AFML-TR-77-145

Standards for Computer Aided Manufacturing

Office of Developmental Automation and Control Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

January 1977
Final Technical Report, March—December 1977

Manufacturing Technology Division
Air Force Materials Laboratory
Wright-Patterson Air Force Base, Ohio 45433

NOTICES

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the Government may have formulated, furnished, or in any way supplied the said drawing, specification, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specified document.

This final report was submitted by the National Bureau of Standards under military interdepartmental procurement request FY1457-76-00369, "Manufacturing Methods Project on Standards for Computer Aided Manufacturing."

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER:

Dennis E. Wisnosky

DENNIS E. WISNOSKY
Manager, ICAM Program Office
Manufacturing Technology Division
Air Force Materials Laboratory

AIR FORCE/56780/6 February 1978 — 400

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFML-TR-77-145	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE *(and Subtitle)*: STANDARDS FOR COMPUTER AIDED MANUFACTURING	5. TYPE OF REPORT & PERIOD COVERED: Final Report April 1976-December 1976
6. PERFORMING ORG. REPORT NUMBER: NBSIR 76-1094(R)	
7. AUTHOR(s): Dr. John M. Evans, Jr., et. al.	8. CONTRACT OR GRANT NUMBER(s): FY145776-00369
9. PERFORMING ORGANIZATION NAME AND ADDRESS: National Bureau of Standards, Department of Commerce, Washington, D.C. 20234	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 78011F, 6 CAM 9999
11. CONTROLLING OFFICE NAME AND ADDRESS: ICAM Program Office, Air Force Materials Laboratory, Wright-Patterson Air Force Base, OH 45433	12. REPORT DATE: June 1977
	13. NUMBER OF PAGES: 352
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

Distribution is limited to U.S. Government agencies only; Test and Evaluation Data; Statement applied November 1976. Other requests for this document must be referred to AFML/LTC,Wright-Patterson AFB, OH 45433

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

CAM architectures; computer aided manufacturing; computer systems; standards; system integration; voluntary standards.

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report identifies and evaluates those existing and potential standards which will be useful to the Air Force in the development and implementation of integrated computer aided manufacturing (ICAM) systems. Such systems, when implemented by the Air Force and by Air Force contractors, will increase productivity in discrete part batch manufacturing by several thousand percent. The use and

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

(Block 20 - Continued)

importance of standards are considered in the context of CAM systems. Since the Air Force will develop the detailed ICAM architecture after this study is complete, existing system concepts and architectures are examined to identify the common elements to guide the further presentation and discussion of relevant standards.

PREFACE

This report covers the results of work accomplished during a nine month sponsored program with the Air Force Materials Laboratory, Air Force Systems Command, Wright Patterson Air Force Base, Ohio under Military Interdepartmental Procurement Request FY145776-00369. Air Force Project Engineers were Captain Dan Shunk and Mr. William A. Harris.

The primary authors of this report are: John M. Evans, Jr., Ph.D.- Project Manager, Joseph T. O'Neill, John L. Little, George E. Clark, Ph.D., James S. Albus, Ph.D., Anthony J. Barbera, Ph.D., Bradford M. Smith, Dennis W. Fife, Ph.D., Elizabeth N. Fong, David E. Gilsinn, Ph.D., Frances E. Holberton, Brian G. Lucas, Ph.D., Gordon E. Lyon, Ph.D., Beatrice A. S. Marron, Mable V. Vickers and Justin C. Walker.

TABLE OF CONTENTS

INTRODUCTION

The Air Force is initiating a mjaor new program to accelerate the establishment of Integrated Computer Aided Manufacturing (ICAM) in discrete part batch manufacturing industries in the United States, especially in the serospace industry. The National Bureau of Standards is providing support to that program by analyzing existing standards relevant to Integrated Computer Aided Manufacturing.

This document is the final report to the Air Force Manufacturing Technology Division of the Air Force Materials Laboratory at Wright-Patterson Air Force Base on the ICAM support project. This report covers all five tasks of the project as are defined in Appendix A.

Task 1 Identify current standards applicable to CAM.
Task 2 Analyze existing formal and de facto standards.
Task 3 Assess the actual usage of standards in industry.
Task 4 Recommend optimal standards for CAM system development.
Task 5 Identify standards organizations and outline a proper Air Force role in standards activities.

This report identifies those existing and potential standards which will be useful to the Air Force in the development and implementation of integrated computer aided manufacturing systems. Such systems, when implemented by the Air Force and by Air Force contractors, will increase productivity in discrete part batch manufacturing by several thousand percent.

The NBS effort provides a comprehensive reference data base on all formal and de facto standards that are considered to be relevant to the Air Force Program. Summary data sheets included in the report form an annotated bibliography on each standards activity for ease of reference.

The report examines the utility of these standards to the Air Force Program and in each relevant standards area recommends a best approach to follow either toward adopting existing standards or toward developing needed standards.

Finally the report outlines the proper role of the Air Force in standards activities. Recommendations are made for a comprehensive and rational approach to computer integrated manufacturing based upon the use of formal standards, definitive technical guidelines, and precise ICAM program policy. These recommendations are made in a framework of programmatic objectives that NBS believes to be essential for the success of the ICAM program.

The work reported here was supported in part by the Air Force Program for Integrated Computer Aided Manufacturing, Manufacturing Technology Division, Air Force Materials Laboratory, Wright-Patterson Air Force Base under MIPR FY 14577600369, Dennis Wisnosky, Program Manager.

EXECUTIVE SUMMARY

The Air Force is initiating a major new program to accelerate the establishment of Integrated Computer Aided Manufacturing (ICAM) in discrete part batch manufacturing industries in the United States, especially in the aerospace industry. This report describes work done at the National Bureau of Standards (NBS) to analyze those existing standards which are relevant to the ICAM Program and to outline a policy to achieve Air Force programmatic objectives thru the use of standards.

The goal of demonstrating computer integrated manufacturing requires the interaction of a multitude of functional modules some of which are already in limited use, some just conceived, and others not yet developed. Coordinating the development of this variety of technologies from an equally large number of different contractors and doing it in such a way as to have each module fit into the architecture of an integrated CAM system is a formidable task.

The goal, however, becomes achievable when approached through the creative use of formal standards, definitive technical guidelines and precise ICAM program policy. These three techniques will allow the complete definition of interfaces between ICAM modules and their environment. Where interfaces match, modules can fit together and system integration becomes possible.

Standards are an essential component of the ICAM Program; essential for the development of the ICAM modules, essential for their successful integration, and essential for encouraging the widespread use of the products developed. The NBS Office of Developmental Automation and Control Technology has headed a multi-disciplinary team of Bureau personnel skilled in many facets of computer technology as it will be applied to manufacturing systems. In a ten month effort culminating in January 1977 this team investigated in manufacturing and documented the utility of standards in the Air Force Program. Standards for computers can be logically divided into ten areas - each documented in a separate chapter of this report.

For these areas the report first identifies those existing and potential standards which are useful to the Air Force in the development and implementation of integrated computer aided manufacturing systems. Next it provides a comprehensive reference data base on all formal and de facto standards. Each standard is analyzed as to its benefits, limitations and degree of actual usage.

The report discusses the utility of these standards to the Air Force Program and in each relevant standards area recommends a best approach to follow either toward adopting existing standards or toward developing needed standards.

Finally the report outlines the proper role of the Air Force in standards activities. Recommendations are made for a comprehensive and rational approach to computer integrated manufacturing based upon the use of formal standards, definitive technical guidelines, and precise ICAM program policy.

So vital are standards to the successful realization of ICAM program objectives that a Standards Office should be created as a focal point for coordination, documentation, and standardization activities. Functions of this office would include:

Coordination with the NASA effort on Integrated Program for Aerospace Vehicle Design (IPAD) to mutually define the CAD/CAM interface;

Cooperation and support of various voluntary standardization activities and of the development of FIPS and MILSPEC standards when necessary;

Development of local ICAM program guidelines where necessary for areas such as documentation and programming standards;

ICAM data base management system administration;

Maintenance and distribution of system definition, software tools, and documented ICAM software.

In developing a proper role for standards in the Air Force program it is instructive to look beyond the ICAM technical goals to identify four programmatic objectives that are essential for success.

Portability of Software
Integratibility of Modules
Distributed Data Processing
Exchangeable Manufacturing Data

With these points in mind it is possible to examine how the creation and adoption of certain standards, guidelines and policy will contribute to achieving ICAM goals. Table] summarizes these various relationships and presents a suggested Air Force position on the effective use of interface standards. Table 2 briefly details the content and utilization of the standards addressed in this report. Data in both Tables are refined in greater detail by the various sections that follow. Collectively this report represents an indepth analysis of the importance of interface standards to the evolution of computer based manufacturing systems.

