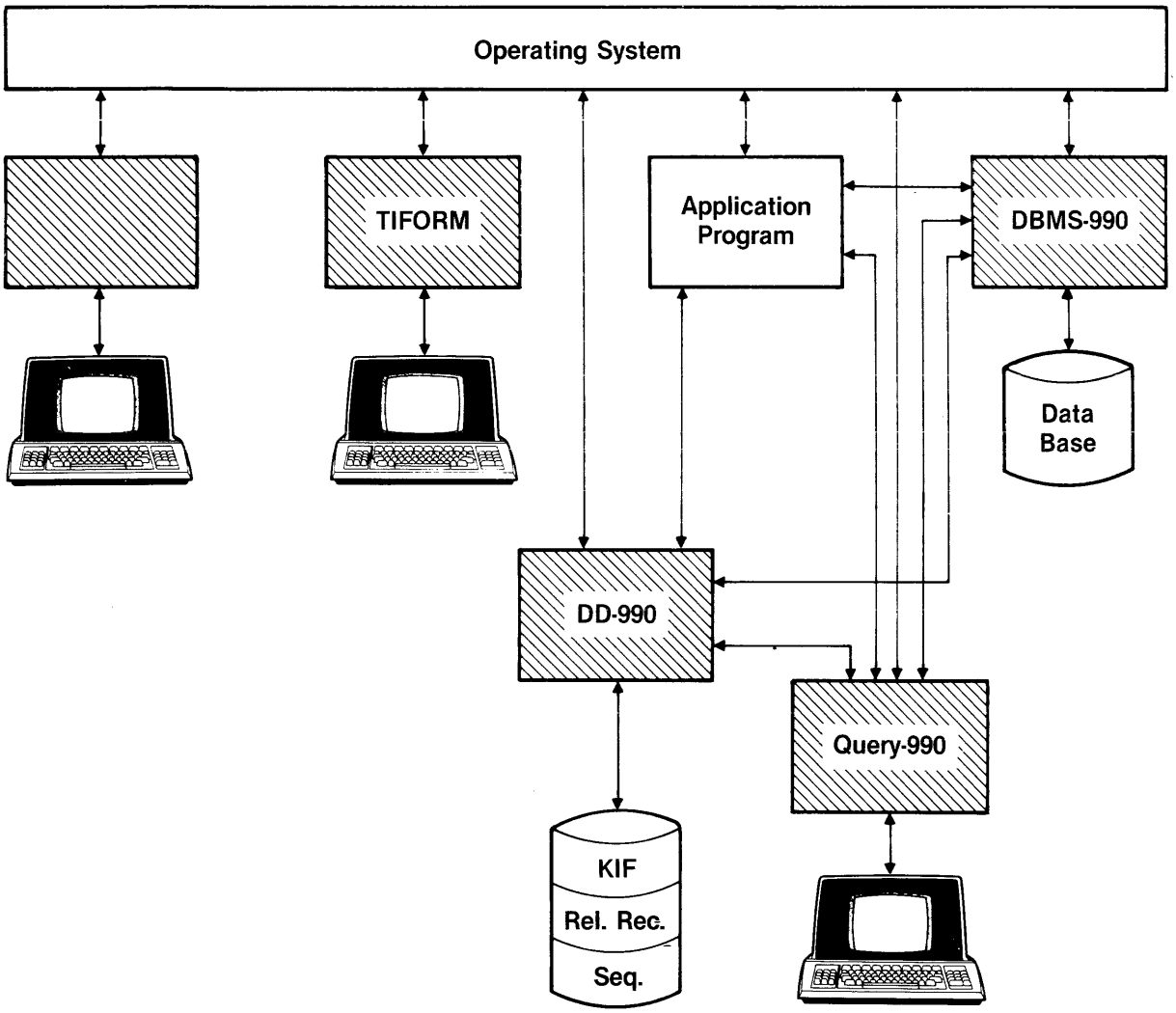
	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		xxx	xxx				xxx
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	xxx
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	xxx	xxx	xxx	xxx	*	xxx
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

* Language Comparison Matrix - DD-990

DD-990 is rated highly in ease of development, in the learning curve, and in the translator speed. It is easy to become familiar with various utilities. It executes very quickly, and it allows the expert user to develop complex applications. It is strong in business applications and data management and access. It is also has a great deal of potential, since there is currently work being done on the committee to produce a standard for data dictionaries, and one of the members of this committee works at Texas Instruments.

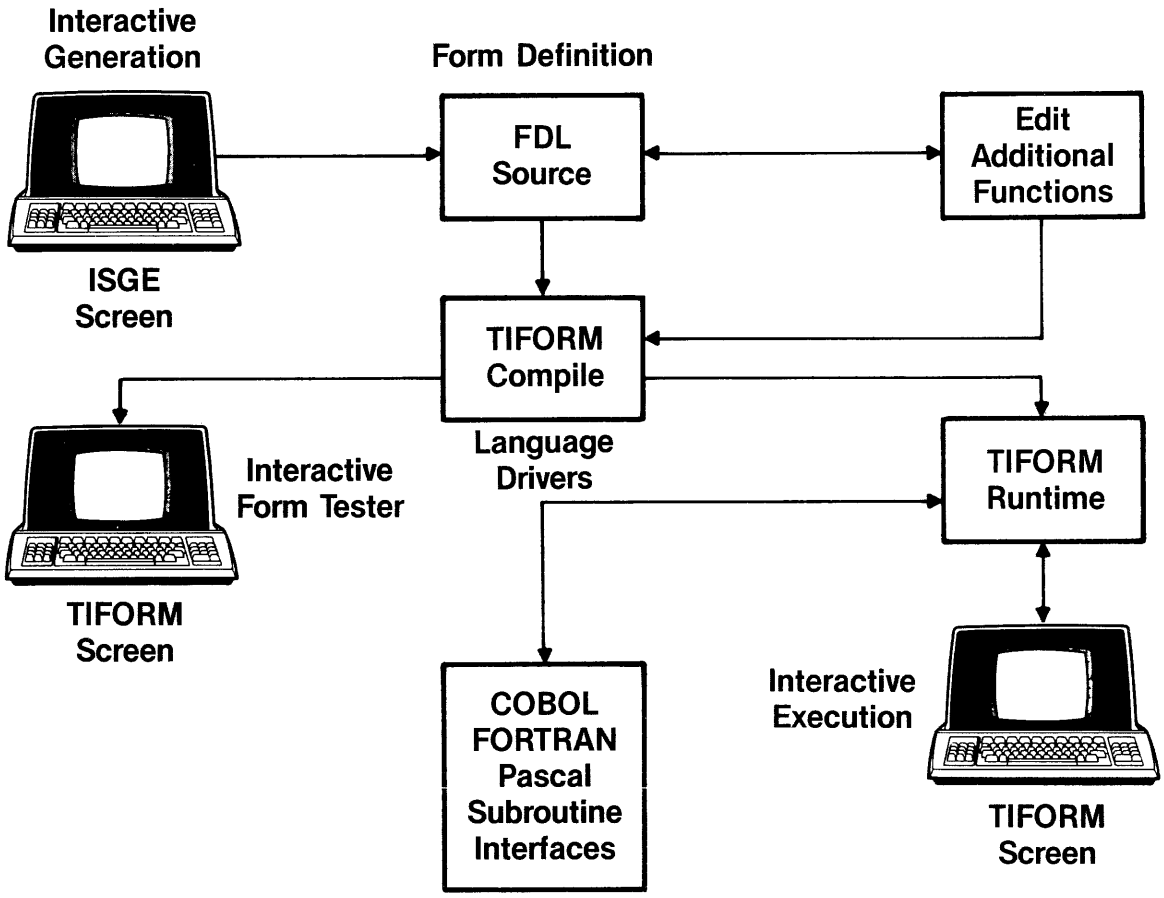
PRODUCTIVITY AIDS



* Productivity Aids - TIFORM

Getting away from data management products, our next product is TIFORM, the Texas Instruments screen management package.

TIFORM PROCESSES



* TIFORM Processes

This is a pictoral view of the components of TIFORM. Each of these components will be discussed in greater detail in the next section. The components include the Interactive Screen Generator and Editor, the Form Definition source, the Form Definition Compiler, the Form Tester, the High Level Language Interface Package, and the Form Executor.

COMPONENTS OF TIFORM

- FORM DESCRIPTION LANGUAGE (FDL)
  - Block structured language used to specify VDT screen layouts
- FORM DESCRIPTION LANGUAGE COMPILER (FDLC)
  - Converts FDL into a program file for the Form Executor
- INTERACTIVE SCREEN GENERATOR AND EDITOR (ISGE)
  - Allows user to design screen layouts on blank screen
- HIGH LEVEL LANGUAGE RUN-TIME INTERFACE PACKAGE (HLLIP)
  - COBOL — FORTRAN — PASCAL
- FORM EXECUTOR
  - Run time package that executes the commands issued by the application
- FORM TESTER
  - Allows user to view and test screen layouts without the application program

* Components of TIFORM

TIFORM is the Texas Instruments screen management package. It is designed as an interface for applications requiring input and output to a video display terminal. The components of TIFORM include the following. First is the Form Description Language, called the FDL. This is a file containing the specifications of the screen layouts written in a block structured language. Next is the Form Description Language Compiler, FDLC. This is a compiler that translates the FDL statements into a program file containing the commands for the Form Executor. The Form Executor is a run-time package, meaning that it is used at the time of execution. This package actually executes the comands as they are issued by the application. The next element of TIFORM needed is the High Level Language run-time Interface Package. This is the link between the application program and the Form Executor. The HLLIP is a set of subroutines called by the application. TIFORM has interfaces for three languages : COBOL, FORTRAN, and Pascal. A final element of TIFORM is the Interactive Screen Generator and Editor (ISGE). This utility allows a user to design screen layouts on a blank screen. In this manner, the user gets a better feel for where the fields will be. The ISGE utility will take the design specifications, build the FDL, and if desired, compile the FDL.

BENEFITS OF USING TIFORM

- ALLOWS A PROGRAMMER TO SEPARATE THE CODE THAT SPECIFIES SCREEN DEPENDENT INFORMATION FROM THE CODE THAT ACTUALLY MANIPULATES DATA RECEIVED FROM A VDT
  - Screen dependent information, ie. physical location on the screen (column and line number), is removed entirely from the application
    - ...eg. If a user has a 5000 line COBOL program accepting data from a VDT, and the user needs to move one input field 2 spaces to the left, he need only modify and recompile a 100 or 200 line TIFORM FDL, instead of the entire COBOL program
  - Terminal specifications are handled by TIFORM
    - ...eg. If the same user wants to transfer the application to a different type of terminal, the application need not be modified, only the TIFORM FDL
- PERFORMS A LARGE AMOUNT OF DATA VERIFICATION ON THE INTERACTIVELY SUBMITTED INFORMATION

* Benefits of Using TIFORM

There are several benefits of using TIFORM. One of the major benefits is that TIFORM separates the code that specifies screen dependent information from the code that actually manipulates the data recieved from the VDT. By screen dependent information, I mean physical location of a field on a screen, such as line and column number. This information is handled entirely in the FDL. For example, if a user has a five thousand line COBOL program accepting data from the screen, and a field must be moved two spaces to the left, the user doesn't have to do anything to the COBOL program. He need only modify and recompile a one- or two-hundred line FDL. In this manner, the user can cut the development time of an application by as much as thirty percent. Another part of the screen dependent information is the terminal type. This can also be handled entirely by TIFORM. For example, if the same user has developed his aplication on one type of terminal, and must move the application to another type of terminal, nothing need be done to any part of the application. TIFORM can be set to automatically poll the terminal for its type. TIFORM also provides a great deal of data verification. For example, the FDL can stipulate that a certain field may only accept numeric data, or that only members of a certain table can be entered for another field. This frees the application of many data checking duties.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

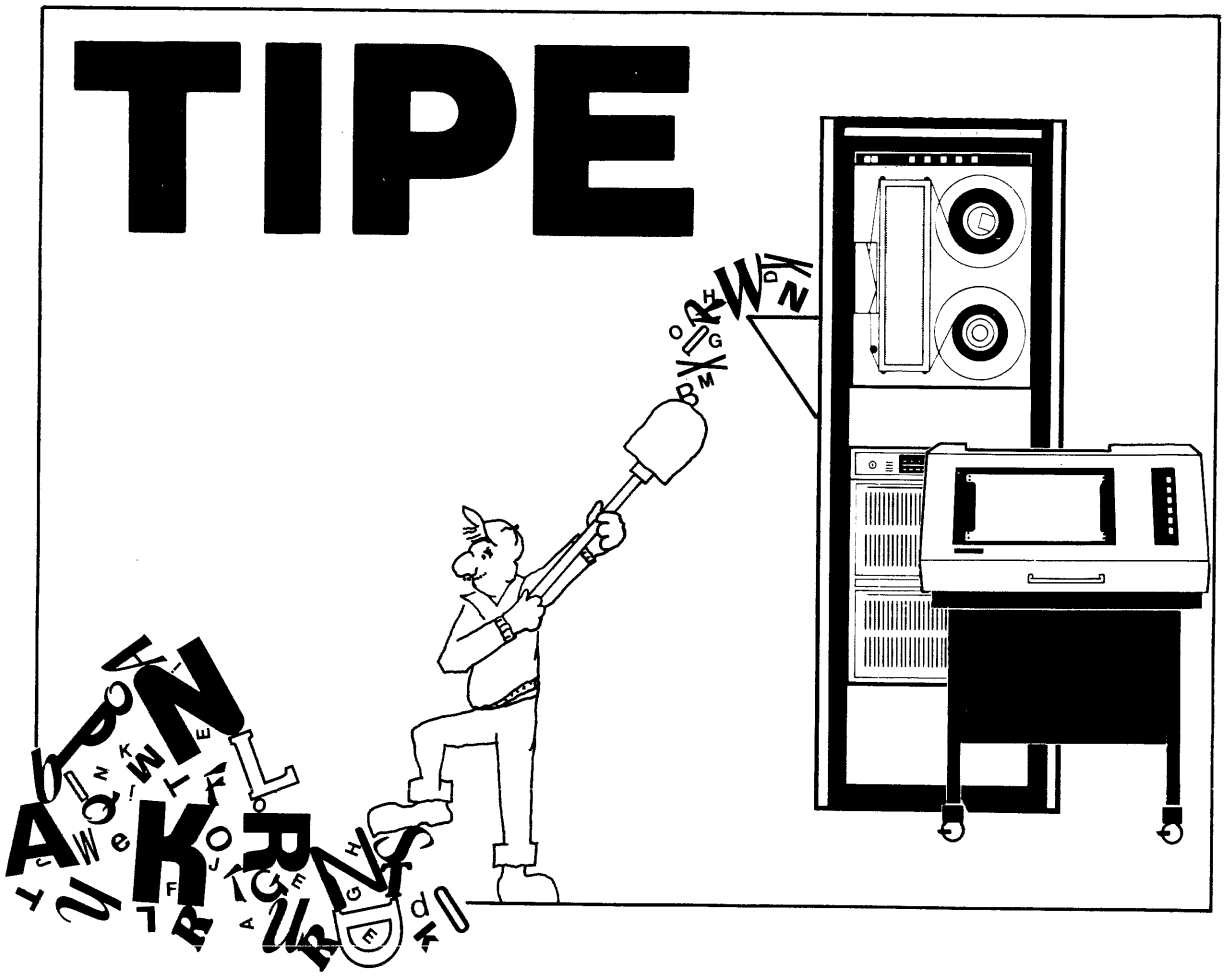
LANGUAGE COMPARISON MATRIX

	BASIC	C — (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		xxx	xxx				xxx
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	xxx
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	xxx	xxx	xxx	xxx	*	xxx
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

* Language Comparison Matrix - TIFORM

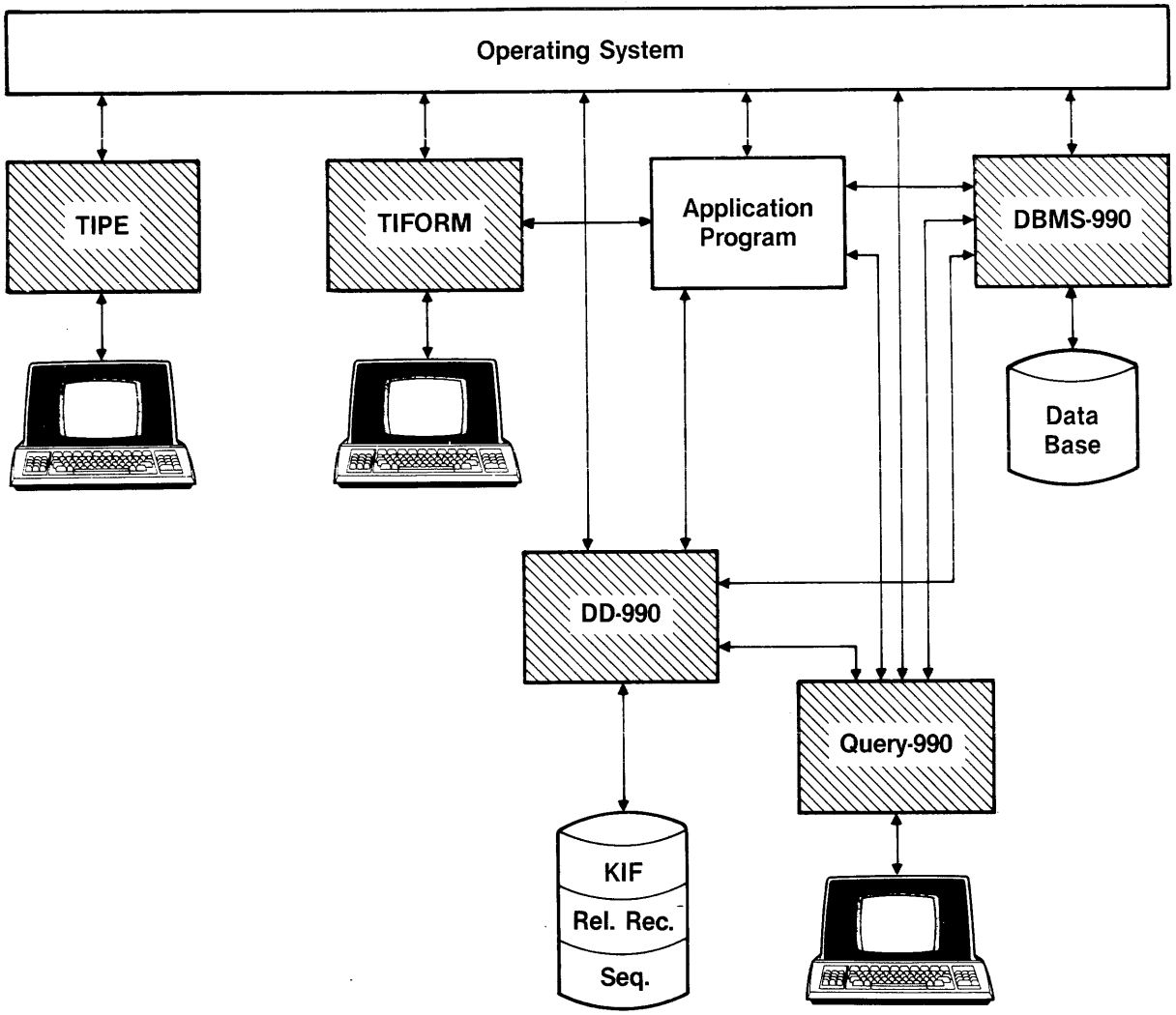
TIFORM is rated high in ease of development. It is not difficult to write complicated forms. TIFORM is also rated extremely high in screen manipulation. There is great potential for TIFORM, since Texas Instruments also has a member on the committee for a screen management standard.