	POLICY	GUIDELINES	STANDARDS
Software Portability	Develop all software in high level programming languages. Allow assembly language only in rare, carefully controlled conditions. Encourage the use of standardized programming languages. Formulate a definitive written statement on portability of applications software.	Develop recommended programming practices. Specify applications program documentation guidelines.	Support expeditious establishment of a Federal Fortran Standard. Utilize ANSI and FIPS standard COBOL for all COBOL programming.
Integratable Modules	Develop functional specs for a competitive procurement of a DBMS for all near term ICAM software. Appoint a Data Base Administrator within the ICAM Standards Office. Develop specifications for ICAM system software.	Develop guidelines for use of DBMS. Distribute guide to ICAM systems software to all contractors.	Initiate participation in FIPS Task Group 24 to consider needs for Federal data base standards. support ANSI S3/SPARC study group to identify need for ANSI standards.
Distributed Data Processing	Develop and implement ICAM with fourth generation computer systems in mind. Specify host level and process level communication protocols. Consider impact of network conception on procurement practices.	Develop guide to communications interface standards, link level protocols and network level protocols. Devise security requirements for distributed computer systems. Advise contractors on implementing host level and process level protocols.	DTE/DCE Interface: RS 232-C RS XYZ CCITT X.21 Link Level: ANSI ADCCP Network Level: CCITT X.25 Communication Codes: ANSI X3.4
Exchangeable Manufacturing Data	Develop policy on specification, purchase, data rights, and exchange of digital manufacturing data. Maintain close liasion with IPAD, ANSI Y14.26 and CAM-I.	Support development of interface with NC manufacturing to allow easily transportable digital data packages.	Use APT or APT and COMPACT II for NC programming. Use IPC-D-350A for future wedges relating to electronic systems.

SUMMARY MATRIX CAM STANDARDS

Standard Class	Major Stds.	Standard Designation	Conflicts	Implementations	Comments
A. NC Programming Languages	*	X3.37 (APT)	COMPACT II	Widespread	Most comprehensive NC language.
	*	COMPACT II/ACTION/SPLIT	APT	Widespread	Easier than APT for simple parts; no formal standard yet.
B. CAD/CAM Interface	*	IPAD (Integrated Program for Aerospace-Vehicle Design	None	None; in development	IPAD will set defacto standard; conflicts may develop.
		IPC-D-350A (Printed Circuit Boards)	None	Widespread	Tutorial example; PC boards only.
	*	ANSI Y14.26.1 (Digital Representation of Physical Object Shapes)	None	None; in development	Intended for transfer of part geometry data.

SUMMARY MATRIX COMPUTER STANDARDS

Standard Class	Major Stds.	Standard Designation	Conflicts	Implementations	Comments
A. Communications 1. Hardware	*	EIA RS-232-C (Serial Binary Data Interchange)	CCITT X.21	Universal	For connection of digital terminal to analog communication system. Used for CRT's, NC tools, other periphals and communications.
	*	EIA SP-1194A (Data Terminal Data Communication Interface) (RS XYZ)	Will supersede RS-232-C	Future	Will supersede RS-232-C, IC compatible, higher data rates. Known as RS XYZ until adopted.
	*	CCITT X.21 (Digital Data Terminal/Data Communication Interface)	RS-232-C, RS XYZ	Limited But Increasing	Defines interface to digital communications system. Will be important by 1980's with digital telephone system. Will be adopted by ANSI.
		ANSI X3.24 (Signal Quality)	None	Almost Universal	
		EIA RS-408 (NC/DTE Interface)	IEEE 488	Extensive	Byte serial. Used with NC tools, paper tape interfaces
		IEEE 488 (Programmable Instrumentation Interface)	EIA RS-408 and IEEE 583	Limited	Instrumentation uses.
		IEEE 583 (CAMAC)	EIA RS-408, RS-232-C & IEEE 488	Limited But Increasing	Instrumentation uses in Nuclear Field
		FIPS PUB 22-1 (Sychronous Signaling Rates)	None	Extensive	
		FIPS PUB 16 (Bit Sequencing)	Numerous	Extensive	
		FIPS PUB 17 (Serial-by-Bit ASCII)	Some Proprietary	Extensive	Bit Sequence specified in FIPS PUB 16
		FIPS PUB 18 (Parallel-by-Bit ASCII)	Some Proprietary	Most Parallel-By-Bit	
		FIPS PUB 37 (Synchronous High Speed Signaling Rates)	None		
		EIA RS-422 (FED-STD-1020; Balanced Voltage Circuits)	None	Limited	Electrical standard for high speed RSXYZ
		EIA RS-423 (FED-STD-1030; Unbalanced Voltage Circuitry)	None	Limited	Electrical standard for low speed RSXYZ
		CCITT V.28 (Unbalanced Double-Current Circuits)	EIA RS-232-C	Europe	
		CCITT V.31 (Single-Current Circuits)	None	Europe	

SUMMARY MATRIX COMPUTER STANDARDS CONT.

		ANSI X3T92/064 (Proposed Minicomputer Interface)	None	Limited	For minicomputer peripherals.
		ANSI X3T9/600 (Proposed Channel Interface)	None	Widespread	Large computer peripherals interface.
	2. Codes				
	*	FIPS PUB 1 (ASCII)	Numerous	Extensive	Fundamental Standard, Conflict with EBCDIC
		ISO 646 (7-bit Code)	Numerous	Extensive	
		CCITT V.3 (International Alphabet)	Numerous	Extensive	
	*	IBM CSS 3-3220-002 (EBCDIC)	Numerous	Extensive: IBM	In conflict with ASCII
		Encryption Algorithm	None	Some Future	For secure data transmission
	*	EIA RS-358 (NC ASCII Subset)	EIA RS-244A	All Newer, NC Tools	Code for NC Data. RS-244A (Flexomoniter code) no longer supported by EIA
	*	EIA RS-274-C (NC Variable Block Format)	None	Extensive	Current NC Data Format Standard
	*	EIA SP-1177A (Advanced NC)	None	All US NC Tools in Future	Future NC (CNC) Data Formats
		FIPS PUB 36 (ASCII Graphics)	None	Some	
		FIPS PUB 14(Hollerith Code)	IBM System 3	Extensive	Punched Card Code
		FIPS PUB 2(PPT Code)	Teletypesetter & other older codes	Extensive	Punched Tape Code
	3. Protocol (Link Level)				
		ANSI X3.28 (Character Oriented	IBM BISYNC, ADCCP	Many, Incompatable	The standard lists some 40 incompatable implementations.
	*	ANSI S3534/589(ADCCP)	ANSI X3.28, IBM BISYNC, DEC DDCMP	Near Future, Extensive	Proposed Bit Oriented Standard. Compatible with IBM SDLC, ISO HLDC
		DEC DDCMP	ANSI X3.28, ADCCP, SDLC, BISYNC	Some DEC	Corporate standard. Conflict with ADCCP
	4. Protocol (Network Level)				
		IBM SNA	CCITT X.25, DNA, ARPANET	Some IBM	Not intended as standard.
	*	CCITT X.25	SNA, DNA	Some, extensive in future	Draft standard. Incorporates ADCCP. Major packet network standard.
		ARPANET IMP/HOST			De facto standard. Commercial networks will follow X.25.
		DEC DNA	SNA, CCITT X.25, ARPANET	Partial DEC	
	B. Computer Programming Languages 1. General Purpose				
		ANSI X3J2/76-01 (BASIC)	None	Extensive	Proposed standard. Time sharing use.
		FIPS PUB 21-1 (COBOL)	None	Extensive	Widespread business use
		ANSI X3.9 (FORTRAN)	PL/I	Extensive	Widespread scientific, engineering use

SUMMARY MATRIX COMPUTER STANDARDS CONT.

		MDC/28, 33 & 34 (MUMPS)	None	Extensive	Proposed ANSI standard. Used for inventory control.
		PASCAL	None	Some	Academic Popularity; Not a standard.
	*	ANSI BSR X3.53 BASIS/ 1-21 (PL/I)	COBOL, FORTRAN	Some, large machines	Current standard does not define subsets. Only standardized "modern" language.
2. Simulation		(SUMMARY SHEET)			No standards.
3. Machine Oriented System Implementation		BLISS	All SILS are competitive	Limited (DEC)	No standards.
	*	PL/S		Some IBM	IBM system language; propriety; extension of PL/I
		BCPL & C		Some DEC, IBM, Honeywell	Examples of modern, user oriented SILS.
4. Artificial Intelligence	*	PL/M, PL/M6800 and MPL (SUMMARY SHEET)		Some DEC, IBM	Subsets of PL/I for Cross Software support for microcomputers. No standards.
C. Data Base Management Systems 1. CODASYL DBTG		CODASYL DBTG	All DBMS approaches are competitive.	Limited But Increasing	COBOL based. Closest to formal standard. Network structure important.
2. Non-CODASYL Self-Contained		(SUMMARY SHEET)		Extensive	Commercial, self contained DBMS systems
3. Non-CODASYL Host Language		(SUMMARY SHEET)		IBM	Examples are IMS and TOTAL. Hierarchical structures. Extensions to programming languages
4. Relational Approach		(SUMMARY SHEET)		Future	Research area.
D. Operating Systems		(SUMMARY SHEET)			No standards.
E. Validation and Testing 1. Compiler/Interpreter	*	(SUMMARY SHEET)	None	COBOL, Others in Future	Important means for insuring portability
2. Mathematical Software		(OVERVIEW)	Some IBM UNIVAC	Limited	
F. Documentation 1. Program & System	*	FIPS PUB 38 (Program Doc'n)	None	Extensive	Documentation standards should be mandatory
		FIPS PUB 30 (Software Summary)	None		ANSI X3K7 developing abstract form
		FIPS PUB (Flowchart Symbols)	None		
2. Data Elements	*	(SUMMARY SHEET)			Existing data element standards not CAM oriented
3. Vocabulary/Nomenclature		FIPS PUB 11/(General ADP)	None		ANSI developing new dictionary
		ANSI X8.1 (AXIS & MOTION)	None	Extensive	

SUMMARY MATRIX COMPUTER STANDARDS CONT.