* TIPE (illustration)

This is how the project leader of TIPE likes to envision our last product in the presentation. While it is certainly not the same as just shoveling letters into the computer, TIPE, the word processor for TI computers, comes quite close.

PRODUCTIVITY AIDS



* Productivity Aids - TIPE

TIPE currently does not interface directly with any of the other products covered today, but it is very important to the productivity of a company.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

TIPE

- **PROVIDES COST EFFECTIVE WAY TO ADD WORD PROCESSING TO 990 COMPUTERS**
- **EASY TO LEARN, EASY TO USE**
- **MENU AND PROMPTING ORIENTED**

* TIPE

TIPE is the Texas Instruments Page Editor. It is designed to be an add-on package to TI's operating systems. It is not meant to compete with stand alone word processors. However, it is a cost effective way to add word processing to Texas Instruments computers. TIPE provides a large subset of word processing features. These will be covered briefly later in the presentation. TIPE is easy to learn, and easy to use. A self instruction manual and an exercise manual are available when TIPE is purchased. TIPE is menu and prompt oriented and designed to interface easily with other Texas Instruments software and file strucutres.

COMPONENTS OF TIPE

- OPERATIONS
- UTILITIES
  - SCI utilities
  - Submenu utilities
- FUNCTIONS
- FORMATTING FEATURES

* Components of TIPE

There are nearly fifty different features to TIPE. Instead of trying to list and explain all of them, I will cover the four groups that best name similar features. The first group is the set of TIPE operations. TIPE operations are displayed on the main menu and represent major categories of the TIPE program. These operations are used to develop, revise, print and maintain TIPE documents. Next is the group of TIPE utilties. There are nine TIPE utilities. Seven are displayed on the utility submenu. The other two are only available through SCI. A TIPE utility does in general one of two things. The utility either provides information, such as displaying the status of a document in the print queue or displaying the contents of a directory, or the utility automatically performs some task, such as converting a document produced on an earlier version of TIPE to the newest version of TIPE. Next are the TIPE functions. These functions allow a user to manipulate text and move through a document. Examples of functions are commands that display the next or previous page of a document, insert text between existing lines, search thru text and replace a string of characters with another string of characters. Finally are the formatting features. The TIPE formatting features are used during a create or edit operation to format the text. This is done by embedding special codes (called ENTER codes) into the text. These codes denote functions which will occur when the document is printed. Examples of the formatting features are single and double underscore, single, double, triple and line and a half spacing, subscript and superscript, and boldface type. These and all the other features make TIPE a fine addition of word processing to TI computers. TIPE may also be integrated with other TI products at various levels. For example you may use Query-990 to generate data that may go into a form letter written with TIPE.

TI-MIX 1983
Techniques and Concepts for the Non-Technical User
Survey of 990 Languages

LANGUAGE COMPARISON MATRIX

	BASIC	C (1)	COBOL	FORTRAN	PASCAL	RPG	DATA DCT	DBMS	QUERY	SORT/MRG	TIFORM	TIPE
EASE OF DEVELOPMENT	*		+		+	+	+			+/2	+	+
LEARNING CURVE	*						+		+	+/2		+
TRANSLATOR SPEED		+		*	—		+	—				
APPLICATION SPEED	—	*	—	*	*		xxx	xxx				xxx
PORTABILITY	+	*	*	*	+/3		—	—	—	—	—	
MAINTAINABILITY	—	+	+	—	*	—	+		+		+	+
SCIENTIFIC/ENGINEERING	+			*		—						
SYSTEMS PROGRAMMING	—	*	—		+/4	—	—	—	—	—	—	xxx
BUSINESS APPLICATIONS	+		*	—		+	+	+	+	+	+	+
SCREEN MANIPULATION	+	—	+	—	—	—	xxx	xxx	xxx	xxx	*	xxx
DATA MGMT/ACCESS	+	—	+	—	—		*	*	*	—	—	—
POTENTIAL		*	+		+	—	+			+	+	+

NOTES:
1 – INCLUDED FOR REFERENCE ONLY.
2 – WITH NEW INTERFACE.
3 – WITHOUT TI EXTENSIONS.
4 – WITH TI EXTENSIONS.

* Language Comparison Matrix - TIPE

TIPE is rated high in ease of development due to the TIPE editor. It allows a user to develop complex documents using the TIPE features. It is also user friendly and therefore is rated highly in the learning curve category.

Mr. J.W. Woodridge
35775 Williamson Drive
Universal City, CA 91608

Dear Mr. Woodridge:

This example of a business letter was created and printed by the new version of the Texas Instruments Page Editor.

Some of the features of TIPE-990 are illustrated in this letter. naturally, a letter cannot tell you everything there is to know about our new version of this software package, but a Texas Instruments representative is available to answer any questions you have.

TIPE offers several features during creation/editing operations that make the job of keying text and formatting so easy for a typist. Some of the features include automatic word wrapping, block copy, move, reformat, delete and store operations, a free moving cursor, and horizontal scrolling to 240 characters. TIPE also offers a column copy, move, delete and erase function, in addition to automatic centering and a background paragraph assembly capability. Other editing features which TIPE provides include a Global Search and Replace capability and a Direct Page Access function which eliminates the need to perform scrolling to reach a page of text.

TIPE features that are available during printing include **bold printing**, automatic underscoring (either single or double), fully automatic repagination which includes widow/orphan functionality, variable line spacing (single space, space and a half, double and triple spacing), stored print instructions for each document, headers and footers, page numbers, subscripts/superscripts, a character overstrike capability, which allows special characters to be formed such as ≠, and even more.

TIPE-990 is designed with the operator in mind. There are a series of conversational messages to help the operator in the performance of specific functions. The variety of functions that can be performed demonstrate the powerful versatility of the package. TIPE-990 is easy to learn, and even easier to use!

Thank you for your interest in the TIPE word processing system.

Sincerely,

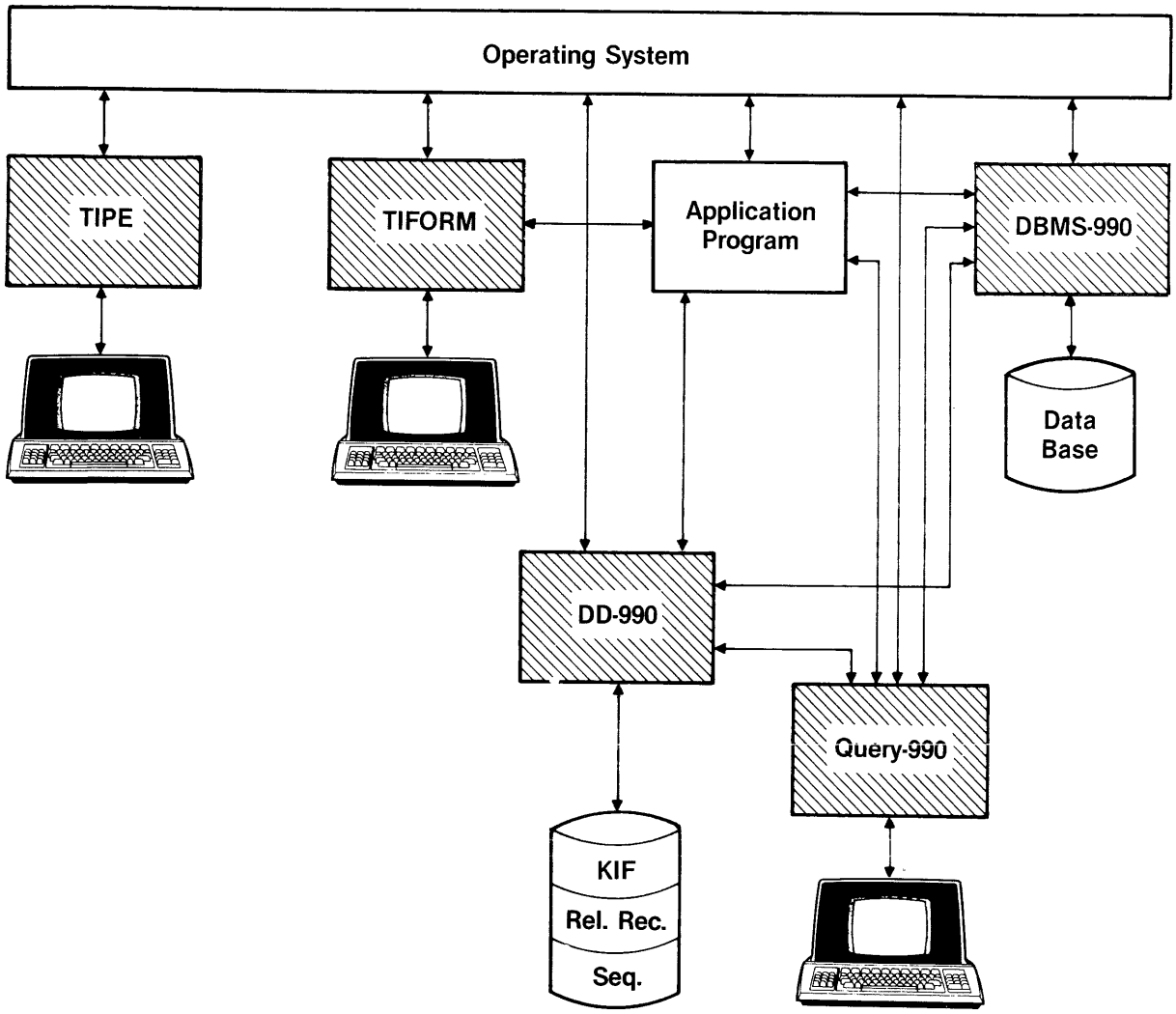
Kris Rogers

* TIPE example

This is an example of a form letter produced on TIPE. Some of the features are highlighted, for example boldface type and underscoring.

PRODUCTIVITY AIDS

SUMMARY

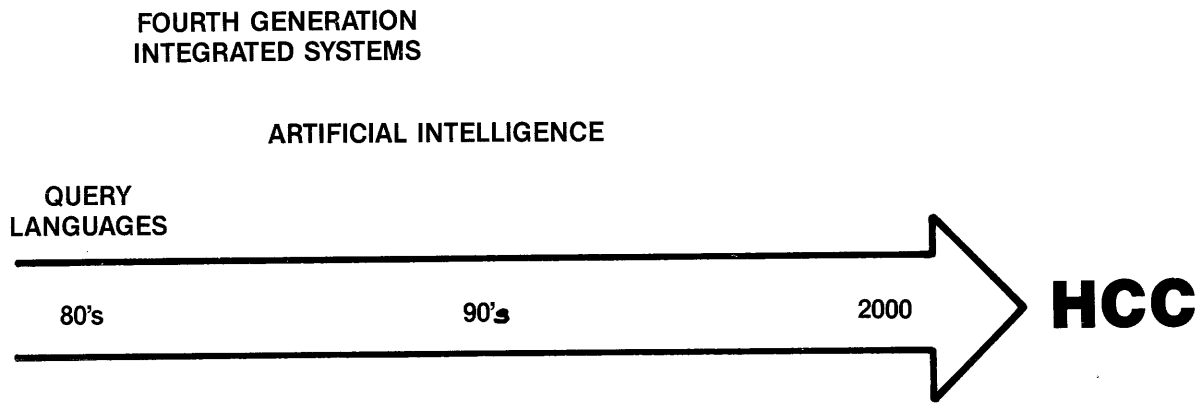


* Productivity Aids - Summary

Now, you have a good idea of the power and time saving potential of some of the tools available with the 990 computer line. If you have any further questions about any of these products, I suggest you attend the presentations based entirely on the particular product.

TI-MIX 1983
Techniques and Concepts for the Non Technical or New User
Survey of 990 Languages

CHRONOLOGY



CHRONOLOGY

You have heard a rather complete overview of where we are in Languages capability in 1983 with regard to standard, readily available software. The future looks very exciting and although "programming" will remain an intellectual exercise for an indefinite time, the tools keep getting better. For example we are already able to see how fourth generation languages, integrated systems and artificial intelligence techniques will reduce programming effort and help make systems more user friendly.

Computer hardware required to operate future software will be different from the off the shelf systems of 1983. It will be interesting to see if these future computers approach the HAL Class Capabilities already predicted for 2001.

990 HARDWARE AND SOFTWARE OVERVIEW AND COMPARISONS

A presentation for TI-MIX 1983
Techniques and Concepts for the
Nontechnical or New User Session

by

Bruce Barlow
Texas Instruments
April 1983
Austin, Texas

ABSTRACT

This presentation will consist of three parts aimed at familiarizing the novice or non-technical user with computers and operating systems in general and the 990 hardware and software in particular. The first session will deal with the fundamentals of computer hardware and the general characteristics of computer operating systems. The specific features of the various 990 computer architectures will also be discussed in this session. The second session will deal specifically with features of the DX10 operating system including memory structure and system files. Finally, the DNOS operating system will be discussed in a similar manner to DX10 and a comparison of DX10 and DNOS characteristics will be given.

GENERAL HARDWARE AND SOFTWARE OVERVIEW

Computers have undergone dramatic changes in size, speed, complexity and price in the last 20 to 30 years. Despite this evolution, the relationship of the computer hardware and the systems's software which supports it has remained fairly constant. Granted, there have been multitudes of computer hardware implementations and corresponding operating systems to support them but the primary characteristics of the basic computer have changed very little.

Fundamentally, computer hardware consists of three main components:

1. A central processing unit which may range from a giant mainframe to a tiny microprocessor.

2. A hierarchical storage system for programs and data which ranges from very high speed Random Access Memory (RAM) to disk storage to tape storage. Types of data storage vary dramatically in access methods, capacity, and price and these factors determine a particular storage medium's place in the hierarchy. Typically, the faster the access the higher the cost-per-bit of storage.

3. An interface to the outside world which can be as simple as a speed sensor on a microprocessor driven cruise control or as complex as a sophisticated color graphics Video Display Terminal (VDT).

An operating system also has certain fundamental characteristics which have remained constant over time. These are:

1. Operating systems provide the direct interface to external hardware devices such as VDTs, disk drives, tape drives and communication devices. That is, each user of a particular computer does not have to understand the hardware interface of a particular device in order to do i/o with that device. The operating system translates a higher level command from a user program into particular hardware interface commands to actually perform the desired function.

2. A benefit of providing a uniform access method to

hardware devices is that the operating system can also control the usage of these devices. It is not desirable, for instance, for multiple users to write to a specific device simultaneously. The operating system controls allocation and sharing of resources.

3. The operating system provides common services for user written programs which prevents each individual user from having to provide that feature in his own program. Perhaps the best example of this is support for "file" oriented access to a disk device. Even if a user has access to a physical disk, it is still desirable to have data he wishes to access logically grouped into "files" and not to be bothered with where that data actually resides on the disk. All general purpose operating systems provide some sort of file management service.

In summary, computer hardware provides for the storage of data and the execution of programs. An operating system is a program that executes on the computer hardware and acts as an intermediary between user programs and the actual hardware devices connected to the computer. Of course, this is a very simplistic view of a computer system but the concepts are the same for the vast majority of computer systems in the world today.

990 COMPUTER ARCHITECTURE

There are 3 basic CPU types in the 990 hardware line: the /10, /10A and /12. To a customer of Texas Instruments the primary difference in these computers is cost and speed but the underlying architectural characteristics are the same. These characteristics are:

1. A 16 bit address space - Essentially, this imposes a limit of 65536 (64K) bytes of program and data area that can be directly accessed by any program.

2. A segmented addressing system - The 64K program just mentioned can be partitioned in from 1 to 3 segments anywhere in physical memory. These segments can be of varying size but the total of the three cannot exceed 64K. This is usually of little concern to the average system user unless he is developing an extremely large application. A brief discussion of how operating system design is affected by this feature will follow later.

3. CRU and TILINE input/output devices - Certain devices with relatively slow access rates (line printers for instance) can have their i/o entirely controlled by a program running in the CPU. That is, each character must be transferred through a Communications Register Unit (CRU) by executing instructions in the CPU. Typically, each character i/o through this type of interface causes an interrupt to the CPU when processing is completed for that character . On the other hand high-speed devices, such has disk drives, would cause too great a burden on the CPU to be handled in this manner and therefore use a TILINE interface to transfer data. The data is transferred in large blocks which happens concurrently with CPU execution of other programs. An interrupt is generated to the CPU after the entire block is transmitted.

There are many other features that are common to the different 990 architectures but the ones listed above are central to understanding program and device limitations on the 990 systems.

990 OPERATING SYSTEMS ARCHITECTURE

The operating systems which run on the 990 architectures also have common features in their construction. Of course, the overriding constraint on this commonality is the hardware design itself and even though DX10 and DNOS have external functional differences there are many underlying similarites.

As was mentioned above, programs executing on the 990 are limited to 64K bytes of address space at any given point in time. This constraint applies to the operating system as well. However, both DX10 and DNOS require that more than 64K of code and data to be resident in memory at all times in order to guarantee good response time to user requests. This seeming paradox is not as severe as it might appear because there are no system operations which require all of the memory-resident code and data for its execution. Each operation the user requests usually requires a specific and predetermined portion of the memory resident code. For instance, a call to read from a VDT device would not requre access to the code that reads from disk files.

The segmentation feature of the 990 hardware allows the operating system to "map in" the particular portion of code required by a given user request. Remember that three address segments are provided which may total up to 64K. Figure 1 shows how this feature is used by the operating system to insure that the right code is mapped in. By controlling the segmentation feature of the 990, an operating system may be larger than 64K bytes yet still provide fast access time to required code and data. Even with this control there are still practical limitations on operating system sizes and the number of different i/o devices which can be supported.

Each i/o device in a particular confguration has its own unique characteristics and state information which must be maintained by the operating system. This information is kept in a memory-resident table referred to as a Physical Device Table (PDT) and there is one of these for every device "genned" into an operating system. In order to guarantee good response time to all i/o requests it is imperative that all PDT's be immediately accessible to the operating system; that is, they are all "mapped in" simultaneously. Given that the code which controls the device (the Device Service Routine or DSR) and the user's data buffer must also be mapped in at the same time it can be seen that there are a finite number of devices which can be supported in a 64K address space. Since the second and

third segment are of a fixed size the first segment cannot grow beyond a certain physical limit without exceeding the 64K address boundary.

THE DX10 OPERATING SYSTEM

DX10 has been the mainstay operating system on the 990 product line for several years. It has evolved into a stable, reliable, high-performance system and continues to receive excellent response from the user community. One of the chief reasons for its popularity is the System Command Interpreter (SCI) which executes "on top of" the operating system. SCI takes high level commands from a user and translates them into primitive operations which the operating system processes. In the strictest sense, SCI is not part of the "operating system" but is an integral part of the DX10 product.

DX10 is a general-purpose system suitable for a wide variety of commercial applications. It is a multi-user system and is capable of supporting a maximum of 15 to 20 terminals in a general application environment. The specific characteristics of the application itself (particularly the kinds of files and number of files accessed) will determine the maximum number of simultaneous users on a DX10 system. A closer look at the memory resident structure of DX10 will demonstrate why this is true.

The PDT structures for all of the devices on a given system reside in an area known as DXDATA (see Figure 2). The area which adjoins DXDATA is the System Table Area (STA). This is an area of memory from which structures to support file and device access are dynamically allocated. Structures exist as long as the file or device is being used and are deallocated when processing is complete. The next adjoining area contains System Routines required for almost all types of user operations. These three areas always comprise the first segment of the operating system map space.

Once a system is generated, the sizes of DXDATA, STA and the System Routines are fixed. The size of DXDATA is primarily determined by the number of devices the user has "genned" in. STA size is specified by the user during system genertaion. The System Routines area is a constant size. It can be seen that as the device count increases a corresponding decrease in the maximum amount of STA will have to take place. Similarly, if a large number of files are required by each application program the number of devices which can be physically generated is limited by the size of the STA needed to support those files.

In addition to its memory resident portion, DX10 requires a system disk to access less frequently used code and data. This disk resident portion of the system is contained on several "system files" which are briefly detailed in the following list. The actual file name as seen on the system disk is shown in parenthesis.

1. System Program File (.S$PROGA) System tasks and any tasks the user wishes to make memory resident are contained in this file.

2. System Image File (.S$IMAGES) This file contains the memory resident parts of DX10 that are created by the system generation process.

3. System Overlay File (.S$OVLYA) This file contains code which "overlays" (temporarily replaces) portions of system task areas when required for that tasks execution.

4. Roll File (.S$ROLLA) This file is used to temporarily save disk-resident task code which was loaded into memory but then that memory was required by a higher priority task.

5. System Crash File (.S$CRASH) This file is required to save an entire copy of memory if there is a system failure (crash). This file can then be examined to determine the cause of the failure.

6. System Loader File (.S$LOADER) This file contains the code which loads the DX10 operating system from the System Image File.

7. Diagnostic File (.SDIAG) This file exists on every disk pack and is used by diagnostic programs to isolate potential prolems on the disk without affecting user data.

8. SCI Procedure Directory (.S$PROC) This is a directory of files and each file is an SCI command which the user may invoke. All the system supplied SCI commands reside here and user-defined commands may be added as well.

There are other files required by DX10 or SCI and these are described in detail in the DX10 system documentation.

THE DNOS OPERATING SYSTEM

DNOS, like DX10, is a general purpose operating system capable of supporting a wide variety of commercial applications. In addition, DNOS provides direct support for SNA and X.25 networking software which are not directly supported on DX10. Other specific differences between DX10 and DNOS will be discussed in the next section.

DNOS is targeted toward the end-user who requires a larger terminal capacity as well as sophisticated communications capabilites. The maximum terminal capacity for DNOS in a general application environment is 30 to 40 terminals. As with DX10, the maximum capacity depends on the particular applications being run on the system. The presence or absence of communications support also affects this capacity.

The primary difference which allows DNOS to support a greater number of terminals in similar environments is the way file structures are handled. Remember that on DX10 the file access structures are allocated in System Table Area. In DNOS a separate area is allocated solely for the purpose of containing these file structures. This allows the first segment of the address space to contain more PDT structures since it does not have to compete with file structures for space. Other structures such as those which support task execution were also removed from the STA to allow more room for device structures. The tradeoff for this type of approach is that more total space is required to contain these separate and dedicated areas for file and task structures. Also, more mapping changes are required to access these structures which may have a negative effect on response time but every effort has been taken to keep map changes to a minimum.

DNOS requires certain disk files to be present on the system disk as does DX10. A list of some of the major files required and a brief description is given below:

1. The Kernel Program File (.XXXXXXXX) This is the program file containing the tasks, procedures and overlays of DNOS. The name of this file is supplied by the user at system generation time. Notice that there is no separate overlay file on DNOS.

2. The Shared Program File (.S$SHARED) This is the program file where users may place memory resident tasks or shared procedures.

3. The Utilities Program File (.S$UTIL) This file contains tasks and procedures for system utility programs. These are generally invoked by SCI commands.

4. The Swap File (.S$ROLLD.S$ROLLA) This is the file where task images are temporarily placed when their memory is required by a higher priority task.

5. The SCI Procedure Directory (.S$CMDS) The directory of files which contains all the standard SCI commands.

6. The User ID File (.S$USER) The directory which contains user id's defined for a given system and their synonym files.

7. The Initializaion Batch Stream (.S$ISBTCH) This file is a batch stream which is executed as soon as the system is loaded. Any user initialization procedures should be placed in this file.

8. The Messages Directories (.S$EXPMSG & .S$MSG) The message directories contain files of expanded error text to assist in explaining error codes and to provide suggested corrective actions for problems encountered.

The .S$CRASH and .S$DIAG files are also required on DNOS and perform the same function as they do on DX10.

There are other memory and disk structure differences between DNOS and DX10 but a comprehensive comparison of all of them is beyond the scope of this presentation.

DX10 and DNOS COMPARISON

Following is a comparison table of DX10 and DNOS. Characteristics are not evaluated in great detail but rather given as suggested reference points for system evaluation.

CHARACTERISTIC	DX10	DNOS
Terminal capacity (application dependent)	15-20	30-40
Minimum memory required	128K	256K
Synonym space	864 bytes	12288 bytes
Resource allocation	Terminal oriented	Job oriented
Full language support	Yes	Yes except FORTRAN 66
Communication support	3270,3780	3270,3780 SNA,X.25
Task-to-task communication facility	ITC	IPC
Accounting	No	Yes
Configuration utilities	Sysgen	Sysgen SCU
Print spooling	No (LP$x)	Yes
System Operator facility	No	Yes

DX10 MAPPING OVERVIEW

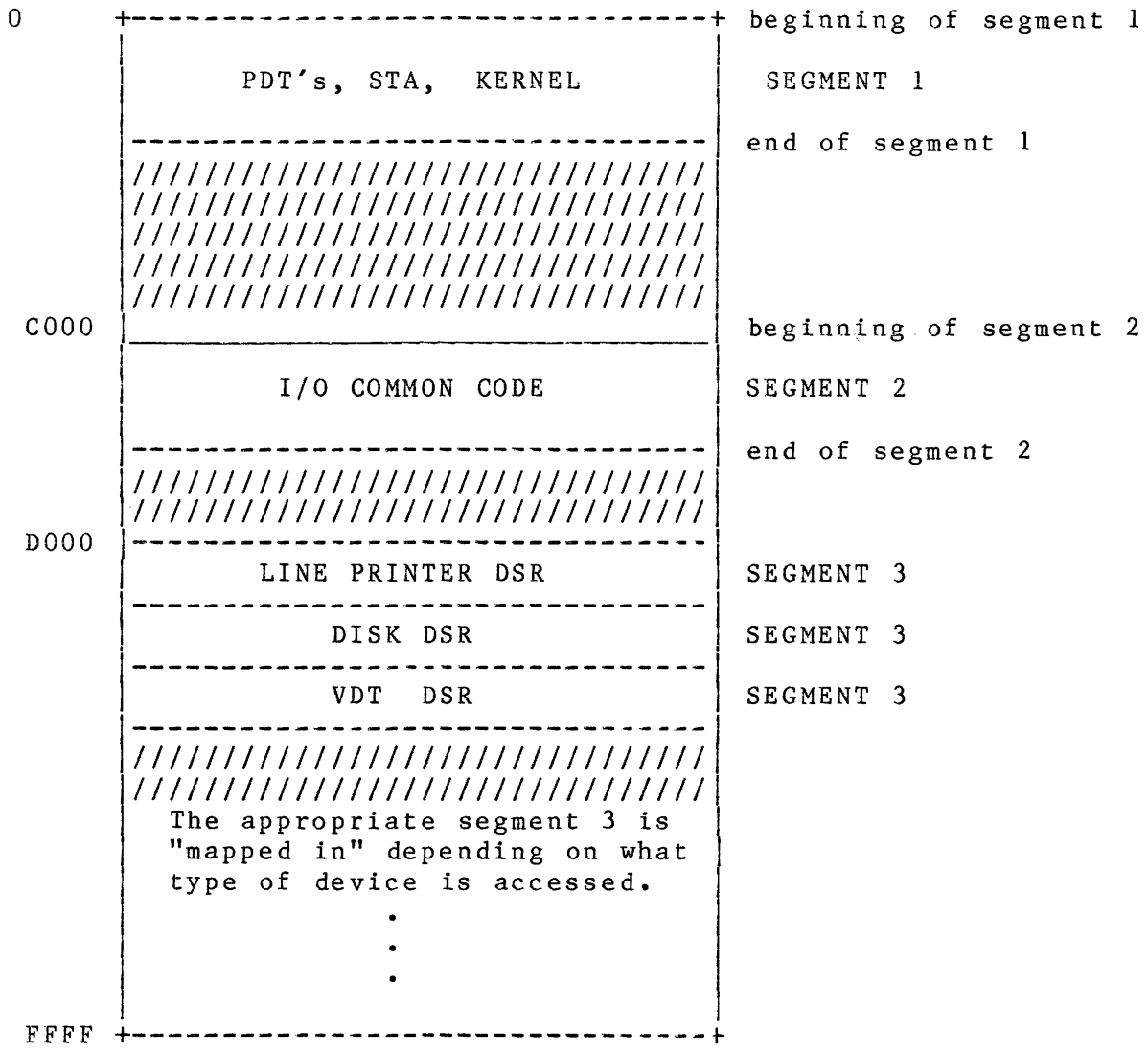


FIGURE 1

EXPANDED VIEW OF SEGMENT 1

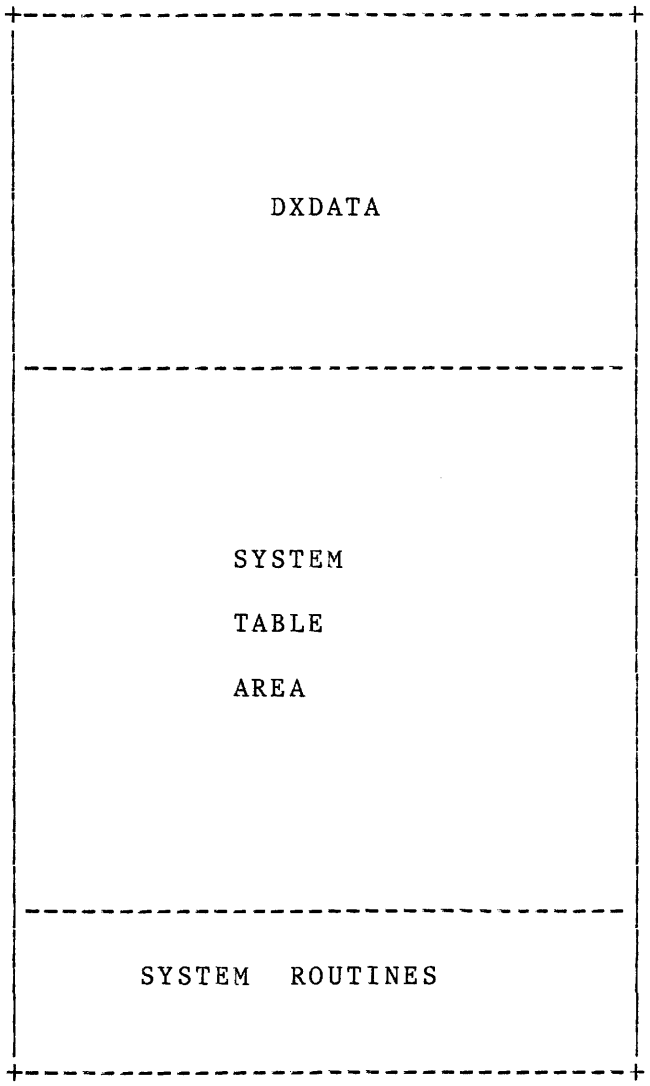


FIGURE 2

UCSD p-System Overview

Techniques and Concepts for the Nontechnical or New User

David Burckhartt
Texas Instruments, Inc.
P.O. Box 2909, M/S 2201
Austin, Texas 78769

This presentation will attempt to answer 2 questions. The first is what is the UCSD p-System and what makes it unique from other microcomputer software systems that are on the market? The second is how do I decide if the p-System is right for my usage and if it is right, how do I get started?

What is the p-System? It is a complete, portable microcomputer software development and execution system environment. The next few paragraphs will explain this definition in more detail.

(Foil 1): The p-System was developed at the Institute for Information Systems at the University of California at San Diego (UCSD) in 1977. It was first implemented on large main frame computers and then migrated down to the microcomputer with enhancements for the interactive environment. Being a non-profit organization, the university was not allowed to market the product at a profit. In order to insure the product would be utilized, a third party was found to acquire distribution rights. SofTech Microsystems, a part of SofTech acquired distribution rights. They now develop, support, and market the p-System.

The p-System was designed to be transportable. That means that it would function on most major microprocessors such as a 9900, 68000, 8088, etc. The P in p-System stands for pseudo system...not Pascal system. Each program is written in a high level language and compiled to pseudo code which is transportable. The system derives its name from the utilization of this pseudo code. Currently, Softech is claiming over 100,000 end users.

(Foil 2): The p-System provides the capability to write a program once that will run on all computers for which a p-code interpreter exists. Hardware independence is thus maximized since the same code will run on different sets of hardware. Development can then be performed on any machine which has a p-System and the resulting code run on any other machine that has a p-System. In addition, the various p-System languages: Pascal, FORTRAN, and BASIC are integrated. Programs may be written in any combination of these languages.

What is p-code? It is the native language or machine code for the p-System. It is an interpretive instruction set or a software computer. In other words, the p-codes are the same on each machine. The difference from machine to machine is the p-code interpreter. Since the p-codes are interpretive, execution

speed is traded off for transportability. In general, interpretive code tends to be smaller than native machine code...thus, larger programs will fit in a smaller address space. The concept of portability utilizing an interpretive instruction set is used in the TI FORTRAN and COBOL compilers.

(Foil 3): What is portability? Creating software that will execute -- today and tomorrow -- on hardware systems other than the system where it was developed. In other words, I want to develop on any system today and continue to run the same software on other piece of hardware available today or tomorrow.