4. Representation of Values		ANSI X3.42 (Numeric Values)	None	Becoming Universal	Important for intersystem compatibility
G. Media					
1. Punched Cards	*	ANSI X3.26 (Hollerith Punched Card Code); ANSI X3.11 (Specification for General Purpose Paper Cards for Information Processing); ANSI X3.21 (Rectangular Holes in 12-Row Punched Cards)	96-Column IBM	Almost Universal	Should be mandatory if cards are used
2. Magnetic Tape	*	EIA RS-346 (Type A Hubs and Reels and Magnetic Tape);	None	Universal	
		ANSI X3.14 (Recorded Magnetic Tape for Information Interchange (200 CPI, NRZI)); ANSI X3.22 (Recorded Magnetic Tape for Information Interchange (200 CPI, NRZI)); ANSI X3.40 (Unrecorded Magnetic Tape for Information Interchange (9-Track 200 and 800 CPI, NRZI, and 1600 CPI, P.E.)); ANSI X3.39 (Recorded Magnetic Tape for Information Interchange (1600 CPI, P.E.)); ANSI X3.54 (Recorded Magnetic Tape for Information Interchange (6250 CPI, GCR));	Tape Standards are in conflict.	Universal	Magnetic Tape Standards will be a necessity.
	*	FIPS PUB 25 (Recorded 1600 CPI)	Non-ASCII formats	Extensive	Recommended Federal Standard.
		ANSI BSR X3.48 (Cassettes)	None	Extensive	
		ANSI BSR X3.56 (Cartridge)	None	Some	
3. Paper Tape		ANSI X3.6 (Perforated Tape Code for Information Interchange);	None	Extensive	Widely used in NC; not recommended for other uses.
		ANSI X3.18 (One-Inch Perforated Paper Tape for Information Interchange);	X3.19	Extensive	Paper tape not recommended.
		ANSI X3.19 (Eleven-Sixteenths Inch Perforated Paper Tape for Information Interchange	X3.18	Limited	
		ANSI X3.29 (Specifications for Properties of Unpunched Oiled Paper Perforator Tape); ANSI X3.20 (Take-Up Reels for One-Inch Perforated Tape for Information Interchange			

STANDARDS IN CAM SYSTEMS

THE IMPORTANCE OF STANDARDS IN CAM

The applications of computers to the control of machines represents a fundamental revolution in manufacturing technology. This trend, which first started with numerically controlled machine tools, has its logical extension in a totally automatic factory which can run seven days a week without human intervention and can automatically reconfigure itself to produce custom designed products at mass production costs.

Numerically controlled machine tools are now widely used and understood in manufacturing in a stand alone context, even for small shops. There is a new technology emerging in which general purpose NC machine tools are integrated with automated materials handling systems and higher level computers for planning, scheduling and control. Such systems are able to perform machining operations on randomly sequenced parts.

Such systems range from the Sundstrand Omniline and the Kearney and Trecker Flexible Manufacturing System, which are operational at Ingersoll-Rand, Caterpillar and Allis Chalmers, respectively, to the proposed Japanese Methodologies for Unmanned Manufacturing Systems program. The Japanese program will result in a prototype automatic factory for the production of machine tool parts. The predicted increases in productivity in this facility will be 7000 to 8000 percent compared with a conventional facility.

The key to productivity increases in such manufacturing systems is the coupling of the various elements of the manufacturing process into an integrated system with a centralized computer control system and data base structure.

In a larger context, the integration of computer aided design with computer aided manufacturing, that is, systems in which design of the part creates a geometric data base which directly results in control programs for the NC machine tools, and in which computer systems are used in process planning, inventory control, scheduling, and control of production, offer even larger problems of development and integration of many dissimilar parts. There is now no company offering complete systems on the market.

The largest manufacturing industries have created their own internal integrated systems which are specific to their needs and which are generally held as proprietary and not placed on the market. One can go to a machine tool company and buy NC tools, to a computer service company and buy time sharing support for programming those tools, and one can go to many sources and buy special computer programs or computer programming support for special applications. Further, one can buy stand alone graphic systems including integral software programs, but often the problems of putting together an entire CAD/CAM system for all of these components involves so much engineering and software development effort as to make such systems practically and economically viable for only the largest firms.

The key to developing and using integrated computer aided manufacturing systems in a free marketplace, particularly for medium and small firms who cannot afford special software and engineering, will be the use of adequate standards to insure the compatability of modular system components obtained from competitive suppliers.

TYPES OF STANDARDS

The oldest standards, and the usual standards that one normally brings to mind, are those for weights and measures, that is, units for length, weight, and volume that are the basis for commerce and trade and for science. These standards originally were based on dimensions of the human body, which produced measures that were reproducible wherever there was a man, such as the cubit or the yard, the fathom, the foot, and digit, and so on. These units in turn produced artifacts such as yardsticks. For example, in the 16th Century a measuring rod was defined as follows:

> "to find the length of a measuring rod the right way and as it is common in craft . . . take 16 men, short men and tall ones as they leave church and let each of them put one shoe after the other and the length thus obtained shall be a just and common measuring rod to survey the land with."

Such standards of weights and measures are now generally based on independently reproducible constants of nature rather than artifacts and are mandatory and controlled by law.

The second class of standards are those set for consumer protection, in the public interest, such as standards involved in building codes, in pollution control, and in the flammability of carpets, draperies, and children's sleepwear.

The third class of standards are voluntary industry standards which are set by consensus agreement among concerned parties. There are now some 20,000 voluntary standards in force which have been created by more than 400 organizations, covering a multitude of products, practices, test procedures, materials, and other characteristics which have been found to be in the interest of those parties involved to reach a common understanding and common practice.

The driving force behind voluntary standards is economic. The first step in this direction occurred at the start of the 19th Century when Eli Whitney produced the first rifles from interchangeable parts which obviated the need for handwork in assembly. Production became simpler and less expensive. Mass production is the logical extension of this concept through an entire industry, and it requires standardization through the entire economy. In fact, it is hard to imagine a time during this century when nuts and bolts wouldn't fit together unless they came from the same manufacturer. Such a situation would obviously bring modern assembly lines grinding to a halt.

De facto standards can be created by common usage or by oligopolistic markets such as exist in the computer field. Such standards are usually internal company standards that are picked up by others using or interfacing to that company's products. An example is the existence of a billion dollar independent computer peripherals industry that produces products that interface with EBCDIC as the code for data interchange, in conflict with the ANSI Standard (ASCII). EBCDIC is thus a de facto standard in the industry.

Finally there are internal company standards that have been set by fiat, planning, or historical accident. Such standards are necessary to the efficient operation of any large organization and can become extremely rigid and formalized. An example, is a part or drawing numbering scheme. However, they are generally unique and are not relevant to the Air Force ICAM program.

Why Voluntary Standards Work

Voluntary standards work because of the economic forces that are involved. Standards can be a powerful management tool to improve efficiency and reduce costs.

In 1920, Herbert Hoover initiated a study of six industries as President of the Federated American Engineering Society. The conclusion was that nearly 50 percent of the cost of production and distribution could be eliminated through standardization and simplification. Fifty percent of the costs of production and distribution is worth the large amount of effort needed to reach voluntary standards in any manufacturing industry.

Standards are not necessarily fixed and inflexible. Obviously, any voluntary system, can only work as long as the parties involved believe the standards are useful. When change is appropriate, change occurs under a voluntary system.

Standards for Computer Aided Manufacturing

Most engineers and computer scientists readily agree that modular design is a good philosophy. Indeed, that is the concept that is most appropriate in CAM. The standards that are most needed for computer aided manufacturing systems are the interface specifications that allow all of the various modular components of manufacturing systems, computer programming, and the computer language and standards that will make the software independent of the specific hardware environment. This concept is particularly important to medium- and small-size firms, those below the top 1000 firms who account for 60 percent of the value of shipments in the discrete parts manufacturing industries.

In many cases the existance of an appropriate interface can actually stimulate technological innovation and market place competition. The development of numerical control and APT offer an excellent example of this concept. When NC tools were first installed in the aerospace industries between 1958 and 1960, it was almost impossible to make tapes that ran tools. Each different system had its own tape sizes, data codes, formats, servo characteristics and programming requirements, and the programs had to be figured out with a hand calculator and punched on a Flexowriter. This state of chaos was brought under control by standards that are now controlled by Electronic Industries Association Committee IE-31 on Numerical Control. Standards were necessary for widespread industry use of NC.

The APT language, which was created in this environment as a computer program to prepare the tapes for all of those dissimilar and incompatible machine tools, was created in a two level structure, where the APT processor produced a CL (cutter location) data file which was then run through a post processor to convert it to specific output format for a given machine tool.

This CL tape or CL file has become as excellent interface standard for computer aided manufacturing systems. For example, the interactive graphics systems now available from several suppliers can produce a CL data file which can be directly run through existing post processors. At this point, from the point of view of information flow in a computer control hierarchy, all of the machine tools are functionally equivalent. In fact, one new CNC system takes CL file as direct input.

This concept of standard interfaces is very powerful and if extended into a total manufacturing system, would allow the possibility of creating a system out of modules bought from competitive manufactures. Alternatively, large aerospace prime contractors could create proprietary application modules to maintain their competitive position without the total cost of overall system design and implementation.

In addition, if the system modules are written in standard languages that are machine independent, then the modules can be transferred to other users and be integrated into a total system without special engineering or software development.

The Air Force has recognized these concepts in the Statement of Work for the NBS support project. The next section will consider the identification of the standards relevant to the Air Force ICAM program.

STANDARDS RELEVANT TO CAM

There are three distinct architectures involved in a CAM system. While these are defined in more detail by Appendix B, the standards which are relevant to these architectures are discussed here.

Examples of Standards relevant to Manufacturing:

- drawing specifications
- fastener standards
- tooling standards
- safety standards
- pallet standards
- part numbering conventions
- quality control standards

Examples of Standards relevant to CAM:

- data base formats
- NC part programming languages
- group technology coding schemes
- standard process plans
- inventory
- programming languages for robots and automatic test equipment
- CAD/CAM interface

Examples of Standards relevant to Computer Systems:

- communication codes and protocols
- documentation standards
- programming languages
- data base management systems
- media standards.

In considering the standards relevant to the Air Force ICAM program, NBS has made the following decisions:

1. No manufacturing standards will be considered. Most of these standards are internal company standards. Hence arbitrary decisions by the Air Force or ICAM contractors will become valid de facto standards. Better yet, every effort should be made to make the ICAM software independent of such detailed data formats so that companies may continue to use internal standards for such things as part numbering. This will maximize the utility of ICAM software. Standards on hardware (tooling, fasteners) will be well known by the designer of new equipment under the ICAM program and can best be selected at the time of implementation. Material will be provided to the Air Force separately on the metric issue.

2. In the CAM area, only NC part programming languages and the interface with design will be considered. There are formal standards in both of these areas. Group technology codes will not be considered since that is the specific subject of study under the ICAM program. Programming languages for robots are not standardized, and ATE and process control languages are oriented to electronic testing and continues process control rather than discrete part batch manufacturing, which includes air frame manufacturers.

   Again, because of lack of formal standards, the Air Force can make decisions that will result in de facto standards but should strive to create a system that will allow each user maximum flexibility in implementation. The CAM-I process planning system (CAPP) is an example of this concept since it allows use of any group technology coding scheme.

3. Standards for computer systems will receive the bulk of the attention of this report. Three key concepts will be pursued in evaluating standards:

   - System integration: data interfaces between CAM application programs
   - Software portability: interfaces between CAM programs and host computer systems
   - Distributed processing capability: interfaces between computers in distributed system

   The second concept is the key to widespread utilization and impact of the Air Force program. The third concept, distributed processing capability, will be important if ICAM is to remain relevant over any length of time. The rapidly dropping costs of mini and microcomputers and the availability of software to create and utilize distributed processing and distributed data structures indicate that the next generations of computer systems will be distributed in nature.

   Within this set of assumptions, then, the standards that are relevant to the Air Force ICAM program, in priority order are:

   A. Communications Standards
   B. Programming Languages
   C. Documentation
   D. Validation and Testing
   E. Media Standards

   In addition, standards and common practices for operating systems and data base management systems will be discussed to identify the technical issues involved system integration.

THE AIR FORCE ROLE IN STANDARDS

INTRODUCTION

- ICAM Standards Office
- Existing Standards Organizations
- Programmatic Objectives

PORTABILITY OF SOFTWARE

- Software Development Philosophy
- Standard Programming System Definition
- System Validation
- System Portability Requirements
- Programming System Implementation
- System Primer and Reference Manual
- Documentation Guidelines and Aids
- System and Application Program Testing
- Summary
- Recommendations

INTEGRATABLE MODULES

- Applications Program Interfaces
- Data Base Interfaces
- System Interfaces
- Peripheral Interfaces
- Recommendations

DISTRIBUTED DATA PROCESSING

- Standards for Distributed Systems
- Recommendations

EXCHANGEABLE MANUFACTURING DATA

- Engineering Drawings
- Digital Data Package
- Recommendations

The goal of complete computer integrated manufacturing throughout the aerospace industry requires the interaction of a multitude of functional modules some of which are already in limited use, some just conceived, and others not yet developed. Coordinating the development of this variety of technologies from an equally large number of different contractors and doing it in such a say as to have each module fit into the architecture of an integrated CAM system would appear to some to be an impossible task.

The goal, however, becomes achievable when approached through the creative use of formal standards, definitive technical guidelines and precise ICAM program policy. These three techniques will allow the complete definition of interfaces between ICAM modules and their environment. Where interfaces match, modules can fit together and system integration becomes possible.

Standards are an essential component of the ICAM Program; essential for the development of the ICAM modules, essential for their successful integration, and essential for encouraging the widespread use of the products developed. They are primarily an arbitrary solution to a recurring problem. Therefore, it is important that any standards evolved by ICAM be relevant to the solution of problems as they are normally perceived and acted upon by various classes of users. Equally important is that they be sufficiently well documented that they can be easily understood and implemented by potential suppliers of the products.

It should be remembered that the standardization of a product, CAM module, interface or technique is infinitely more important than standardization on it. In other words, the establishment of an acceptable functional definition of the product is the most difficult and meaningful part of standardization. If this part is done well, that is if it results in broad implementation and use, then informed management decision policy can be formulated. Policy can be set regarding the strictness or looseness of the standards enforcement and the degree to which a product's use must be considered mutually exclusive in relation to the use of similar products. Therefore, standardization effort in the ICAM Program must be viewed as an activity which will create management options by making nonstandard things more understandable, usable, and controllable rather than an activity which limits options.

ICAM Standards Office

So vital are standards to the successful realization of ICAM program objectives that a Standards Office should be created as a focal point for coordination, documentation, and standardization activities. Functions of this office would include:

- Coordination with the NASA effort on Integrated Program for Aerospace Vehicle Design (IPAD) to mutually define the CAD/CAM interface;
- Cooperation and support of various voluntary standardization activities and of the development of FIPS and MILSPEC standards when necessary;
- Development of local ICAM program guidelines where necessary for areas such as documentation and programming standards;
- ICAM data base management system administration;
- Maintenance and distribution of system definition, software tools, and documented ICAM software.

All subsequent recommendations assume an ICAM Standards Office. The minimum staff of such an office would be three professionals: one responsible for the CAD/CAM interface, one would be the Data Base Administrator, and one responsible for maintenance of the system definition and documented ICAM software. Additional clerical staff may be needed as the ICAM program matures.

The ICAM Standards Office will have to contend with three general types of relevant system standards, namely,

True standards that exist and must be maintained, improved, corrected, reimplemented, tested, documented, etc.;

Those that do not exist and that therefore must be created, then maintained;

De facto standards which are really not standards at all but which work and therefore require practical consideration.

All three types will require a commitment of resources if they are to remain or become responsive to ICAM requirements. Table 1 summarizes the various standardization efforts to direct interest to the ICAM program and the degree of Air Force commitment which is recommended by NBS. This commitment need not be of direct involvement of the program staff. Rather, it would be wise and more effective to arrange expert technical representation on the various standardization activities. The technical representatives could intelligently and authoritatively express ICAM needs and be in a prime position to analyze the best use of the developed standard in ICAM products. This function would be given the emphasis of a primary mission and not as a collateral duty assigned to a member of the ICAM staff.

Active Standards Organizations

Voluntary industry standards for products and engineering procedures are set by consensus agreement among concerned parties. There are now some 20,000 voluntary standards in force which were created by more than 400 organizations, covering a multitude of products, practices, test procedures, materials, and other characteristics which were found to be in the interest of parties involved to reach a common understanding and a common practice.

The driving force behind voluntary standards is economic. In a study that was done in the 1920's on standardization during World War I it was found that the cost of production and distribution could typically be reduced as much as 50% through standardization. Cost figures that are comparable to that figure are possible in dealing with computer aided manufacturing systems.

Standards for numerical control and CAM are voluntary standards. The second interim report has identified 64 standards which apply to computers in manufacturing. These standards have been developed by several different organizations:

Electronic Industries Association

The EIA has an engineering committee (IE-31) on numerically controlled machines. This committee has been in existence throughout most of the history of numerical control, and is responsible for those standards that will stimulate the interchange of information between different NC systems, particularly paper tape size, command and data formats, and electrical interfaces between numerical controls and machine tools. The committee has attempted not only to standardize de facto marketplace practices, but to play a role of leadership by issuing bulletins that point out preferred directions for design and implementation of various aspects of machine tool control systems.

Table 1 - Participation on Standards Groups

Technical Area	Standards Group	Degree of Participation	Technical Objectives
COMMUNICATIONS CODES	ANSI X3.4	Monitor	Maintain awareness of basic ASCII code.
PROGRAMMING LANGUAGES			
FORTRAN	ANSI X3J3	Monitor	Insure 1977 revision is made available from ICAM.
COBOL	ANSI X3J4	Monitor	Extract methodology and guidelines for input to programming policy.
	FIPS TG-9	Monitor	
PL/I		Monitor	Define useful subsets to run on smaller computers.
PASCAL			
DATA BASES			
CODASYL	ANSI/X3/SPARC	Monitor	Identify the need for ANSI Standards in DBMS.
	FIPS TG-24	Active	Provide ICAM requirements for DBMS in a distributed system environment.
NC LANGUAGES			
APT	ANSI X3J7	Active	Influence Complete Specification of language for portability.
	FIPS TG-19	Active	Develop Federal Standard and guidelines for use.
COMPACT	ANSI X3J5	Review	Maintain awareness and watch for CLTAPE output.
CAD/CAM INTERFACE			
Physical Object Description	ANSI Y14.26	Monitor	Analyze for use in ICAM's interface with Design.
	CAM-I Geometric Modeling	Monitor	
Electronics	IPC	Review	Evaluate developments for use with Avionics systems.
General	NASA IPAD	Monitor	Resolve any incompatibilities which are noted.
COMMUNICATIONS			
Data Terminal Equip	IE-30	Active	
Peripheral Interfaces	ANSI X3T9	Active	
Link Level Protocols	ANSI TG-4 X3S3	Monitor	
Packet Network Protocols	ANSI X3S3	Monitor	
Host to Host Protocols	None		
DNC Interface	IE-31	Active	Help refine interface descriptions.