(Foil 4): Several approaches are taken to portability. Since languages such as COBOL are defined by an ANSI Standard, people talk about the portability of programs in terms of language standards. At a minimum in general, the programs must be recompiled and there may be changes required in things like screen management.

(Foil 5): In fact, there are several problems with the language approach to portability. The standard does not define all the elements of the language. Some elements are undefined or are implementor defined. In particular with COBOL, the assign statement has portions that are implementor defined and the internal precision of intermediate calculations in the COMPUTE statement is implementor defined. There are differences in the environment. For example one system may support 8 secondary keys on keyed-indexed files while another might support 14. Screen management is not defined by ANSI Standard so each implementor has defined a different way of implementing this feature. Thus, portability is not as good as one might wish.

(Foil 6): There has been a lot of press recently about standard operating systems. This occurs when the functionality of the operating system has been implemented on a number of different machines. What is ignored in this case is the transportability of the user program. If the program is in a high level language such as COBOL, the user may be able to move with minimal effort. In general, it is hard to provide all the same functionality of any operating system across a series of different hardware configurations.

(Foil 7): Several problems occur. There may be different sizes of memory, different memory mapping requirements and different capabilities of perpherials and the mircoprocessor itself. For example, in the CP/M world, there is CP/M for the 8080 based machines, CP/M-86 for the 8086/8088 based machines, and CP/M-68K for the 68000 based machines. Why three different names? They are physically different. In the case of the p-System, it is just the p-System,no matter what machine it is

executing on.

(Foil 8): The p-System takes the environment approach to portability. Since the entire system is interpretive, the OS, Host Interface, and the Program all are portable. This means that all the compilers and utilities are also portable.

(Foil 9): Not only that, but this portability extends across any microprocessor. The hardware is not the limiting factor.

(Foil 10): How is this accomplished? The operating system, compilers, utilities, and applications are all written in p-code. The p-code is then translated by the interpreter written for the particular target machine. The final piece of the system is the Basic I/O System or BIOS that is the low level code to interface to the particular devices on a particular machine. As you can see from the diagram, the running program is isolated from the architecture of the particular machine by the interpreter and BIOS.

(Foil 11): In fact, the applications developer can write his program in any p-System supported language, use the p-System compiler(s) to create p-code, and it will execute on any host equipped with a p-System without change.

(Foil 12): Furthermore, a p-code object program can be moved from one machine to another and executed without change if the disk recording standards are the same or communications capability is available -- even if one CPU is a Z80 and the other is a 9900, 8088, etc. Its true. Too good to be true you say? There are some restrictions... one must program for transportability. If a program has taken advantage of some hardware such as function keys on the keyboard, then the program may not port without change. Most developers in the p-System understand the differences that different sets of hardware may provide and utilize methods parameterize their applications when moving between hardware systems. Even so, moving a p-System program from one machine to another is easier than in any other portability scheme.

(Foil 13): The next question is always what about applications? There are software solutions available on the p-System for most application areas. This chart shows a comparison in several catagories. A p-System Application Catalog is available from SofTech Microsystems that lists the various applications available and where to obtain them.

(Foil 14): This foil summarizes why the p-System is unique in the market. Key points are that the environment is

transported...not just programs. Both source and object programs are transportable. Disk formats are predefined. There are few implementor defined elements and the p-System is adaptable to any machine architecture. In any case...assembly code is probably not transportable so the user should write his application in a higher level language.

How does one decide if the p-System is right for their application and environment? I believe that this decision needs to be made based on the company strategy for selling and using computers. If the user is content to sell only IBM or TI Professional Computer class products, then the choice would be MS-DOS and one of the MS-DOS language products. If the user wants the application to run on many different types of microprocessors, then the choice is the p-System.

(Foil 15): If the corporate strategy is to sell machines utilizing various microprocessors such as Apple machines with the 6502, and Radio Shack machines with the Z80 or 68000, or the IBM or TI Professional Computer with an 8088 and desires to run the same software base on each of these machines, then the p-System is the best system to utilize. If the strategy is to sell ONLY TI machines from the Business System 200 through the Business System 800, then the best plan is to utilize the DX10 or DNOS systems and their associated languages and productivity aids.

(Foil 16): While the p-System is supported on all TI products from the 99/4A Home Computer through the Business System 800, it is most efficient when run on a single user microprocessor type system. On the larger TI products, it is supported as a task running under DX10. While this allows the user to execute p-System applications in the DX10 environment, the overhead of DX10 plus the design of the p-System does not make this a viable multiple terminal environment when the number of terminals becomes large. TI designed operating systems and languages are more efficient in the multi-terminal DX10 style world. The DX10 p-System allows the user to run p-System applications in concert with his DX10 applications. It is not intended as the primary use of the system.

(Foil 17): Texas Instruments has chosen the p-System to be utilized on many machines within the company. Some of the primary reasons are listed on this foil. In particular, prevention of software obsolescence by new hardware offerings and the ability to select hardware based on the suitability to the task rather than the software available to run on that hardware are key reasons.

In summary, the p-System has the strongest transportability story of any system on the market. To obtain that

transportability, the user trades off performance although tools are available to increase performance if that becomes a problems. The decision to "go p-System" is a strategic decision that must be made based on the marketing plans of the company or their planned utilization of computers within the company. As with any system, the p-System has its limitations and quirks, but many large software houses have seen the wisdom of utilizing this system in the implementation of their products. Most manufacturers, DEC, IBM, HP, Texas Instruments, and others either provide the p-System as part of their product line or a third party has provided the p-System for a particular machine. The utilization of the system and its pervasiveness are increasing each day. It certainly is and will remain a viable contender in microcomputer systems arena.

Introduction to SCI for the Nontechnical or New User

presented in the

Techniques and Concepts for the Nontechnical or New User Session

by

Gary Imken

Texas Instruments Austin, Texas

April 6, 1983

This presentation introduces the System Command Interpreter (SCI) to nontechnical and new users. The first part of this presentation describes SCI in general terms and presents a scenario of a typical session which will be of interest to both nontechnical and new users. The latter part of this presentation explains more about how SCI works and will be of interest primarily to new users with some technical expertise. Specific details of SCI's operation can be found in the DX10 and DNOS manuals and are not presented here in order to focus on the main concepts.

WHAT IS SCI?

SCI can be described in several ways. The most basic definition of SCI is that it is the interface between the user and the DX10 or DNOS operating system. SCI is that part of the operating system which prompts the user to determine what actions he would like to perform.

At a more detailed level, SCI can be described as a compact, parameter-collecting program which invokes tasks using a standard interface. By design, SCI is not intended to perform numerous operations. Instead, the intent is to keep SCI small and fast so that user requests can be quickly passed on to the operating system with a minimum of overhead. In its operation, SCI will aid the user in his selection of a command to execute by displaying helpful information on the terminal to guide him. Once a command has been selected, SCI will optionally prompt the user for additional information which is needed to perform the command. For example, if the user enters the command to print a file, SCI will prompt the user to determine the file name and the name of the destination printer. Finally, SCI provides a standard interface to operating system tasks and user written tasks which simplifies the incorporation of new programs written by the user into the operating system.

Structurally, SCI can be described as an interpreter for a small set of primitive commands. These primitive commands control the actions of SCI by determining the menus of helpful information displayed to the user, by providing a description of the syntax for user inputs, and by manipulating these user inputs before they are passed to the appropriate task. SCI's primitive commands can be collected together into files of commands called procedures, or procs, which are also executed by SCI. The DX10 and DNOS operating systems use these procs to implement the command choices that are available to the user. For example, the print file command mentioned previously is actually a proc consisting of primitive commands.

SCENARIO OF A TYPICAL SESSION FOR A NEW USER

When a user enters the appropriate key sequence at a terminal to invoke DX10 SCI, the following information will be displayed on the screen (DNOS has a similar display):

```
SYSTEM COMMAND INTERPRETER - PLEASE LOG IN

                                    USER  ID:
                                    PASSCODE:
```

This display requests the entry of the USER ID and PASSCODE of the person who is logging onto the system. The USER ID is typically assigned by some system coordinator to all those individuals who will be using the system. The PASSCODE is optionally chosen by the user to prevent someone else from using his USER ID. Most systems usually require the entry of this information, but the DX10 and DNOS systems do permit this display to be eliminated if so desired.

After the entry of the USER ID and PASSCODE (or immediately if the USER ID and PASSCODE are not desired), the following display, or menu, will appear on a standard DX10 system:

```
T E X A S  I N S T R U M E N T S
D X 1 0  S Y S T E M  3 . 5 . 1

SELECT ONE OF THE FOLLOWING COMMAND GROUPS

        /DEV   - DEVICE OPERATIONS
        /FILE  - FILE OPERATIONS
        /PDEV  - PROGRAM DEVELOPMENT
        /SMAIN - DX10 MAINTENANCE
        /SOP   - DX10 OPERATION

[]
```

The box symbol ([]) is used by SCI in conjunction with the cursor to indicate the input line for command entry. The command groups shown on the menu - /DEV, /FILE, /PDEV, /SMAIN, and /SOP - indicate general categories of DX10 commands and are intended to aid the user who needs help locating a specific command name. Both the box symbol and the standard menu display are modifiable by the user. For example, a user might wish to display his company's name and a set of command groups different from those shown in this display. If the user enters one of these command group names such as /FILE, a more detailed submenu will be displayed as follows:

```
FILE OPERATIONS

        /DBMS  - DATA BASE MANAGEMENT SYSTEM
        /DIR   - DIRECTORY COMMANDS
        /EDIT  - TEXT EDIT COMMANDS
        /FILEC - FILE COMMANDS
        /LUNO  - LUNO COMMANDS
        /STAT  - STATUS COMMANDS

[]
```

This menu is similar to the previous menu except that the categories are divisions of file operations. If the user were interested in text editor commands, he would enter /EDIT on the command line and the following menu would be displayed:

```
TEXT EDIT COMMANDS

  CL  - COPY LINES           IF  - INSERT FILE
  DL  - DELETE LINES         MHR - MODIFY HORIZONAL ROLL VALUE
  ML  - MOVE LINES           MR  - MODIFY ROLL VALUE
  SL  - SHOW LINE            MRM - MODIFY RIGHT MARGIN
  SVL - SAVE LINES TO FILE   MT  - MODIFY TAB SETTINGS

  DS  - DELETE STRING        QE  - QUIT EDITOR
  FS  - FIND STRING          RE  - RECOVER EDIT
  RS  - REPLACE STRING       XE  - INITIATE TEXT EDITOR
                             XES - TEXT EDITOR WITH SCALING
```

[]

This menu is different from the preceding menus in that it contains actual commands rather than command groups. Actual command names do not begin with a slash (/) as the menu displays do. (As mentioned earlier, each of the command names shown here and in all DX10 and DNOS menus are actually procs.) A typical command that might be selected from this menu is the MRM command. When MRM is entered the following display appears:

```
MODIFY RIGHT MARGIN

   RIGHT MARGIN POSITION: 80
```

This display shows the full name of the command entered and displays a prompt which requests input from the user. In this example, the prompt has an initial value of 80 which is a frequently used response. If this is not an acceptable value to the user, he can overtype it with the desired value. Otherwise he can accept it as is.

As another example of a DX10 command, the user could follow a different path through the SCI menu structure and arrive at the print file command which appears as follows:

```
PRINT FILE

        FILE PATHNAME(S):
            ANSI FORMAT?: NO
          LISTING DEVICE: LP01
  DELETE AFTER PRINTING?: NO
    NUMBER OF LINES/PAGE:
```

This command prompts the user for five items of information, some of which have initial values displayed. The first prompt asks the user for one or more file pathnames which are to be printed. A list of items can be entered for this prompt, but not for any of the other prompts shown. The second and fourth prompts

must be answered with a Yes or No response. SCI will only accept responses that begin with Y or N for these prompts. The third prompt, LISTING DEVICE, also has an initial value, but it is slightly different from the second prompt. In this case, the initial value will be the response that was entered on the previous invocation of Print File. For example, the first time a new user invokes Print File, this prompt will have no initial value. If the user enters LP01 as his response, then the next time he uses Print File, LP01 will appear as the initial value. If the user enters LP03 as his response, then the next time he invokes Print File, LP03 will appear as the initial value. The fifth prompt, NUMBER OF LINES/PAGE, does not have an initial value, but it is different from the other prompts in that the user may enter a null response for it. If a null response is entered, SCI will provide an appropriate default value for the prompt.

As a user gains familiarity with the DX10 or DNOS system, he will use the menu structure less frequently since he will know the command names. SCI does not restrict the user to entering only those responses which appear on the current menu. The menus are only intended as a source of helpful information, so the user may enter any valid command name regardless of which menu is currently displayed.

The implementation of DX10 and DNOS commands as procs offers several advantages: (1) Procs are easily modifiable. To add or change a command, all the user has to do is edit a file. No compilations are needed, no reconfiguration is needed, and no reloading of the operating system is necessary. This greatly simplifies the job of the user who wants to customize the standard system by adding his own commands and perhaps modifying some of the existing DX10 or DNOS commands. (2) Procs can invoke other procs as well as primitives. This allows the user to construct his high level commmands in a more modular fashion. For example, he may write one proc which performs a frequently needed function and is invoked by several other procs. (3) Procs make it easy for the user to package a customized product for transfer to other systems. Typically the system dependencies can be isolated in the procs so that the programs invoked by the procs can be written in a more system independent fashion. (4) Procs can be restricted so that certain procs can only be executed by sufficiently privileged users. Thus those system functions which require a more sophisticated user can be isolated from the less sophisticated user.

COMMAND PRIMITIVES IN THE SCI LANGUAGE

As previously mentioned, SCI executes a set of primitive commands that control the displays that the user sees on the terminal and that manipulate the user's inputs. This section gives a brief description of these primitives without going into syntactic details in order to show the capabilities that are available through the primitive commands:

.PROC and .EOP - These primitives work as a pair to define the beginning and end of an SCI procedure when a procedure is installed in the DX10 or DNOS operating system.

.BID, .QBID, and .DBID - These primitives are used to invoke a task from SCI. The .BID primitive suspends SCI until the task is complete, .QBID lets the task and SCI execute concurrently, and .DBID is used to debug a task from SCI.

.DATA and .EOD - These primitives work as a pair to define an in-line data file which is to be copied to a file on disk. This file may contain dynamic information such as the responses entered by the user after prompting by SCI and other user specific data.

.EVAL - The .EVAL primitive is used to evaluate arithmetic expressions. For example, .EVAL could be used to add the results of two or more numeric values entered by the user in response to SCI prompts.

.EXIT - This primitive is used to exit from an SCI procedure. For example, if an SCI procedure named A invoked a procedure named B, then a .EXIT command in B would cause SCI to resume execution in A at the point at which A had been left.

.IF, .ELSE, .ENDIF - These primitives are used for conditional execution in the same manner as high level languages which support these constructs. The .IF primitive checks a condition and if true executes the following commands until a .ELSE or .ENDIF is encountered. A false condition causes execution to begin following the .ELSE if it is present, or following the .ENDIF if .ELSE is not present.

.LOOP, .UNTIL, .WHILE, .REPEAT - These primitives are used for repetitive execution of commands. The commands between the .LOOP and .REPEAT are executed until the condition specified in the .UNTIL is true or while the condition specified in the .WHILE is true. The .EVAL primitive is sometimes used in conjunction with such loops to specify a loop counter.

.MENU - This command controls which menu, if any, is the next one to be displayed to the user.

.OPTION - The .OPTION primitive allows the user to select certain options affecting the operation of SCI. For example, .OPTION can be used to allow both lower and upper case input to SCI and to control the default main menu that SCI displays to the user.

.PROMPT - This command causes SCI to prompt the user for additional information. This is often used in situations where the set of prompts that are to be displayed to the user are dependent on values entered earlier.

.SHOW - This primitive is used to display a file to the user. SCI provides scrolling and positioning commands to use in conjunction with the display.

.SPLIT - This primitive splits off the first item from a list of items. For example, the list of file pathnames specified in the Print File command described previously are separated using the .SPLIT primitive before the task that does the printing is invoked.

.STOP - The .STOP primitive is used to terminate the execution of SCI and log the user off the terminal.

.SVC - This primitive is reserved for operating system use.

.SYN - The .SYN primitive is used to define synonyms to SCI. Synonyms are typically short names that can be used in the place of longer names. For example, a user might wish to assign a frequently used file name consisting of many characters as the value of a short synonym name. Synonyms can also be thought of as the variable names which appear in higher level programming languages, i.e. as names for which there is an associated value.

.USE - The .USE primitive allows the user to specify which directories are searched to find the command names he enters. By default, SCI will search a standard directory, but the user may specify other directories to be searched with the .USE command.

AN EXAMPLE PROC

The following proc is presented to illustrate how a typical SCI proc would appear. As in the preceding discussion, no attempt will be made to go into any syntactic details or into very many specifics:

```
EX (EXAMPLE PROC)=2,                    ! PROC NAME AND PRIVILEGE LEVEL
ENTER NAME(S) = (NAME)                  ! PROMPT FOR USER INPUT
.SYN L = "(&ENTER NAMES(S))"            ! SAVE PROMPT IN SYNONYM L
.LOOP                                   ! BEGIN A LOOP
  .SPLIT LIST="@L",FIRST=F,REST=L       ! EXTRACT NEXT ITEM FROM LIST
  .BID TASK=DISPLAY, PARMS=(@F)         ! PASS ITEM TO DISPLAY TASK
  .WHILE "@L", NE, L                    ! CHECK FOR END OF LIST
.REPEAT                                 ! REPEAT THE LOOP
*--------------------------------
.IF @$$UI, NE, SYS000                   ! CHECK FOR USER ID SYS000
  .DATA .LOG, EXTEND=YES, SUB=YES       ! APPEND MESSAGE TO LOG FILE
   - USER @$$UI EXECUTED PROC EX -
  .EOD
.ENDIF                                  ! END OF CHECK
```

The first two lines of the proc EX contain the proc name, the privilege level of the users who can execute it, and the prompt ENTER NAME(S) which will be displayed to the user. The list of names entered by the user in response to this prompt is assigned to the synonym L and a loop is entered which splits successive names off the beginning of this list and passes them to a task for display. The line beginning with the asterisk is a comment line added for appearance. Below the comment line, a .IF statement is used to check for a particular user id of SYS000 and if the user executing the proc has a different user id, the .DATA command is

used to append a message to a log file of messages. The lines of text to the right of the exclamation marks in the above example are also comments which may optionally appear for documentation.