The EIA committee has set standards on character codes; formats for numerical control tapes; and interfaces between numerical controls and machine tools, peripheral equipment, and higher level computers.

National Machine Tool Builders' Association

The NMTBA develops various publications relevant to the manufacture, procurement and utilization of machine tools. These include a Directory of NC Machine Tools and Related Products, an NC Character Code Cross Reference Chart, a Definition and Evaluation of Accuracy and Repeatability of NC Machine Tools, Common Words as They Relate to NC Software, and Guidelines for the Measurement of Machine Tool Utilization.

Note: For a listing of standards on the components of machine tools and on the design, manufacture and sales of machine tools, the reader is referred to the Standards Index published by NMTBA.

American National Standards Institute

The American National Standards Institute is the nationally recognized coordinator of voluntary standards development and the clearinghouse for information on national and international standards. Its federated membership includes some 180 voluntary organizations representing virtually every technical discipline, every facet of trade and commerce, organized labor, and consumer interests. These organizational members join with some 1000 individual companies, representing both large and small business, and with representatives of government at federal, state, and local levels in programs dedicated to meeting this country's needs for national consensus standards through voluntary action.

ANSI's approval procedures for recognizing standards as American National Standards ensure a consensus of affected interests. Its requirements for due process and the right to appeal actions at several levels of review establish confidence in, and credibility for, the standards it approves. It provides and administers the voluntary system through which standards, no matter what their origin, may be recognized and accepted nationally and internationally.

Whereas anyone may submit proposed American National Standards to the Institute, the Institute recognizes only three methods for the development of evidence of consensus for approval of American National Standards. These are the Accredited Organization Method, the American National Standards Committee Method administered by the ANSI X3 Committee, and the Canvass Method.

Computer Aided Manufacturing-International

CAM-I is a non profit organization devoted to advancing the state-of-the-art in computer aided manufacturing. CAM-I has an active standards committee. Current projects include a glossary of CAM terms and an analysis of the architecture of CAM.

International Organization for Standards

The ISO is a worldwide federation of national standards institutes (ISO Member Bodies). The work of developing International Standards is carried out through ISO Technical Committees. Every Member Body interested in a subject for which a Technical Committee has been set up has the right to be represented on the Committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. The United States is one such Member Body and is represented on the various ISO committees by ANSI.

Draft International Standards adopted by the Technical Committees are circulated to the Member Bodies for approval before their acceptance as International Standards by the ISO Council. ANSI generally adopts in total those ISO standards it feels have merit. It is recommended that the Air Force should consider ISO standards for informational purposes only and not for adoption.

International Consultive Committee on Telegraph and Telephone

CCITT is the French abbreviation for the International Consultive Committee on Telegraph and Telephone which sets standards primarily in the field of communications. In most nations of the world CCITT recommendations are given the force of law. This is not true of the USA. The CCITT is an organ of the International Telecommunications Union (ITU) which is reported to be the oldest international standardizing body in the world. The ITU is now an organ of the United Nations. The USA is represented on the CCITT by the the US State Department in contrast to the ISO where the USA is represented by ANSI.

Society of Manufacturing Engineers

The SME has a standards committee that is currently evaluating standards for computer aided manufacturing systems. Present projects include a newsletter of CAM standards and an analysis of the architecture of CAM. SME publications include technical papers, reports and text books on NC and CAD/CAM.

Numerical Control Society

The Numerical Control Society Standards Committee has been active in identifying and promoting those standards which apply to NC and CAM. While the committee does not write or set standards, they have recently published a directory of standards for NC machine tools and for the use of computers in manufacturing.

Aerospace Industrial Association

The AIA sets standards relevant to aerospace manufacturing. These standards are issued as National Aerospace Standards (NAS) and include machine tool specifications, systems configuration specifications, and standards adopted from EIA and ANSI.

Department of Defense

DoD writes the bulk of standards that influence Federal procurement of NC and CAM equipment. These military standards and military specifications primarily cover machine tool specifications. DoD issued military standards are available from the Navy Publications and Forms Center.

National Bureau of Standards

NBS administers the Federal Information Processing Standards (FIPS) Program which develops standards for Federal procurement and use of computer systems. Whenever possible these FIPS standards adopt existing ANSI voluntary industry-user standards. However, the FIPS Program may develop in house standards or cite those developed by other industry standards groups to satisfy the requirements of the Federal ADP Community. The FIPS program is organized around interagency Task Groups each of which addresses a single standardization area. Currently there are 16 Task Groups. Membership is open to all agencies of the US government. Draft standards developed in the Task Groups are submitted to NBS for coordination with all Federal agencies and with the public and industry. Final approval is by the Secretary of Commerce on behalf of the President. FIPS standards are mandated in all Federal procurements and can only be waived when justified by the head of an agency. Copies of standards are available from the National Technical Information Service of the Department of Commerce.

Programmatic Objectives

In developing a proper role for the Air Force in this area, it must be recognized that a standard is never an end in itself, but rather a means to an end. Therefore, a careful mapping must be made of program objectives and the manner in which these objectives become more attainable through the creation and adoption of certain standards. The National Bureau of Standards endorses the Air Force view that there are four programmatic objectives that are essential to the success of the ICAM program.

Integratable Modules	Distributed Data Processing
Portability of Software	Exchangeable Manufacturing Data

These programmatic objectives, diagrammed in Figure 1, will strongly influence the Air Force role in ICAM standardization.

Each objective will now be examined in detail with specific recommendations made regarding required standards, guidelines, policy and new developments.