OTHER SCI FEATURES

SCI provides other features in addition to those previously mentioned. These additional features and some of those which have been only noted in passing are as follows:

(1) Tasks invoked by SCI may be run in foreground or background mode. This means that the user may chose to have SCI wait until his task completes before continuing execution, or he may choose to have SCI execute concurrently with his task.

(2) SCI itself may be run in a background (batch) mode in which it executes from a file of commands rather than receiving its input from a terminal. This is very useful for such things as compiling many modules or performing time consuming functions which can be set up and run without human intervention.

(3) Special procs can optionally be defined by the user which are automatically executed when a user logs on and when he logs off. This is useful in situations where one might want to record the amount of time that the user was logged onto the terminal. It is also useful in situations where a certain set of initialization procedures needs to be invoked when a user logs on.

(4) Synonyms have already been mentioned, but it should be added that SCI saves synonyms between a user's sessions so that a synonym defined during one session will be available in a later session.

(5) SCI does validation on the prompted inputs of procs so that mistakes are caught early and the user is reprompted until a correct response is entered.

(6) SCI provides initial values for prompts that represent frequently chosen values. This speeds the entry of commands and gives the user helpful information about the proper form of the response.

(7) SCI will display an optional news file to all users who log onto the system which contains whatever information the users of that system might deem appropriate for everyone to see.

ADDITIONAL FEATURES IN DNOS SCI

The DNOS version of SCI supports all the capabilities of the DX10 version of SCI plus has some additional features. Some of the more significant of these features are as follows:

(1) DNOS SCI allows alternate type specifications for prompts. For example in DX10, the value of a prompt could be specified as

an integer or a name, but not both. DNOS would allow the value of a prompt to be specified as either integer, name, or both.

(2) In DNOS the SCI type checking of prompts has been expanded to allow prompts to be given default values, to be elements of a specified set of values, and to be within a range of numbers.

(3) The .IF statement has been extended in DNOS to allow syntax checking of a value. For example, a value could be checked to determine if it is an integer, a name, an element of a specified set of values, etc.

(4) The error messages in DNOS SCI are more detailed than in DX10 SCI. In particular an error in a proc in DX10 will report the type of error that occurred, but DNOS SCI goes beyond this to also report the proc name and line number where the error occurred.

(5) The .DATA command has been extended in DNOS to allow the copying of data from an in-line file to a default terminal local file which will be displayed to the user at a later time.

SUMMARY

In summary, SCI has been one of the strong points of the DX10 and DNOS operating systems. It has proven to be a reliable piece of software which is very configurable and adapts to the needs of a wide class of users. It satisfies the needs of the casual user who desires a consistent, easy to use interface to the operating system as well as the needs of the user who wants to build on the operating system to provide his own customized interface.

NOTES

A LINK EDITOR OVERVIEW WITH COBOL SPECIFIC APPLICATIONS

Rodney Lancaster

TIMIX '83'

Techniques and Concepts
for the Nontechnical or
New User

Data Systems Group
P. O. Box 2909
Austin, Texas 78769

TEXAS INSTRUMENTS
INCORPORATED

Date: February 15, 1983

A LINK EDITOR OVERVIEW WITH COBOL SPECIFIC APPLICATIONS

1. INTRODUCTION

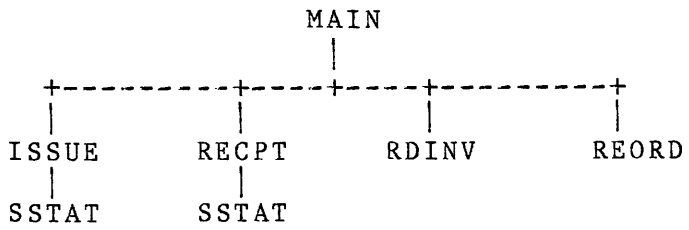
This presentation will overview the TI Link Editor and TI program files. It will show how COBOL programs can be linked several ways to save memory, enter execution faster, and save disk space. Specific items that will be covered are:

1. Program files - what are they?
2. Creating program files.
3. Linking and installing a COBOL task on a program file.
4. What are and using shareable procedures.
5. Using COBOL subroutines that are shared across several tasks.
6. Overlaying COBOL subroutines.

An application that we might consider for use in our discussion of linking and program files is an inventory management system that issues and receives materials and is capable of generating a reorder report. This application makes use of the following programs and subroutines:

1. MAIN - Main program.
2. ISSUE - Issues the materials from inventory.
3. RECPT - Receives materials into the inventory.
4. REORD - Creates a reorder report.
5. RDINV - Reads the inventory file.
6. SSTAT - Displays the status of a part-number in the inventory.

The call tree for this progam looks like:



2. PROGRAM FILES & SIMPLE LINKING

A program file is a special form of relative record file used to contain executable programs in memory image form. For COBOL the object produced by the compiler is in ASCII (readable character) format. This object can be linked into a COMPRESSED (Binary) format. Both ASCII and COMPRESSED format can be executed directly using the XCP or XCPF commands (Execute Cobol Program) but require conversion into memory image format at the load time. The program file is already in memory image format and does not require conversion at the load time and thus loads much faster than an object file.

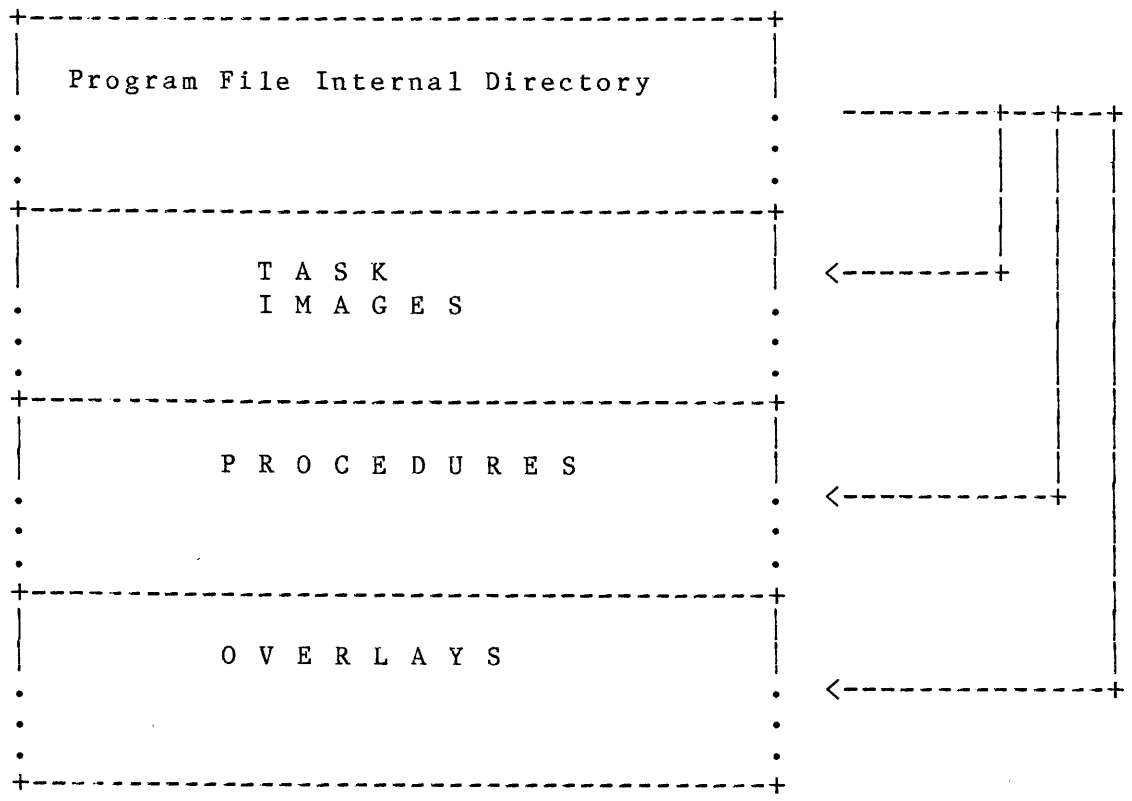
Program files also allow several facilities to reduce memory requirements. In a program file you can use Shared procedures and/or overlays.

A procedure is a segment of the task that usually consists of executable code. Shared procedures are procedures in a program file that are shared by more than one task. Shared procedures decrease the amount of disk storage required to store a program file.

Overlays are parts of a task that stay on the disk until explicitly requested. When it is requested the overlay replaces a part of the task previously in memory. Overlays reduce the amount of memory required to run a COBOL program. The program file also contains information that points to the different sections within a program file (tasks, procedures, overlays).

A diagram of a program file looks like the following.

STRUCTURE OF A PROGRAM FILE



Program files are created either by the Link Editor when linking the program or using the CFPRO (Create Program File) command.

```
[ ] CFPRO

CREATE PROGRAM FILE
                  PATHNAME: MY.PROGFILE
      MAX NUMBER OF TASKS: 25
MAX NUMBER OF PROCEDURES: 10
   MAX NUMBER OF OVERLAYS: 10
        INITIAL ALLOCATION: 85
     SECONDARY ALLOCATION:
              EXPANDABLE?: YES
```

The CFPRO command prompts are pretty simple to understand with the possible exception of the two ALLOCATION prompts. These two prompts are asking for how many disk units (ADUs) to start with initially and if it must increase beyond that the number of disk units to get when needed.

Since a program file cannot be displayed to a terminal directly a map must be made to see where each item in the program file is kept. The command to look at a program file is MPF. MPF is executed as follows.

```
[] MPF

    MAP PROGRAM FILE
          PATHNAME: MY.PROGFILE
LISTING ACCESS NAME:
```

The two prompts are:

1. PATHNAME - This is the pathname of the program file to be mapped.

2. LISTING ACCESS NAME - Is where you wish the listing to go. The default is to the terminal that executes the command. Any other input should be a file pathname or device name (LP01).

The map produced contains a wealth of information and looks like:

```
FILE MAP OF MY.PROGFILE
 TODAY IS 15:16:07 TUESDAY, MAR 02, 1982.

TASKS: MAXIMUM POSSIBLE = 25
ID   NAME LENGTH LOAD PRI S P M R D E O C OVLY P1/SAME P2/SAME INSTALLED
01 TXSF    0928  2340   4   P   R D           01/Y            11/11/81
02 GN990A  6674  0000   4   P     D       02                  11/11/81
03 TXMD    1166  2340   4   P   R D           01/Y            11/11/81

PROCEDURES: MAXIMUM POSSIBLE = 10
ID   NAME LENGTH LOAD    M D E W C                           INSTALLED
01 TXDXFC  2334  0000      D                                  11/11/81

OVERLAYS: MAXIMUM POSSIBLE = 10
ID   NAME LENGTH LOAD   MAP D  OVLY                         INSTALLED
01 INTCTA  42C2  23B2       D                                 11/11/81
02 BUILDA  2D76  23B2       D   01                            11/11/81
```

The Tasks are in ID order and the name of each is given. The length in bytes and the load address relative to the beginning of the program are given. Also given is the priority of the task and the last overlay used by the task. The first procedure used by the task is given as well as what the procedure ID number is and whether the procedure is located in this program file or in the system program file. The same information is given for the second procedure used by that task. The date of

installation is also given. There are indications whether certain flags are set. The flags are:

1. S - system program
2. P - privileged
3. M - Memory resident
4. R - Replicatable i.e. more than one copy can execute at a time.
5. D - Delete protected
6. E - Execute protected
7. O - Overflow protected for arithmetic
8. C - uses the Writeable Control storage area.

The Procedures are in ID order and the name of each procedure is given. Its length in bytes and load address is also given as well as its installation date on the program file. The flags that can be set for procedures are:

1. M - Memory resident procedure
2. D - Delete protected
3. E - Execute protected
4. W - Write protected
5. C - Uses the Writable Control Storage area.

The Overlays are also in ID order and have their names given as well. The length of the overlay in bytes, the load address of the overlay, and the installation date are given. The only flags that can be set for overlays are:

1. MAP - There is a relocation bit map associated with the file
2. D - Delete protected
3. OVLY - number of the previous overlay associated with the task.

The Link Editor requires a file containing a sequence of link edit commands that let the Link Editor know how this program is to be constructed. The following is a simple Link Edit control file that might be used for our application program.

```
FORMAT IMAGE,REPLACE          (use memory image format)
PROC RCOBOL                   (procedure name)
    DUMMY                     (procedure exists on another)
                              (program file)
    INCL .S$SYSLIB.RCBPRC     (attach task to procedure)
TASK BIG                      (task name)
    INCL .S$SYSLIB.RCBTSK     (task entry vector and data area)
    INCL .S$SYSLIB.RCBMPD     (main program designator for COBOL)
    INCL OBJ.LAB04            (main program)
    INCL OBJ.RDINV            (subr to read inventory file)
    INCL OBJ.ISSUE            (subr to issue stock)
    INCL OBJ.RECPT            (subr to add to quantity on hand)
    INCL OBJ.REORD            (subr to create reorder report)
    INCL OBJ.SSTAT            (subr to display status of part)
END
```

Each command in the file serves a purpose in telling the Link Editor what to do and how the program is to be put together.

1. FORMAT IMAGE, REPLACE - Specifies that the FORMAT of the program file is to be in memory image form and if it exists in the program file replace it.

2. PROC RCOBOL - Defines the beginning of the procedure segment and assigns the name RCOBOL to this procedure.

3. DUMMY - Specifies that the item just specified, in this case the procedure, is either already in the program file or is in another program file.

4. INCL .S$SYSLIB.RCBPRC - Tells the Link Editor to use the COBOL Runtime routine RCBPRC to attach the following task to the specified procedure.

5. TASK BIG - Signifies the beginning of the task segment and gives it the name BIG.

6. INCL .S$SYSLIB.RCBTSK - Means include the task entry vector for the COBOL Runtime.

7. INCL .S$SYSLIB.RCBMPD - Signifies to include the COBOL main program designator.

8. The rest of the INCLs are the main program and all the

subroutines for this program.

To link the program together use the command XLE.

```
[ ] XLE

EXECUTE LINKAGE EDITOR
      CONTROL ACCESS NAME: MY.CONTROL.FILE
LINKED OUTPUT ACCESS NAME: MY.PROGFILE
      LISTING ACCESS NAME: MY.LINKMAP
               PRINT WIDTH: 80
```

The prompts for XLE are:

1. The CONTROL ACCESS NAME is the pathname of the Link Editor control file mentioned above.

2. The LINKED OUTPUT ACCESS NAME is the pathname of the actual program file.

3. The LISTING ACCESS NAME is the pathname for the Messages produced by the Link Editor as well as the Link Map. The default is to the terminal that executed the command.

4. The PRINT WIDTH allows you to specify the width of the Listing file.

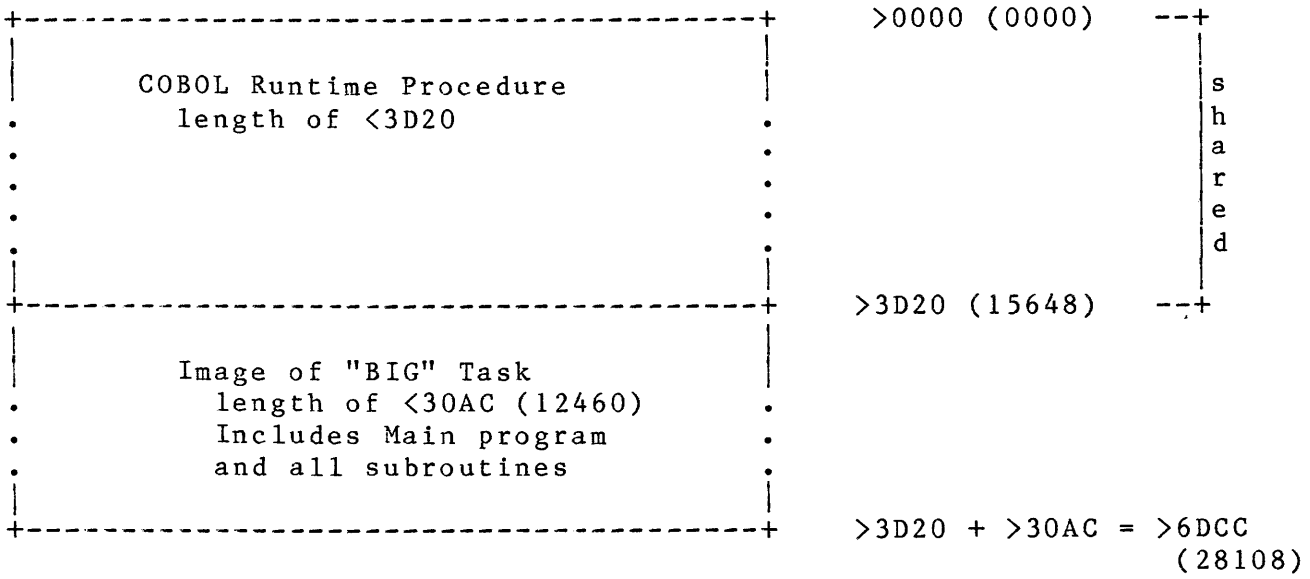
The following is a map of the program file after linking and installing the COBOL task BIG.

```
FILE MAP OF MY.PROGFILE
  TODAY IS 19:47:01 TUESDAY, MAR 02, 1982.

TASKS: MAXIMUM POSSIBLE =  10
ID NAME LENGTH LOAD PRI S P M R D E O C OVLY P1/SAME P2/SAME INSTALLED
01 BIG   30AC  3D20   4        R             10/N              3/ 2/82

PROCEDURES: MAXIMUM POSSIBLE =  10
ID NAME LENGTH LOAD    M D E W C                              INSTALLED

OVERLAYS: MAXIMUM POSSIBLE =  10
ID NAME LENGTH LOAD   MAP D  OVLY                            INSTALLED
```

Look at the P1/SAME column. It indicates that the COBOL Runtime procedure is not in this program file and therefore can be shared. We have saved already 15648 (>3D20) bytes of storage in the program file itself without sacrificing any speed of

execution. Also note that none of the Inventory software subroutines are shared.