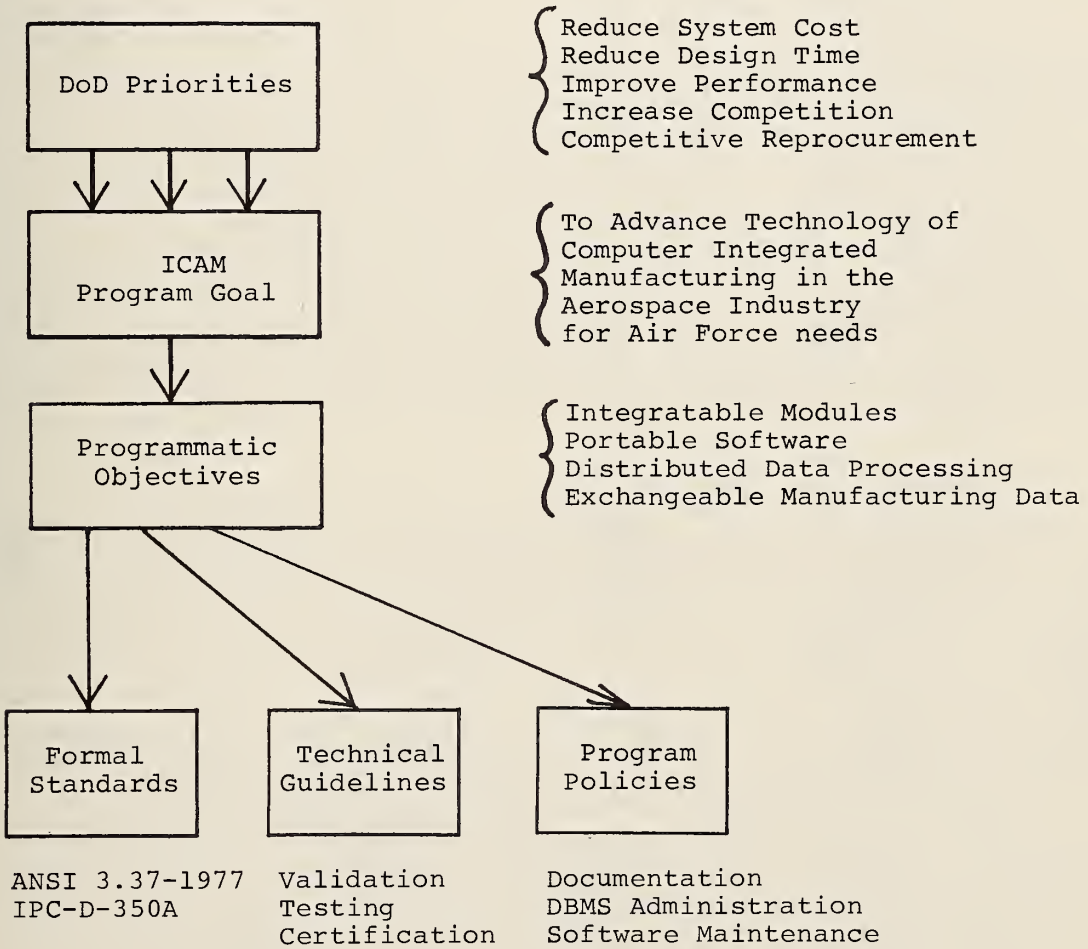


Figure 1 - ICAM Programmatic Objectives

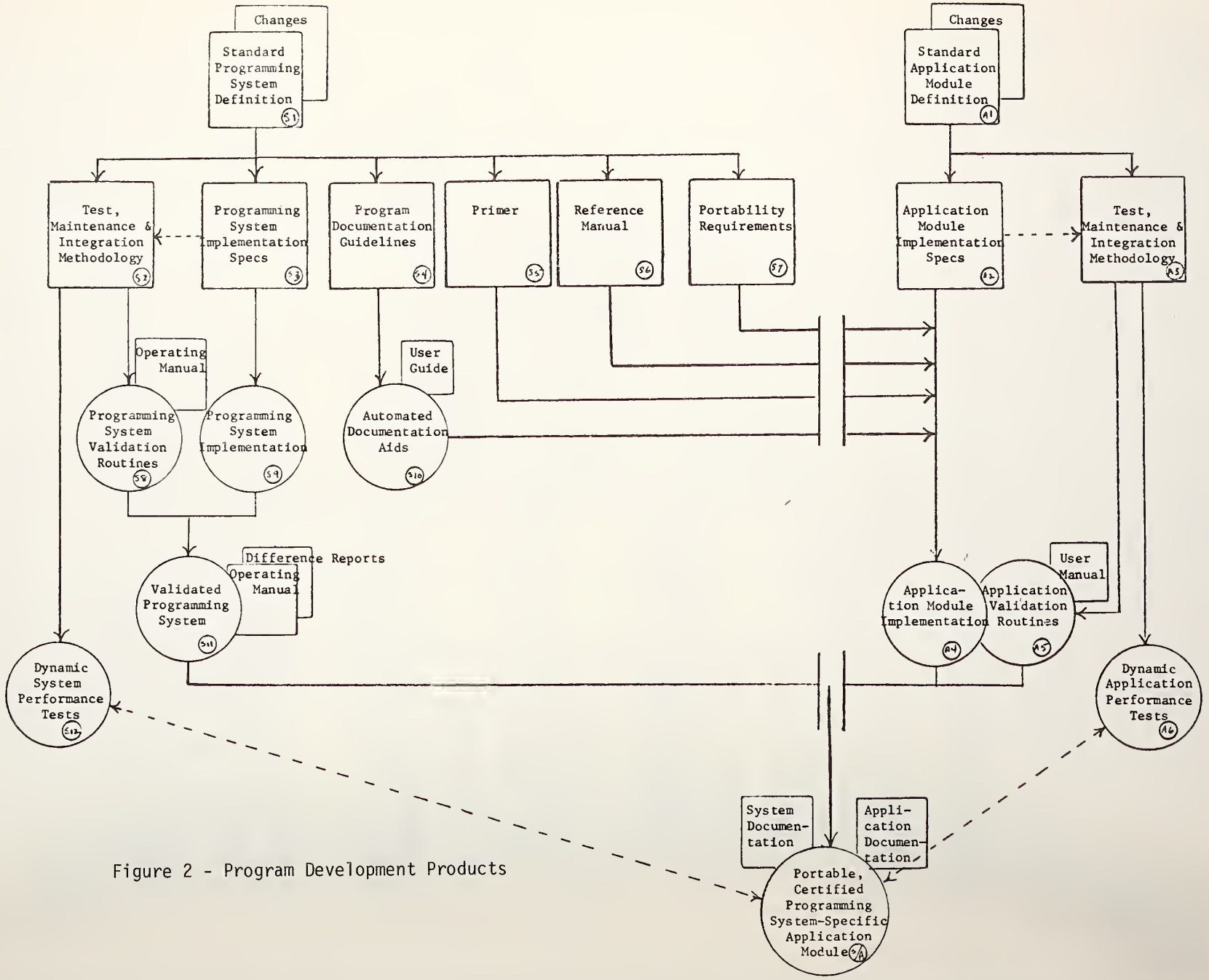


Figure 2 - Program Development Products

PORTABILITY OF SOFTWARE

The ability to transport computer programs among different computer systems with a minimum of software engineering is an essential component of the ICAM Program; essential for achieving the Air Force objectives to stimulate the widespread use of integrated CAM systems and essential for meeting the Department of Defense objectives to provide good technology transfer.

While many will agree that application program portability is desirable, most will disagree on how to achieve it. Obviously the specifying of a few ANSI standards will not produce the desired result. Many more elaborate schemes have been tried -- most having only limited success. Some have sought a common subset of a programming language which will execute on a variety of different configurations. One shortcoming of this approach is that it results in such a limited language set that coding becomes very inefficient. True portability of applications programs without sacrificing language power demands attention to minute detail in the definition, specification, documentation and testing of both system and application software.

Software Development Philosophy

Such an approach was used very successfully with the MUMPS programming system. It evolved over a three year period from several proprietary language dialects into a widely used, public domain programming system capable of easily transportable applications programs among 23 configurations of 15 different computer manufacturers. Experience gained by NBS personnel from the MUMPS success story has been formalized and tailored to address Air Force needs for an ICAM software development philosophy to achieve application program portability. Figure 2 depicts the major segments of this philosophy with the double vertical lines indicating the separation of system functions on the left from the application program functions on the right. The model's notation uses squares to indicate a document or set of papers and circles to indicate a computer program. Vertical lines indicate dependence of a lower entry upon a higher one. Horizontal lines indicate influence or interaction.

Figure 2 is intended to present a model of a software development philosophy which summarizes the processes and products that should exist or be brought into existence for the orderly development, testing, and management of applications programs such that they will be transportable across different computer lines. As this model represents only a single programming system, there will be one such model for each language used, i.e., FORTRAN, COBOL, PL/I, etc.

The system software which results from the definition, specification, implementation and testing on the left can be viewed as an "aid" to the development, testing, and operation of the applications modules shown on the right. Since applications programs must be expressed in a language that is acceptable to the computer, the construction of the system software which interprets that language becomes of critical importance. It defines the media of communication between man and machine, and much of what will be easy or difficult for the man in this relationship will be a direct result of the quality of this system software, its documentation and its techniques for testing and debugging of programs. Each of the functional elements of this software development philosophy will now be examined.

Standard Programming System Definition (S1)

This is the principal system element in the model. It is a meticulous and comprehensive functional definition of the programming language. The term "standard programming system definition" should evoke the image of a public domain document, one that is the result of a consensus arrived at by a representative group of interested users and suppliers, and one that has been used as the basis for successful implementations, hopefully on a variety of computers. Of the hundreds of programming system definitions in everyday use, only a few of them can qualify as standards. To date, only five programming language system definitions have been submitted for consideration as National or Federal Standards: FORTRAN, COBOL, PL/I, MUMPS, and BASIC.

Many of the newer system tools are highly proprietary, and no standards action is currently being taken on them. What this means is that the Air Force must either sponsor a variety of standardization activities that it feels are needed or resign itself to the use of proprietary products and the attendant miseries of sole source procurements, hardware dependencies, and the like. Clearly the route is easier if standardized products are chosen for Air Force use. However, ICAM may not wish to prohibit the use of non standard languages where their capabilities are superior. In these cases an effort should be initiated to formalize the Programming System Definition through a concensus opinion of users and suppliers so that compilers may be implemented on other computers to effect portability.

By way of clarification, the suggestion here is not that end products like compilers, data base management systems, and operating systems, should be in the public domain, but rather that standard definitions, upon which such products can be based, should be in the public domain. This approach is sound from a business standpoint, in that the definitional activity becomes a resource-sharing operation, thereby attracting to the task of orderly system definition, growth, and maintenance many of the most talented and interested individuals from the user and supplier communities. Also the implementation activity is strictly a free enterprise operation in which suppliers are rewarded for their ingenuity and innovativeness, and thereby are encouraged to separately build, from these standard definitions, ever more capable and efficient standard products.

System Validation

Functions in this area represent elements of the first phase of system testing, that of system validation, wherein an assessment is made of the degree to which a programming system implementation such as a compiler (S9) conforms to the standard definition (S1) which it purports to follow. Shown in Figure 3 the actual components of such a test are a supplier-developed programming system implementation (S9) and system validation routines (S8). The routines consist primarily of data and are specific to the standard programming system in question (S1) but are implemented in a general fashion that enables them to interact with any product (S9) that is based on the standard definition (i.e., any product that follows the development plan depicted by the vertical path S1, S3, S9). The validation routines are then "run" together with the programming system implementation, (S9), to provide a pass/fail analysis. Upon completion of the test, the programming system implementation (S9) is classified as either valid (S11) or invalid (in which case it remains an "S9" product). As can be seen in the figure, establishment of programming system validity is crucial to the development of portable application programs, as it provides the single system "link" with application programs (A4), thereby making such development possible.

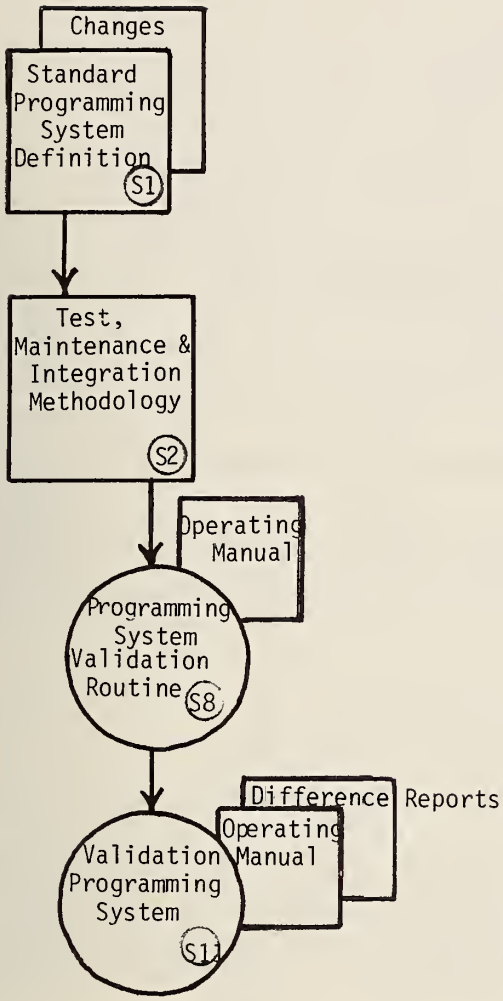


Figure 3 - System Validation

A few words might be said here about the "difference reports" that are shown accompanying the validated programming system (S11). In order to be a constructive aid to standardization, one that will be used by both implementors and users, the validation process should support rather than interfere with the other important processes of system definition, specification, implementation, documentation, and maintenance. A recommended way of doing this is to allow, and in some cases, even encourage differences in all of these areas so long as application program portability objectives are not compromised. The following is an example of how this can be done. Assume that a supplier "extends" his specification (S3) beyond the standard (S1), implements new, nonstandard features, and incorporates them in his proprietary product (S9, S11). The supplier will then want to document these extensions in his instructional literature. However, this is not sufficient for the user who wants to preserve program portability, as the use of these extensions will render his application programs nonportable. Such a user will require notice of the presence of the extensions so that he can invoke a management prerogative to either permit or prohibit their use. Since the notice must precede the choice, the notice should always appear; its ideal place is in a document similar to the portability requirements (S7), whose purpose is to give an "early warning" of impending problems. If an extended system implementation (S9, S11) is chosen for operational use, then the user should write an installation-specific version of the portability requirements document which clearly delineates policy regarding the use of nonstandard constructs.