The COBOL Runtime procedure exists on the system program file and is shared by all COBOL tasks that are linked to it. A diagram of what the program file looks like is below.



To execute this program on DX10 you should first assign a luno (Logical Unit Number) to the program file as follows:

```
[] AL

ASSIGN LUNO
          LUNO:
   ACCESS NAME: MY.PROGFILE
 PROGRAM FILE?: YES
```

The prompts are:

1. LUNO - You may select the luno number to be assigned. The only time this causes a problem is if the luno is already assigned.

2. ACCESS NAME - The pathname that the luno is assigned to.

3. PROGRAM FILE? - Whether this pathname is a program file or not.

This Luno that is returned must be used in the XCTF (Execute

COBOL Task Forground) command. The XCTF command is as follows:

```
[] XCTF

EXECUTE COBOL TASK FOREGROUND <VERSION: 3.4.0 82062>
      PROGRAM FILE LUNO: use value returned from AL
        TASK ID OR NAME: BIG
             DEBUG MODE: NO
    MESSAGE ACCESS NAME:
                SWITCHES: 00000000
          FUNCTION KEYS: NO
```

The prompts are:

1. PROGRAM FILE LUNO - The Luno number returned from the AL command.

2. TASK ID OR NAME - The name of the task in the program file.

3. DEBUG MODE - Whether the COBOL debug utility is to be entered. It should be noted that a module RCBTSKD must be used insead of the module RCBTSK. RCBTSKD contains the debug module.

4. MESSAGE ACCESS NAME - The pathname to be used for COBOL messages and errors.

5. SWITCHES - The COBOL switches that may be used. 0 - off, 1 - on.

6. FUNCTION KEYS - Whether the function keys are inabled for use in the program by using the ON EXCEPTION condition.

3. LINKING SHAREABLE PROCEDURES

In the simple example I made mention that the COBOL Runtime procedure RCOBOL is shared with all COBOL tasks that are linked to it. You can set up procedures of your own that can use this ability to be shared and save space in the program file. A shared procedure could be a set of subroutines that can be used by more than one task I.E. the RDINV (Read inventory), ISSUE (issue stock report), RECPT (receive new material into the inventory), REORD (create the reorder report), and SSTAT (show the status of a part in the inventory).

Below is a Link Control file that sets these up as a procedure that can be shared between more than one task.

```
FORMAT IMAGE,REPLACE          (memory image format)
PROC RCOBOL                   (procudre 1 name)
    DUMMY                     (procedure already exists)
    INCL .S$SYSLIB.RCBPRC     (COBOL runtime proc)
PROC INVSUBS                  (procedure 2 name)
    INCL OBJ.RDINV            (subr to read inv file)
    INCL OBJ.ISSUE            (subr to issue stock)
    INCL OBJ.RECPT            (subr to add to quant on hand)
    INCL OBJ.REORD            (subr to create reorder report)
    INCL OBJ.SSTAT            (subr to display status of part)
TASK SHARE
    INCL .S$SYSLIB.RCBTSK     (task entry vector and data)
    ALLOCATE                  (force data segment allocation)
    INCL .S$SYSLIB.RCBMPD     (COBOL main program designator)
    INCL OBJ.LAB04            (main program)

END
```

You should notice that the FORMAT statement thru the INCL of the RCBPRC is the same. The new items to be looked at are:

1. PROC INVSUBS - The procedure containing the code of the subroutines. The INCLs that follow up to the TASK command tell the Link Editor to include the named subroutines in this procedure.

2. TASK SHARE - Starts the task segment by the name of SHARE.

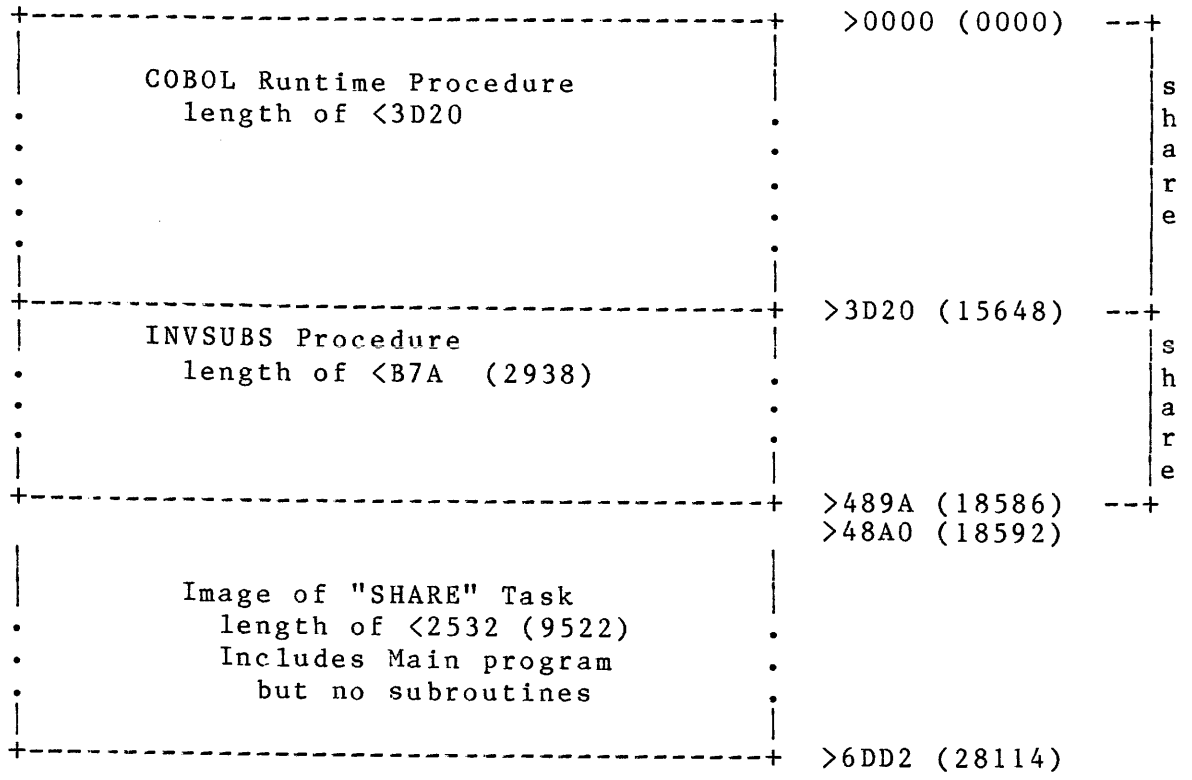
3. ALLOCATE - Allocates storage for the DATA areas of the procedures. This also includes the DATA areas of the shared procedures.

All the other items were mentioned previously.

After using XLE to install the task SHARE and the procedure INVSUBS the map looks like:

```
 FILE MAP OF MY.PROGFILE
   TODAY IS 20:49:43 TUESDAY, MAR 02, 1982.

TASKS: MAXIMUM POSSIBLE =  10
ID  NAME LENGTH LOAD PRI S P M R D E O C OVLY P1/SAME P2/SAME INSTALLED
01 SHARE  2532  48A0   4       R             10/N    01/Y      3/ 2/82

PROCEDURES: MAXIMUM POSSIBLE =  10
ID  NAME  LENGTH LOAD   M D E W C                              INSTALLED
01 INVSUBS 0B7A  3D20                                           3/ 2/82

OVERLAYS: MAXIMUM POSSIBLE =  10
ID  NAME  LENGTH LOAD  MAP D  OVLY                             INSTALLED
```

The diagram of this program now looks like:



It may be convenient, but it is not required, to make a partial link of the shared procedure. The following shows a Link Control File that will accomplish this.

```
PARTIAL                  (create a partial link)
TASK INVSUBS             (le idiosyncracy)
    INCL OBJ.RDINV       (include object for all
    INCL OBJ.ISSUE         routines that comprise
    INCL OBJ.RECPT         the shared procedure)
    INCL OBJ.REORD
    INCL OBJ.SSTAT
END
```

The two things to notice are:

1. PARTIAL - This lets the Link Editor know that this is not a complete program.

2. TASK INVSUBS - This is an idiosyncracy of the Link Editor. Even though this is going to be linked in as a procedure for some odd reason the Link Editor requires the TASK statement.

To create the partial link use the XLE command. It should be noted that this partial link is installed on an object file and not a program file. To Use this partial link, a Link Control file must be created that includes this object file as a procedure. The following is an example of this.

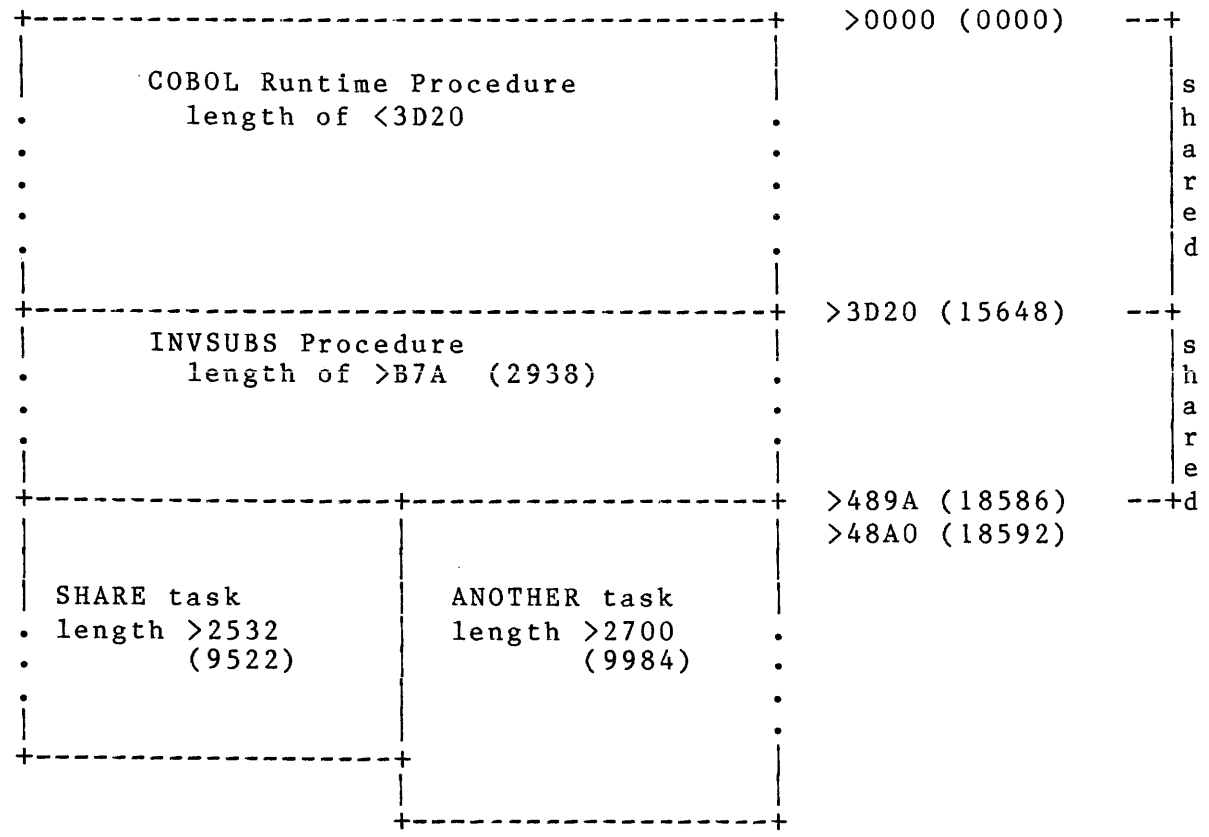
```
FORMAT IMAGE,REPLACE
PROC RCOBOL
    DUMMY
    INCL .S$SYSLIB.RCBPRC
PROC INVSUBS
    DUMMY
    INCL OBJ.PARTIAL          (include the partial link)
TASK ANOTHR
    INCL .S$SYSLIB.RCBTSK
    ALLOCATE
    INCL .S$SYSLIB.RCBMPD
    INCL OBJ.ANOTHR
END
```

The one new thing to notice is instead of listing all the subroutines separately, the object file from the partial link is used for the INCL. Also notice that the procedure is DUMMY'd as if it already was on the program file.

After using the XLE command to install this task on a program file the map looks as follows.

```
 FILE MAP OF MY.PROGFILE
   TODAY IS 12:57:47 WEDNESDAY, MAR 03, 1982.

TASKS: MAXIMUM POSSIBLE =  10
ID   NAME LENGTH LOAD PRI S P M R D E O C OVLY P1/SAME P2/SAME INSTALLED
01 SHARE   2532  48A0   4       R              10/N    01/Y      3/ 2/82
02 ANOTHR  2700  48A0   4       R              10/N    01/Y      3/ 3/82

PROCEDURES: MAXIMUM POSSIBLE =  10
ID   NAME LENGTH LOAD    M D E W C                              INSTALLED
01 INVSUBS 0B7A  3D20                                           3/ 2/82
```

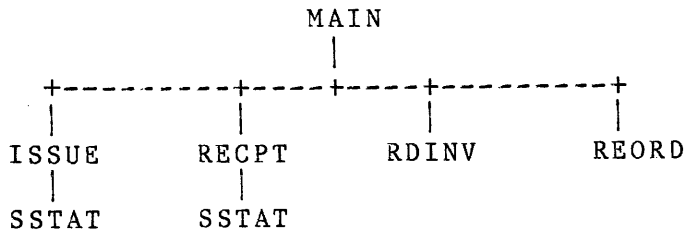
The diagram for this map containing two tasks and shared procedures follows.



When more than one task uses this shared procedure you will be saving 2938 (>B7A) bytes of disk space for each task in the program file that uses this shared procedure.

4. OVERLAYS

A way to save space in memory is to use overlays in your program file. An overlay is a part of the task that stays on the disk until it is explicitly called for. The overlay replaces the part of the task in memory that uses the same load address as the overlay. This allows the saving of space in memory by making the program size in memory be the sum of the procedure, the task, and the size of the largest overlay to load. Before we get into the Link Control File that might be used for our application lets review the call tree for this application.



This call tree allows for four main overlays (ISSUE, RECPT, RDINV, and REORD) as well as two sub overlays (SSTAT being called by both ISSUE and RECPT). The following is an example of how we might set up the Link Control File to accomplish our overlay setup.

```
FORMAT IMAGE,REPLACE
LIBRARY .S$SYSLIB          (library required for load directive)
PROC RCOBOL
    DUMMY
    INCL .S$SYSLIB.RCBPRC
PHASE 0, MAIN              (level zero is the root node named MAIN)
    INCL .S$SYSLIB.RCBTSK
    LOAD                   (use the automatic overlay loader)
    INCL .S$SYSLIB.RCBMPD
    INCL OBJ.MAIN
PHASE 1, OVLY1             (level one: subr ISSUE)
    INCL OBJ.ISSUE
PHASE 2, OVLY11            (subr SSTAT is an ovly under ISSUE)
    INCL OBJ.SSTAT
PHASE 1, OVLY2             (level one: subr RECPT)
    INCL OBJ.RECPT
PHASE 2, OVLY21            (subr SSTAT is an ovly under RECPT)
    INCL OBJ.SSTAT
PHASE 1, OVLY3             (level one: subr RDINV)
    INCL OBJ.RDINV
PHASE 1, OVLY4             (level one: subr REORD)
    INCL OBJ.REORD
END
```

There are five main new items in this Link Control File.

1. LIBRARY .S$SYSLIB - This is required for the load directive to work properly. The library command allows you to include routines that are contained in that library file specified and are called by the routines in the Link Control File. If a routine is named in the Link Control File and is in a library specified, only the leaf name need be named surrounded by parenthesis. I.E. .S$SYSLIB.RCBPRC could have been put as (RCBPRC) since it is in the library .S$SYSLIB.

2. PHASE 0, MAIN - This replaces the TASK statement from the other program files. The 0 lets the Link Editor know that this is the root of the overlay structure.