System Portability Requirements

This document originates in the standard programming system definition (S1) and may even be an appendix to that document. Its purpose is to highlight, for the benefit of implementors and application programmers, aspects of the system that must be accorded special attention if program transferability (i.e., portability of application source code between various system implementations) is to be achieved. It makes an a priori identification of potential problem areas that one could be expected to encounter during the specification phases of standard systems (S3) and standard applications (A2), and issues appropriate cautions regarding the definition and implementation of system extensions and, to application programmers, corresponding cautions regarding the use of such extensions.

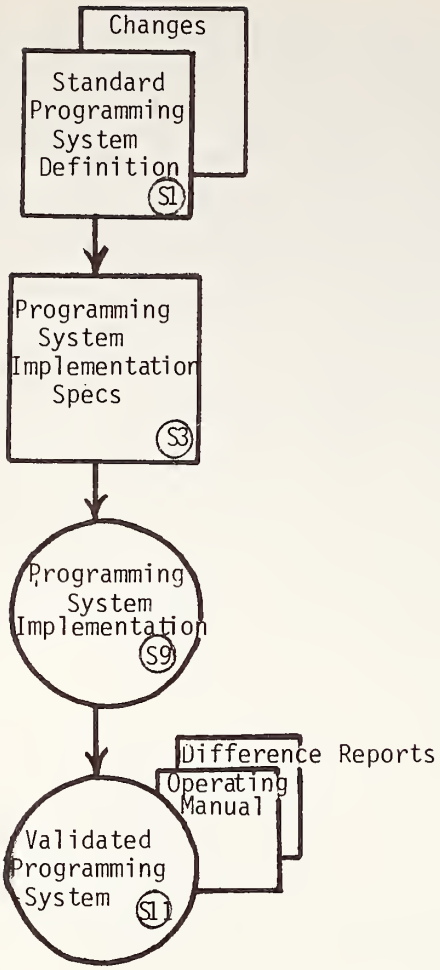


Figure 4 - System Implementation

Programming System Implementation

Products required for the implementation of a validated programming system are shown in Figure 4. Programming system implementation specifications, (S3), are owned, developed, and maintained by the various system suppliers and consist of the standard programming system definition (S1), with or without extensions, and are augmented by configuration-specific implementation instructions. These specifications serve as the basis for an "interim" product, a programming system implementation (S9) and the validation version of that implementation (S11).

System Primer and Reference Manual

A primer (S5) and a reference manual (S6), represent two different levels of training aids, the former for the novice technicial, the latter for the more experienced one. They serve as a bridge between system implementation (S3) and application implementation (A2). Both can be developed centrally as part of the definition phase (S1), thereby enabling potential implementors to share costs, without risk of violating antitrust laws. Furthermore, implementors who find it necessary or desirable to extend their system implementations beyond that of the standard may extend these documents accordingly.

Documentation Guidelines and Aids

Program documentation guidelines (S4) and automated documentation aids (S10), represent two system products that are indispensable to the program preparation process. The guidelines (S4) are specific to the programming system in question, taking into consideration certain eccentricities of that system and the resulting need for lucid representation of logic, data, etc. The automated aids are computer programs that render the system "self-documenting." These are an optional extension to the system guidelines. They encourage and assist programs in the production of detailed, uniform application program documentation.

SYSTEM AND APPLICATION PROGRAM TESTING

Figure 5 shows how criteria used for testing an applications program closes the loop in this philosophy of software development. The testing of the applications modules produced insures that they do in fact conform to the criteria established in the test methodology (S2 and A3). Application Validation tests (A5) certify that an implemented module satisfies the problem to be solved, that it adequately detects and handles all erronious input data and that it is coded in acceptable programming language for protability requirements. Dynamic tests (S12 and A6) are those which probe the behavior of system software or application programs while they are in an operating state. The tests differ substantially from the static tests performed under system or application validation (S8 and A5). Dynamic measurable behavior generally includes timing and memory space utilization.

In brief, dynamic system tests are ones that probe the behavior of system components (e.g., compilers, operating systems, data base management systems, hardware devices, etc.) either individually or in combination; they are "driven" by hooking them on to "live" applications (S/A, resulting from the interaction of S11 and A4) or by hooking them on to simulated applications (S/A, resulting from the interaction of S11 and A6, where A6 is a "rigged" substitute for a live application (A4)). Dynamic application tests, on the other hand, probe the behavior of application programs by isolating and analyzing certain aspects of the "A portion" of the S/A entry.

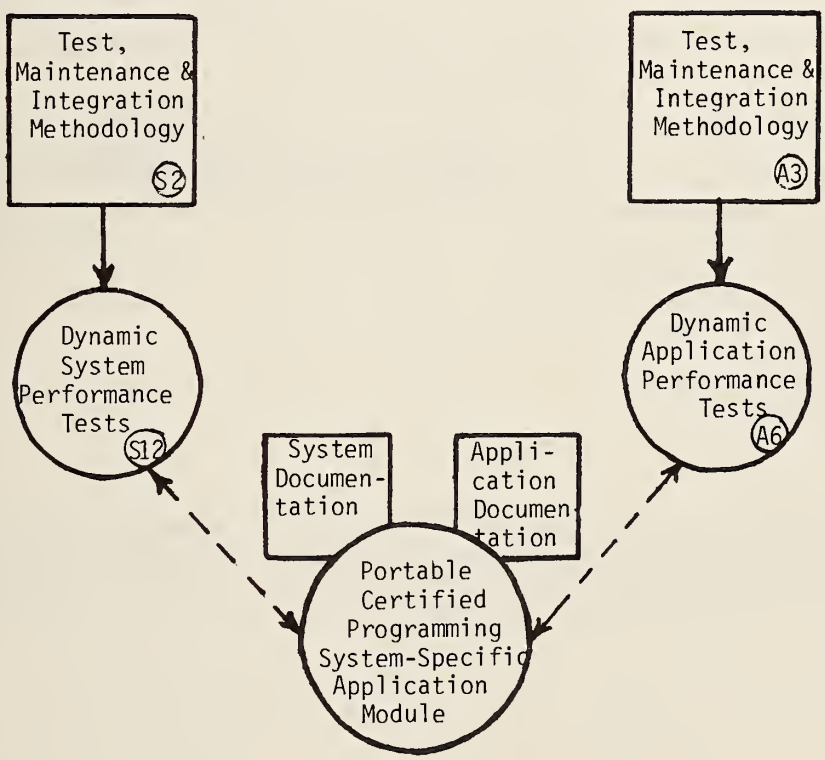


Figure 5 - Program Testing

These dynamic tests provide information that is indispensable to the processes of understanding, maintaining, and improving the performance of both system and application components of complex automated systems.

Summary

Portability of applications programs among different computer systems or configurations is an essential component of the ICAM Program. This portability cannot be achieved without giving detailed attention to both the programming system and the applications program development process. A model of a software development methodology has been presented which addresses the various elements of standards, guidelines, and discipline that are needed by ICAM to permit the development of:

1. portable application program modules,

2. maintenance and improvement methodologies for both system and application programs, and

3. "fall-back" procedures for the eventual replacement of individual system or application modules.

For this methodology to work all of the elements starting with the standard programming system definition must be available to an ICAM contractor. Table 2 shows for the five existing standard system definitions the availability of the system elements. Where none exist, resources will be required for the needed development.

Table 2 - Available Software Development Definitions and Supporting Products

	FORTRAN	COBOL	MUMPS	PL/I	BASIC
Programming System Definition	X	X	X	X	X
System Implementation Specs	X	X	X	X	X
System Validation Routines	X	X	X		X
Applications Validation Routines					
Dynamic Performance Tests	X	X	X		
Difference Reports		X			
Primer and Reference Manual	X	X	X	X	X
Portability Requirements		X	X		X

Air Force insistence on the availability of standard system software and willingness to sponsor the type of effort needed will vastly facilitate solutions to the long term problem of producing portable software for computer integrated manufacturing systems.

RECOMMENDATIONS

(Policy)

1. ICAM should formulate a definitive written policy on how it intends to achieve portability of applications software among different manufacturers and configurations of computer systems. The document should conform closely to the philosophy presented above.

2. Applications programs should be developed only with high level programming languages except in those rare instances where acceptable performance can only be achieved through assembly language. These cases must be carefully controlled and documented.