3. LOAD - Signals the Link Editor to use the automatic overlay loader in .S$SYSLIB.

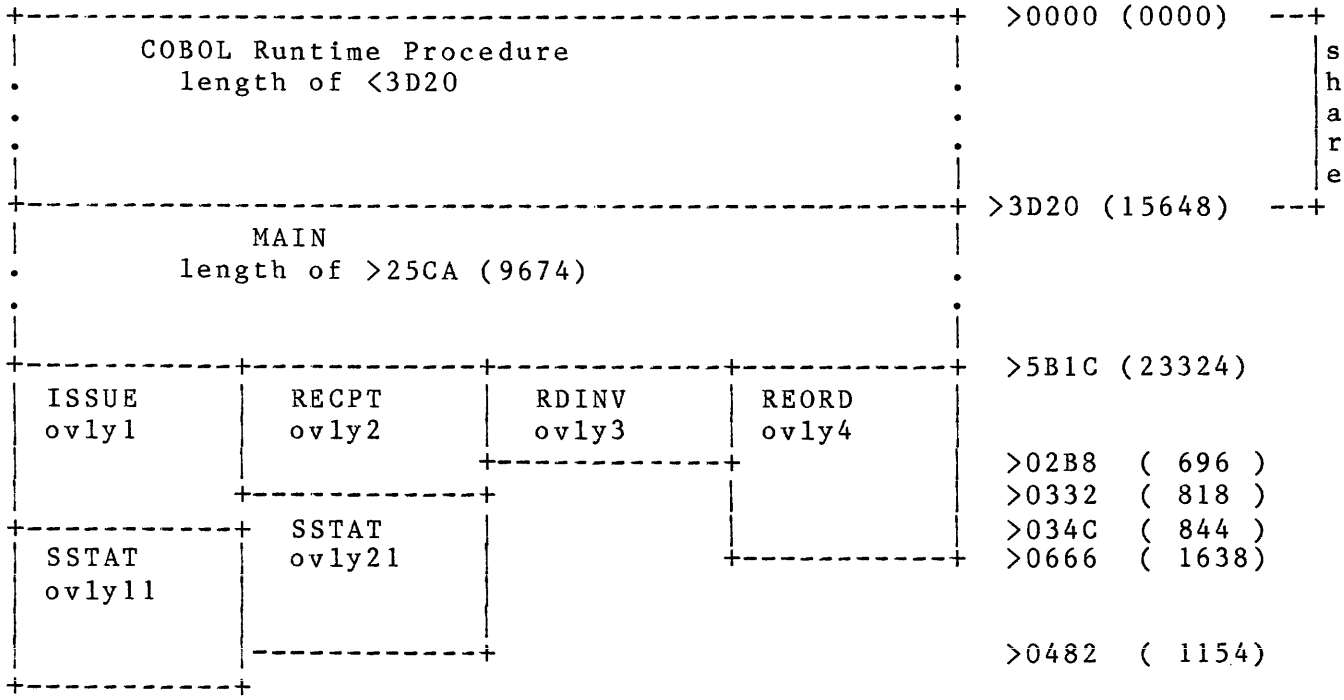
4. PHASE 1, OVLY1 - This command signifies the beginning of a new phase in the overlay structure, assigns a level 1 (first level overlay) and a name to the overlay.

5. PHASE 2, OVLY11 - This command signifies the beginning of a new phase in the overlay structure, assigns a level 2 (second level overlay) and a name to the overlay.

After using the XLE command to link and install the MAIN routine and the overlays on a program file the map of the program file would look like the following.

```
 FILE MAP OF MY.PROGFILE
   TODAY IS 16:43:52 WEDNESDAY, MAR 03, 1982.

TASKS: MAXIMUM POSSIBLE =  10
ID  NAME LENGTH LOAD PRI S P M R D E O C OVLY P1/SAME P2/SAME INSTALLED
01 MAIN   25CA  3D20   4       R          06  10/N              3/ 3/82

PROCEDURES: MAXIMUM POSSIBLE =  10
ID  NAME LENGTH LOAD    M D E W C                              INSTALLED

OVERLAYS: MAXIMUM POSSIBLE =  10
ID  NAME LENGTH LOAD   MAP D  OVLY                           INSTALLED
01 OVLY1  034C  5B1C                                          3/ 3/82
02 OVLY11 0482  5E68           01                             3/ 3/82
03 OVLY2  0332  5B1C           02                             3/ 3/82
04 OVLY21 0482  5E4E           03                             3/ 3/82
05 OVLY3  02B8  5B1C           04                             3/ 3/82
06 OVLY4  0666  5B1C           05                             3/ 3/82
```

A diagram of the program file looks like the following.



Notice that with overlaying, mem required is the root plus the longest overlay sequence. For our example this amounts to:

```
>5B1C  +   ( >034C + >0482 ) = >62EA
23324  +   (   844 +  1154 ) = 25322
```

Without overlaying, the memory required is the root plus the sum of all the unique overlays. In our case this works out to:

```
>5B1C + >02B8 + >0332 + >034C + >0666  = >6AB8
23324 +   696 +   802 +   844 +  1638  = 27320
```

The savings from using overlays in this example is >07CE or 1998 bytes in memory.

5. SUMMARY

We have looked at three main ways to link a COBOL program for speed, disk space savings and for saving memory space. All three methods use the memory image format in the program file and thus will not need to be converted in the process of loading into memory. Depending on your needs one of these three methods will help in speed of execution, saving of disk space, and effective memory utilazation. Use the following list to help decide which method to use.

1. If speed is the most important, and the program is not huge (larger than 64K bytes long when linked) use the normal linking procedure that puts the whole program in memory at the same time.

2. If you have a set of subroutines that will be used in several tasks, use the shareable procedure linking method to save disk space.

3. If the program is large (bigger than 64K bytes) then use the overlay method to utilize memory more effectively. Using overlays will slow down your program due to loading the overlays from the disk.

When an application is ready to be put into operation consider the method of linking carefully. Linking is not hard, but like all programming careful thought should be used.

COMPUTER SYSTEM HARDWARE

TI-MIX SYMPOSIUM, NEW ORLEANS, 1983

TECHNIQUES AND CONCEPTS FOR THE NONTECHNICAL OR NEW USER SESSION

Art Miller

Texas Instruments

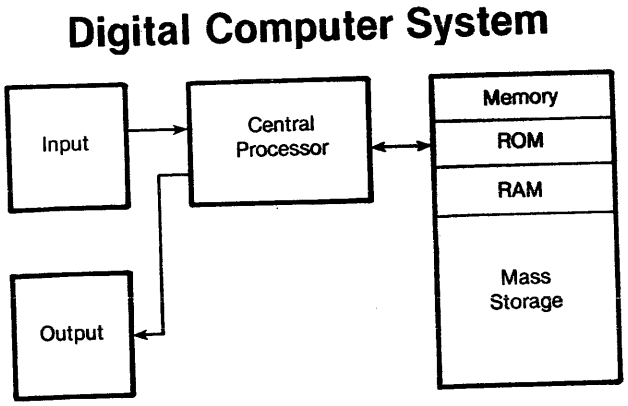
Austin, Texas

Digital computer systems are functionally composed of only a few elements. There is some means of placing input into the computer; a means of receiving output from the computer; a computational element to perform manipulations on the input; a program that defines what those manipulations are; data to be input, stored or manipulated; and memory to keep track of results or systems operation. There are many types of input, output, computer architectures, and memory. Even more varied are the programs that have been written for each configuration, or arrangement, of these varied elements and the data that they operate on and with. The purpose of this paper is to present a basic approach to these elements for the uninitiated user of computer systems.

Requirements for Digital Computer

- Computational Element (CPU)
- Memory (RAM, ROM, mass storage, paper)
- Input Element (CRT, VDT, KSR, mass storage)
- Output Element (terminal, printer, memory)
- Program
- Data

This is a block diagram, or symbolic representation, of a possible computer system. Note in this example that input and output both have to pass through the central processor, or CPU, in order to reach the memory elements of the system. This is a typical type of configuration, but is not the only architecture presently used. There are, for example, machines where input can enter directly into system memory. Our initial concentration will be on the central processor - what it does and why it does it.



All digital computers are presently based upon Boolean Logic. This area of mathematics was developed by George Boole, an English mathematician during the 1800's. The whole pretense of Boolean algebra is that there are only two possibilities for each element - either "1" or "0." This is contrasted with decimal arithmetic where there are ten possibilities for each element - "0" through "9."

Boolean Algebra

In the beginning there was nothing,
and he* called it '0' . . .

And then there was something,
so he called it '1' . . .

So it was necessary to define a
whole lot of operations around 1 and 0!

* George Boole († 1864, English Mathematician)

The real ascendancy of Boolean algebra did not occur until the invention of the digital computer during the 1940's. This type of mathematics is perfect for electrical processing elements defined by digital logic - each element is either "on" or "off." Use of Boolean algebra allows definition of addition and subtraction. Shifting and addition then define multiplication, while shifting and subtraction define division. Use of recursive operations, (operations performed repeatedly), and other mathematical relationships such as truncated series approximations, allows development of a full complement of mathematical operations. These mathematical and data manipulation, (shifting, negating, etc.), operations form the arithmetic logic unit, or ALU, portion of the CPU.

Boolean Logic

- Great for electrical function
  - "On" = "1" "Off" = "0"
- Defines addition
  - Multiplication is "shift" and add
- Defines subtraction
  - Division by shift and subtract
- Recursion yields integration, exponentiation, etc.
- Mathematical ability resides in CPU
  - Arithmetic logic unit (ALU)

For the sake of completeness, these are examples of Boolean Operations. The dot and plus sign, normally understood to indicate multiplication and addition, define different operations in Boolean algebra. These operations are "And" and "Or," and are best explained by example tables giving a set of inputs and outputs. Since my intent is not to teach a Boolean algebra course or to make you digital logic designers, this is as far as we will explore the subject.

Boolean Operations

$x \cdot 1 = x$

$(x + y)(x' + z) = xz + x'y$

$x + 1 = 1$

$(x + y + z + \ldots)' = x'y'z' \ldots$

$(x + y)(x + y') = x$

$x \cdot x' = 0$

etc., for $(x, y, z) = (0, 1)$

So far we understand that a digital computer requires data, arithmetic and manipulation capability on that data, and a program to provide information on the intended relationships between pieces of data. (I.E., a program may define billings to be the sum of individual sales receipts, or it may define billings to be the product of average sales per receipt times total number of reciepts.) However, it is always true that initial input and final output must be in human usable form. The basis for providing a more readable format arises from coding. In some cases this coding is in several levels.

Central Processor (CPU)

- Digital computers require
  - Data
  - Arithmetic capability
  - Data manipulation capability
  - Information on relationships (program)
- Basis for representation is coding

Coding allows us to represent a large number of bits as a small number of symbols. These tables show the evolution of the hexadecimal coding system where four bits, (individual sets of "1" or "0"), can be represented by a single alphanumeric character. Another example of a common coding scheme is the ASCII character set wherein the 128 possible combinations representated by the hexadecimal characters 00 through 7F, (0000000 to 1111111), encode the characters of a standard typewriter keyboard. This allows the computer to recognize, for example, the character "40" as the letter "A."

Coding

A set of "N" bits (sets of "on" or "off" possibilities) has 2^N *ordered* combinations . . .

1 Bit = 2^1 = 2
0
1

2 Bits = 2^2 = 4
00
01
10
11

3 Bits = 2^3 = 8
000
001
010
011
100
101
110
111

Most Common Code is "Hex"
(Hexadecimal ≡6 + 10 or 16 combinations)

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - A
0011 - 3	1011 - B
0100 - 4	1100 - C
0101 - 5	1101 - D
0110 - 6	1110 - E
0111 - 7	1111 - F

Code convention yields shorthand, i.e.,:

A40 ≡ 1010 0100 0000

Data in a computer typical exists in ordered sets of fixed length. A general shorthand for these fixed length patterns is the "byte," which is a set of eight "bits." The computer architecture then defines the width of the data and address paths which determines much about the capability of the computer system. Typical data paths are, for example, the 8-bit Apple computer, the 16-bit minicomputers, the 32-bit super-minicomputers, and the 64-bit mainframe computers exemplefied by the IBM 370 machine. Each set of data then has associated an address, or location, where the data may be found. Having an address for data then allows ordered access to the data (recall), storage and recall of results, and branching to new locations in a program according to results. it is important to note that data in memory can be either just data or the instructions in a program. Therefore, it matters where in the data you start operating. A computer architecture alway has a "Reset" function that allows you to start operation at a know location in the data.

Data Streams

- Computers handle data groups of "fixed" length
  - 8 bits is 1 byte
  - Defines memory organization, data path size
- Lists of data groups have an order numbering (address)
  - Ordered access
  - Storage and recall
  - Branching based on results
- Members of list may be data *or* instructions
  - It matters where you start
  - 'Reset' defines starting at a known address

This is a block diagram of a central processor unit, or CPU. This arrangement of structures is often referred to as a von Neumann machnine since all input to and output from the memory must be routed through the central processor unit. I will step through several instructions to show how the processor would implements instructions:

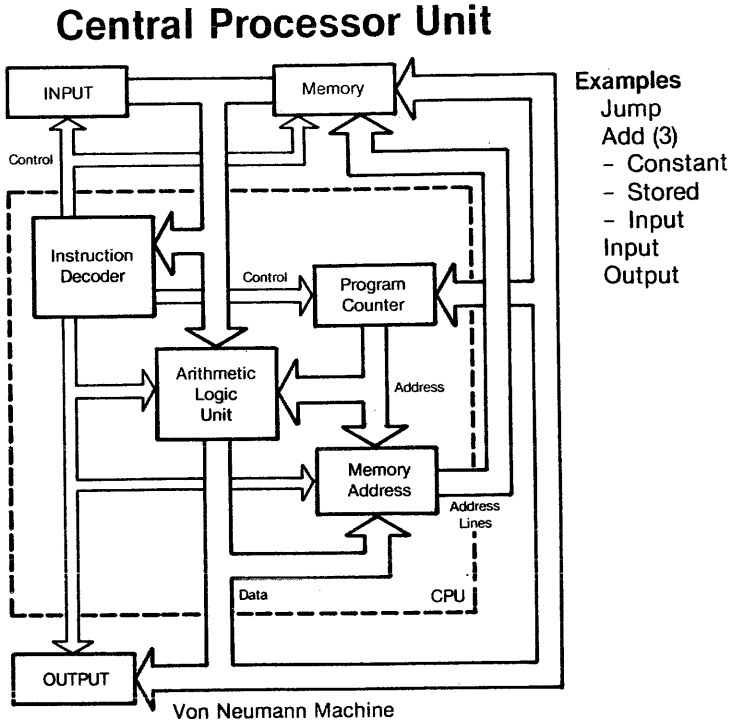
Jump - When the program counter provides the memory address register (storage location address holding point) with an address for memory, the output of memory is assumed to be an instruction and is loaded into the instruction decoder. If the instruction is a "Jump" command, then the next location in memory is loaded into the memory address register as the spot to jump to! It is also possible to jump by a fixed amount from where you are in the program. This is called a "Jump Relative."

Add - When an "Add" instruction is decoded, the numbers to be added are retrieved from memory and/or input and the addition is forced by the control lines. It is possible that one of the numbers is the next number in the program, (add a constant), is in memory, (add a stored value), or is waiting at the input. In the case of the latter an "Input" command is executed first.

Input - When an "Input" command is activated, control lines deactivate the memory output and activate the output of the input element. Then the input data is available to the CPU.

Output - An "Output" command is the reverse of "Input" in that the control lines force the results of the operation, available at the output of the ALU, into the output element.

A typical processor element will understand several hundred variations of commands. This set of allowable variations is refered to as the instruction set of the computer, and corresponds to the minimum understandable syntax, or language, for the computer.

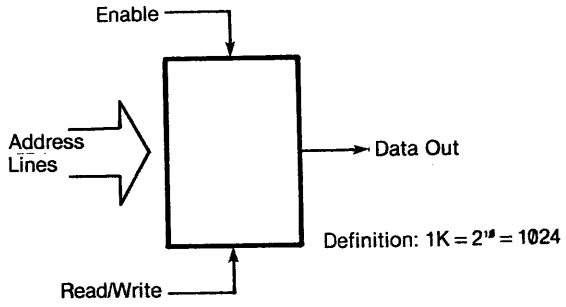


Programs and data are typically stored in memory. While operating, they are usually placed into RAM, or random access memory, to provide quick access for fast operation. The first successfal RAM was core memory. This memory was implemented with small, magnetic donuts woven into a grid of access lines. Needless to say, this memory was very bulky, but had an advantage in that it was nonvolatile. This means that it retained data even when the power was turned off.

New memory technology is primarily based upon silicon integrated circuits. It is presently possible to put over 250,000 bits of data on a single chip one-quarter of an inch on a side. The dominant technology now, though, is 64K DRAM with over 64,000 bits of data on a single chip. DRAM stands for dynamic random access memory. This type of RAM requires constant updates, or "refresh" cycles to keep the data in memory from being forgotten. Static memory refers to memory that only requires the power to remain on for memory to be maintained.

Memory - Random Access (RAM)

- Matrix of storage locations
- First successful RAM was magnetic core
- Dominant technology is volatile, Dynamic 64K RAM
  - 2^{16} bits of information on single chip



Read only memory, or ROM, is typically used to store commputer diagnostics and program load routines - the types of programs needed to start operation of a system. Several methods of creating ROM's are available: A ROM is mask-programmed, meaning that the data is inserted into the circuit during manufacture and can never be changed. These chips are the cheapest method in large volumes when the contents are stable. PROM's allow data to be electrically encoded one time after manufacture. These are typically used for small production runs, or during development.

EPROM and EAROM chips are used to store data that may change on a routine basis. EPROM's can be erased by extended exposure to ultraviolet radiation. EAROM's can be erased by electrical signals. EPROM's are usually used during development, while EAROM's seem to find favor in storing slowly updated material, such as system calibration values in measurement equipment.

Memory - Read Only (ROM)

- Same organization as RAM
  - "Write once," read many times
  - Nonvolatile
- Different methods to input data
  - ROM, PROM, EPROM, EAROM
- Store starting, permanent programs for computers
  - Diagnostics
  - Load routines

Booting the System!

Mass storage devices run the gamut. The purpose of such devices is to provide long-term, non-volatile, high-capacity memory for a system. A unit might, for example, provide room to store several different programs and the data for each. When a program is activated, then it is typically moved into RAM along with its data where it then operates. The first types of mass storage available were the friendly paper systems - paper tape and punch cards. Hollerith punch cards were first used before the turn of the century to help with the census. These paper systems are bulky and subject to abuse. As a result, more common methods today rely on magnetic media.

In magnetic media, a material is coated with a magnetic material. A head runs close to the material and causes flux (magnetization) polarization changes in the material dependent upon the inputs to the heads. Magnetic tape units were the first major units available, but disk technology has taken hold due to its faster access speed to random data. (Tape drives have data stored piece after piece. The data must always be randomly accessed.) Over the years, large hard disk drives (using a stiff platter) have found favor in large computer systems. Smaller computer systems serving one or a few users have relied on flexible, or floppy, disk drives. In this case the media is much thinner and bendable. The advent of winchester, (sealed), disk drives has allowed the technology to mature to where small hard disk drives are the largest growing market.

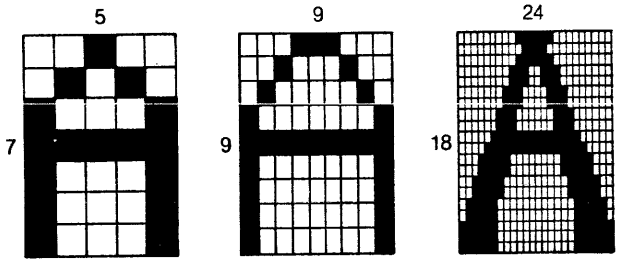
Note that disk drives rely on an ability to record much data in a small area. In order to accomplish this, they must ride close to the surface of a disk. However, in order to keep from wearing the surface of the unit and the read/write head of the drive, the heads should not ride on the surface. (Typically the surface is spinning at 3600 rpm - much faster than a record player.) In winchester units the head rides so close to the surface that a smoke or dust particle is larger than the head clearance. Hitting such a particle can cause the airfoil-shaped head to actually waver in its path and "crash" back into the surface, destroying the head and the disk and the data. Due to possibilities of this and other failures, data is usually "backed up," or copied and stored by some other device in a redundant fashion.

Future optical technology disk devices utilizing lasers will allow us to move from several megabytes of data per disk surface into the gigabyte (one billion bytes) per surface range.

Now that we have discussed the internal workings of a computer - memory and the central processor - we are ready to discuss input and output. These usually take the form of encoding or decoding data. In most cases a display of the data takes place, and the data is presented as alphanumer - a combination of letters, numbers and characters. These characters are made up of individual little on-off areas on a screen that are grouped to form characters. As shown on this foil, the density of the array available to form each character has a drastic effect on the readability of the display. In higher-priced systems it is possible to obtain pixel-addressable displays which allow high-quality graphical presentation. This allows a mixture of text and graphics in a dot matrix type of system.

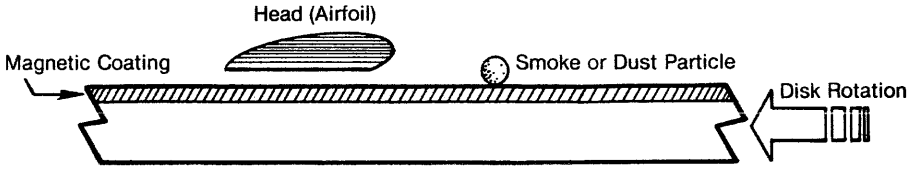
Computer Display Elements - Dot Matrix

- Display mechanism is typically an array of dots (pixels)
  - Array broken into small display arrays per character
  - Character codes built into ROM
- Pixel-addressable arrays require much memory/better control
  - Low cost systems typically use block graphics
- Typical arrays are 5x7, 9x9, and 9x15
  - Near letter quality arrays can be 24x18, etc.
- Denser arrays print slower (more passes) or cost more



Memory - Mass Storage

- First systems for mechanical calculators
  - Hollerith cards, paper tape
- Magnetic tape improved density, durability
  - Data still serial
- Dominant technology is magnetic disk
  - Fast, near random access
  - Coated media may be flexible (floppy) or rigid
  - New technology is Winchester (sealed)
- Future technology is optical (laser)



Typical input elements are Video Display Tubes. These elements are really television monitors with keyboards and electronics. They are very flexible to edit, and are typically very fast units. A typical screen of data is 24 lines high with an 80 column display. This can display 1920 characters, and can be completely changed in one second with a unit interfaced at 19,200 baud. (One baud is a bit per second. Baud implies a communications overhead of stop and start bits surrounding a character. Therefore, each eight-bit character requires ten bits in a transmission mode.)

Input Elements - VDT

- Television display that shows dots
  - Incorporates screen, keyboard, controller
- Easy to change (edit) inputs
- Typical units are serial interface
  - Change screen at 300 baud to 19,200 baud
    (30 characters to 1920 characters per second)
  - Typical screen is 24 lines, 80 characters per line
    (1920 characters per screen)
  - Usual display is green monochrome

A keyboard send-receive unit, or KSR, typically displays both input and output on a paper copy. These units are much slower, typically printing at 30-120 characters per second, but are much more portable than a VDT unit, especially when hard copy print out is required. (In the case of a VDT this generally requires the addition of a separate printer unit.)

Input Elements - KSR

- Incorporates keyboard and paper display
  - Thermal paper or plain paper
- Evolved from teletype (TTY) functionality
- New units are 30-120 characters per second input/output

The typical output element is either a video display tube or a printer. The usual device is an RO, or receive-only printer. Dot matrix printers can allow the flexibility of mixing graphics and text and seem to offer the best speed for the price. Band printers are usually higher speed devices than matrix printers and address 300-1200 line per minute output requirements. Print wheel or ball devices are like electronic typewriters and are used for the requirements where a fully formed character is required for excellent readability. This document was printed on a print wheel printer. They typicall offer 12-50 character per second printing.

More exotic printers for high-speed and specialized output use lasers and ink jets to form the output. These units can cost hundreds of thousands of dollars each and are used in instances where pages of output are required per second.

Output Element - RO Printer

- Methods of print vary
  - Dot matrix
  - Band printer
  - Ink jet
  - Print wheel or ball
  - Laser
- Speeds from 20 characters to several pages per second
- Typical units are 80-150 cps, dot matrix
  - Allow mix of text and graphics

The subject of programs and languages is the topic of another paper in this series, and I shall not get into that final area of computer systems. The finish of this presentation is a series of photographs showing the build up of a Series 300 Business System to display the areas we have just discussed on a computer system.

Your Part in Business System Documentation
Techniques and Concepts for the
Nontechnical or New User Session
by

Charles E. Watkins and Adrienne Gardner
Texas Instruments
Austin, Texas

1. INTRODUCTION

Until recently, TI developed computer systems primarily for use by data processing professionals. This meant that we could count on the readers of our documentation to be familiar with most of the concepts and terminology involved. We could also count on our users to be an experienced readers of technical documentation and to know how to use and maintain a technical document set. However, once TI stepped into the end-user market, we had to reassess our documentation strategy in light of the needs of a new type of reader. We could no longer assume that the reader has a background in data processing. We could no longer assume the reader knows the language and structure of technical documentation. In sum, we could no longer assume that our tried and true approach to technical documentation would meet the needs of this new group of users.

The first products aimed at this type of customer are on the low end of the Business System Series, the Business System 200 and the Business System 300. As one of the writers who worked on the DX10 Micro manuals and the operator's guides for Business System 200 and 300, I'm in a position to give you some background on how we developed the documentation for these products and how we think you can get the most out of it. First, I'd like to give you the big picture of the documentation for the Business System series and show you how all the parts fit together. Then, we'll look at the parts individually and you'll see how they're organized and where you should look to find the information you need. Finally, I'll go over the procedure for adding to your Business System documentation and software as your system grows to meet your needs.

2. DOCUMENT TYPES

Apart from the technical manuals used by the people who service the equipment, there are three types of documents in the Business System series. The first type tells you how to set up and operate the system itself. The second type tells you how to install and use one of the ready-made application packages such as Payroll or Inventory Control. The third type explains to people with some programming experience how to develop new application software for your Business System computer. Together, these three types of documents provide everything you need to know for the everyday operation of the system and for development of your own applications.

2.1 System Operator's Guide

In addition to a Business System 200 or 300 computer, a typical system could include either or both of two disk units (WD500 or WD800), either or both of two printers (840 RO or 810), and in the case of Business System 300, one or more Business System Terminals. Also, we offer no less than four operating systems for the Business System computers--DX10 Micro, DX10 Run-Time, the full DX10, and the p-System (tm UCSD).

Obviously, no single operator's guide could cover everything without distracting the reader with lots of information about parts of the system the reader doesn't even have. And since we wanted to develop step-by-step procedures that pertain to the actual hardware on site, we couldn't use generic terms throughout the operator's guide.

What we decided to do is create a modular operator's guide. Each part, or module, would cover one aspect of the system and we would pack the module that deals with a piece of equipment in the carton with the hardware. The shipping carton for the computer has the module on the computer display unit and keyboard, along with the binder, tabs, and some general information. The shipping carton for each model disk drive contains a module on how to set up and operate that particular type of drive. Likewise, the carton that contains one of the printers also contains a module with the procedures for setting up and operating that type of printer.

By inserting these modules into the binder that comes with the computer, you build up a single operator's guide that is tailored to your equipment. When you receive your operating system software, it comes with two modules--one on how to install the software and another on how to use it from day to day.

This leaves only the information on troubleshooting--what to do in case of a problem. For Business System 200, we decided that this topic deserved its own module in the operator's guide, but we still didn't want to burden our readers with information about equipment they don't have. So what we did was package the troubleshooting information with the module for each piece of equipment and ask the user to insert it behind the Troubleshooting divider in the operator's guide binder. This allows the Business System 200 user to build a troubleshooting module tailored to the system at hand.

By the time we did the Business System 300 operator's guide, we decided that this approach required more effort than it was worth, so we did away with the separate Troubleshooting section and left the troubleshooting checklists as part of the modules dealing with different parts of the system. As a result, when a piece of equipment goes with both Business System 200 and 300, there's a page in front of its troubleshooting checklist that tells you to move it to behind the Troubleshooting divider if you have a Business System 200 or to leave it in place if you have a Business System 300.

2.2 Application User's Guides

The Business System 200 and 300 computers support a variety of applications, ranging from Word Processing to Account Management. TI sells these applications as packages consisting of the application software and the application user's guide. Unlike the operator's guide, which deals with general system operations, an application user's guides deals with only a single application. Each application user's guide contains procedures for installing the application software and for performing all the tasks associated with the application.

Within individual applications, the only difference between the Business System 200 and 300 computers has to do with software installation. Since the Business System 200 and 300 computers use different operating systems, the procedures for installing applications are slightly different for the two systems. Rather than make two different versions of each application user's guide, we figured out a way to have only one version and still avoid the potential problem of a reader using the wrong procedure.

Since the Business System 200 came out first--even before the details of the Business System 300 installation procedures had been worked out--we went ahead and included the Business System 200 installation procedure at the front of each user's guide. Then, when we got ready to ship Business System 300, we prepared what we call a change package for the application user's guides.

There'll be more about change packages later on in this presentation, but for now it's enough to know that the change packages for the application user's guides contain the procedures for installing application software on the Business System 300.

2.3 Programmer's Guides

Even though we anticipated that many Business System customers would prefer to buy ready-made applications, we wanted you to have the option of developing your own programs as well. So we gave you two choices: you could buy the run time system and packaged applications, or buy the development system. The development systems come with various programming language packages determined by the operating system you selected -- DX10 Micro, DX10, or p-System. Four high level languages are available: BASIC, COBOL, Pascal, or Fortran. Each language development package includes a programmer's guide, operating system manuals, and an error message manual.

If, for example, you bought the COBOL language package on the DX10 Micro operating system, you would get four DX10 Micro manuals. The first, the user's guide, is intended for everyday use and easy reference. It explains how to use the Text Editor, how to create and maintain files, and how to use all the commands. The second, the COBOL programmer's guide, assumes that you have read the user's guide and know how to program in COBOL. It describes the enhancements TI has made to ANSI standard COBOL and shows you the most efficient ways to develop and run COBOL programs on your Business System 200 computer. The third, the operating system handbook, assumes that the reader is an experienced assembly language programmer. This manual describes the nuts and bolts (so to speak) of the DX10 Micro operating system: how to perform device and file I/O, how to write new commands, and how to debug assembly language programs. The last manual, on error messages and codes, lists the DX10 Micro error messages and suggests recovery procedures.

3. DOCUMENT USE

Unlike most other documentation published by TI, the Business System series has had to address readers who do not happen to be data processing professionals. We had to consider how much you really need to know before you can begin using a computer as part of your work. We also had to decide how to present this information in a way that would be easy to find and easy to use. We eventually developed a style and format based on four elements:

* Equipment tours -- Each module that deals with a piece of equipment begins with some unpacking instructions and then continues with a short tour of the equipment. In the tour, you find out about all of the lights and switches, the important electronic and mechnical parts, and the supplies that go with the equipment. Any time you need to do something with the equipment, you can refer back to the tour to check the nomenclature.

* Step-by-step procedures -- Every repetitive task is covered by an operating procedure. The procedure begins by telling you when it should be performed and what you need to perform it. The rest of the procedure consists of steps, each of which requires a single action on your part. When a procedure involves a piece of equipment that might vary from system to system (such as performing a disk unit self-test), it refers you to another procedure that deals specifically with that equipment.

* Background paragraphs -- Though we have greatly simplified the data processing terminology used in the Business System guides, sometimes we absolutely have to introduce a data processing term or concept. When such a termcomes up, we introduce it in a background paragraph and list it in the glossary.

* Troubleshooting checklists -- Even though the Business System products have been engineered for a high degree of reliability, we must recognize that problems sometimes do occur. To save the you the inconvenience and expense of a service call, we have included troubleshooting checklists for both the equipment and the software. The troubleshooting checklists cover most of the common problems that you can solve yourself and many situations where you can help isolate the problem by collecting data for the service person. We have organized the checklists by symptom so you can quickly find out what to do. With each symptom we give a list of things to check, reserving the service call for the last resort.

4. DOCUMENT MAINTENANCE

To keep your Business System documentation current, you should update your Business System manuals as necessary, keep a log of back up activity, and keep a record of modifications made to your applications. To suggest improvements to future manuals, send in the User's Response Sheet at the end of each manual.

4.1 Change Packages

TI makes all changes to the Business System manuals and software updates available to you through your dealer. If you buy an updated version of your operating system or application software, you may also receive updated documentation. Sometimes an entire manual will be revised. In that case, simply discard the old manual. However, when changes to the documentation are slight, TI issues a change package, as we did to include the installation instructions for the Business System 300 application packages. Each change package includes replacement pages and an explanatory cover letter. When you get a change package, simply remove the obsolete pages from your manual and substitute the new ones.

4.2 Back up Log

Your back up log keeps track of when you made copies of your software. Your log should list the date, the time, and the volume name of the back up disk. If it takes more than one volume for the back up, each volume must have the same name; we recommend that you number the volumes manually to avoid confusion.

If you keep two generations of back ups, give them different volume names. For example, if you back up your accounting applications at the end of each month, you might name the current backup volume APRIL83 and the next month's MAY83.

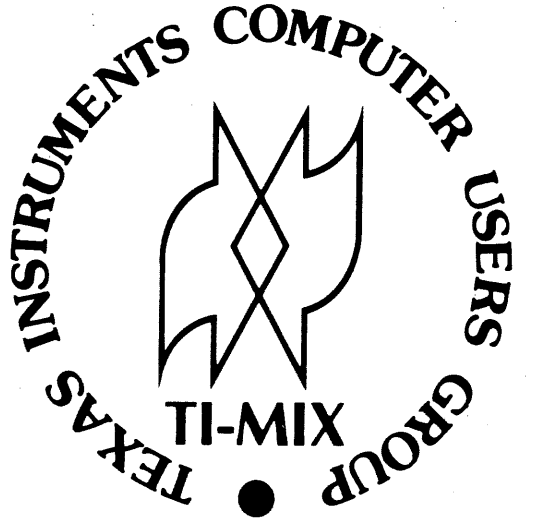
How frequently you should back up your files depends on your application. Copying your software regularly ensures that you have a recent copy of your application data. Then, in case of data loss or damage, having a recent backup of your information makes data recovery easier.

4.3 Application Notes

Keeping application notes makes it easier for you to tailor your application package to the needs of your business. Many applications allow you to select options when running specific programs. Write down in your manual, or in a notebook kept for this purpose, how you respond to options in the program. For example, if you were running an accounts payable program, the system might prompt you to ask whether to post the transactions to the general ledger. You can respond YES or NO; the default response is NO. If you keep a general ledger, you may want to override the default and respond YES. Note this change in your application notes so that the next person running this application -- perhaps your secretary -- will respond appropriately to the prompts.

4.4 User's Response Sheets

Finally, there is something you can do for us: return the User's Response Sheet located at the back of each manual. Each letter goes back to the Publications department, where it is given to the writer working on the next release of that manual. Writers need to know what is unclear or incomplete in order to improve your Business System documentation.


TEXAS INSTRUMENTS COMPUTER USERS GROUP
TI-MIX