3. The Air Force should encourage the use of standardized programming languages. NBS believes their effective use to be the key to software portability.

4. While FORTRAN and COBOL will suffice for near term applications, conversion to the use of a more modern language should be encouraged whenever the various standardized supporting products are available to meet ICAM requirements for portability.

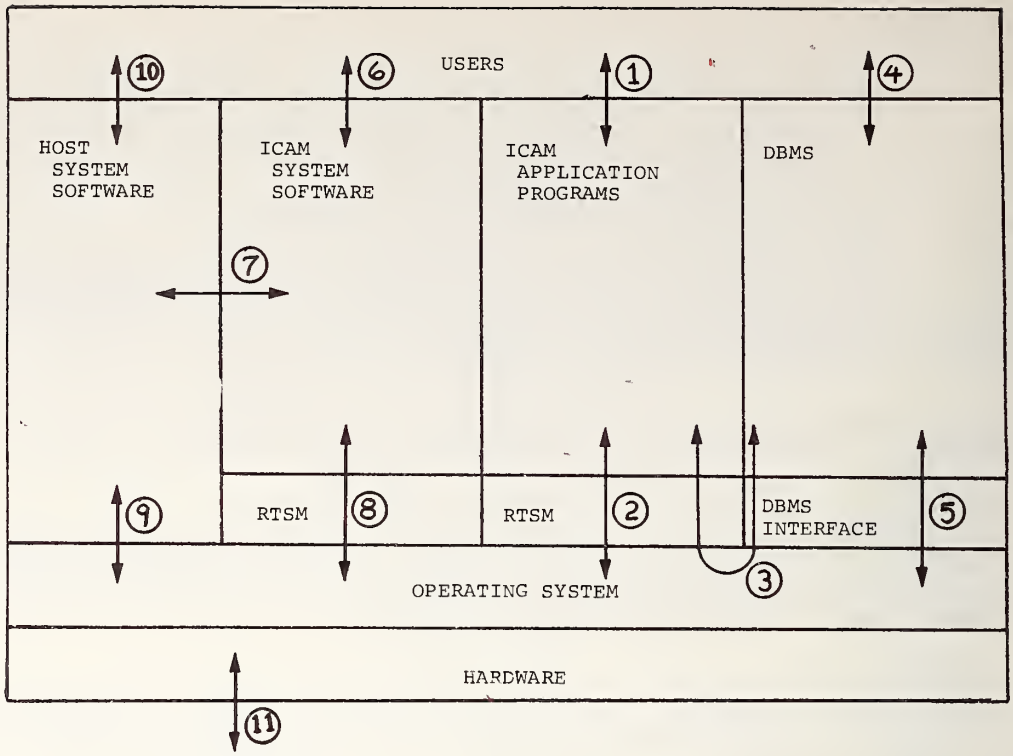
(Guidelines)

1. Guidelines for recommended programming practices are needed to assure that applications code generated will be legible and maintainable.

2. Program documentation guidelines are required to be consistent with the software development philosophy and to assist in the transfer of software technology among different Air Force contractors.

(Standards)

1. The Air Force should support the expeditious establishment of a Federal FORTRAN standard based upon a revised national standard. If ANSI does not approve the revised FORTRAN in 1977, CAM officials should support NBS effort to establish a Federal standard from the best available ANSI proposal.



MAJOR INTERFACES

APPLICATION PROGRAM INTERFACES

1. User Language
2. Run Time Support Monitor: one for each programming language (interprets system calls)
3. Data Manipulation Languages (DML) and sub-schema Data Definition Language (DDL)

DBMS INTERFACES

4. Self contained language queries of data base
5. DBMS/Host interface: designed for each host system ICAM system interfaces (supplied by vendor)
6. User/System commands (machine independent)
7. ICAM System/Host System interface: designed for each host system
8. ICAM System/Host interface: run time support monitor for higher level language used to implement ICAM system

HOST SYSTEM INTERFACES

9. Host System/Host interface implemented by vendor
10. Operating System Command Language (to be avoided if possible)

HARDWARE AND COMMUNICATIONS INTERFACES

11. Interfaces to Networks and Local peripherals

MAJOR ICAM FUNCTIONS

ICAM APPLICATIONS PROGRAMS

Tool Selection
Process Planning
Inventory Control
Etc.

DBMS

Management of all data (distributed)
Query response/report generation

ICAM SYSTEM SOFTWARE

Text Editing
Debugging and Test Software
Math Libraries
Etc.

HOST SYSTEM SOFTWARE

Assembler
Compilers
Linkers
File Management Commands
Etc.

FIGURE 6 - SYSTEMS INTERFACES

INTEGRATABLE MODULES

The development of a true general Integrated Computer Aided Manufacturing System using software developed by several different contractors requires a partitioning of the system into logically separable modules with well defined interfaces.

Figure 6 schematically shows the location of the major interfaces of a large integrated system such as ICAM. A few of these interfaces will be explicitly considered:

- Interfaces between ICAM applications programs and the host system
- Interfaces between ICAM applications programs through the DBMS
- System software interfaces

Applications Program Interfaces

The first of these interfaces is the key to software portability: with adequately standardized and validated programming languages, the applications programs will be essentially independent of the host system. System calls are intepreted by the run time support monitor (RTSM) which is supplied by the host system vendor for each supported programming language. If the language is standardized, the RTSM is also thus allowing portability. The user interface (1) is a real time interactive user language, if any, allowing the user to interact with the system in real time. The location shown for interface (1) is the logical interface; the physical interface is, of course, through a terminal with its hardware interface (11) and the operating system. Note that there are no interfaces between applications programs and the system software.

The language standards relevant to interface (2) are discussed in Portability of Software.

Data Base Interfaces

From the point of view of integration of application modules, the most critical element is the data base. A single (but, perhaps, distributed) data base, to be accessed by all programs through a well defined interface, is the key to the integration of a manufacturing operation into a complete computer-aided system. This data base becomes the source of all information for, and hence the interface between, all applications programs. This critical module - the data base - must be carefully structured and managed. This is the job of the data base management system (DBMS).

Over 200 different DBMS packages are presently available although there exist significant variations in the capabilities and features provided by these systems, certain common functions and interfaces are noted. Applications programs communicate with the data base through interface (3) often with a form of data manipulation language. Interface (5), of course, is essential for interacting with the operation system to store and retrieve the data on the various physical storage devices. This interface sometimes uses a device media control language which is specific to each host computer. Often query packages are provided to users to allow for non-procedural type requests on the data in a self contained query language through interface (4).

No standards exist as yet in the area of data base management systems. The CODASYL specifications have gone the farthest to define and structure an approach which has modular architecture and well defined interfaces. But while there are several CODASYL-type systems available, they are by no means identical and the specifications themselves are still in a state of change. Consequently, there can be no one DBMS identified as being "best"

for ICAM use. As a single DBMS will be critical to achieving integration of ICAM software modules, a competitive procurement of a commercial data base software package is recommended to support all near term projects. Emphasis should be placed on obtaining modular architecture, well defined interfaces, portability of applications programs, integratability of ICAM modules, and future adaptability to a computer network system with distributed data bases. The evaluation for selection should include a benchmark demonstration of performance on a typical CAM application.

The key to successful operation of a large DBMS is the Data Base Administrator who is recommended as a key person in the Air Force ICAM Standards Office. The Data Base Administrator:

a. creates and maintains the data definitions for existing application system, and establishes the data definitions for new systems;

b. maintains a library of the data available on the data base;

c. provides data base documentation to the analysts and programmers, such as cross reference listings between data and programs;

d. controls the schema and subschema, thereby controlling access to the data base;

e. is responsible for improving the efficiency of data base operations, including performance monitoring activities, keeping statistics on the use of different data, and monitoring the use of physical file spaces; and

f. in general, maintains integrity and security of the data, including definition of backup requirements and recovery procedures.

System Interfaces

Certain system software will, by necessity, be specific to the host computer system and hence will be supplied by the host vendor. In particular, compilers, linkers, accounting programs, and the system executive will all be host specific.

The interface between a user and the host system software is shown as interface (10). Communication across this interface must be expressed in the language of the host system software and hence will vary from system to system. Extensive use of host specific software will endanger system portability and hence use of this interface should be discouraged. A better way of talking with the host system software while preserving portability is through interfaces (6) and (7) and the use of specially developed ICAM system software.

ICAM systems software should contain all of the software tools needed for the development, maintenance and operation of the applications modules on the host computer with the exception of the usual vendor supplied system software mentioned above. This ICAM systems software, written in a high level language so as to relatively transportable itself, will contain such programs as:

Text Editor - For entering, correcting and modifying applications and general text such as program specifications and design documentation.

Program Librarian - For storing all program texts, associated job control statements, common data definitions, test data, and for maintaining a chronological record of modifications between distinct versions. Includes appropriate access controls for members of the project group. This may require a DML interface to the DBMS.

Test and Debug Programs - For analyzing program behavior during preparation and execution on test data input, and deriving execution statistics and traces to help correlate program output with the results of individual high-level language statements. These programs will be specific to a particular high-level language such as FORTRAN.

User Interface - Programs which present a standard interface to the user and translate system commands into host specific commands at interface (7). This is essentially a transparent "shell" that provides a machine independent system executive.

Documentation Aids - Programs to assist in documenting applications programs as they are developed.

Project Manager - For recording chronologically the activity of the individual project members on defined application program modules and deliverables of the project.

Each item of ICAM systems software being written in the high level language interacts with the host operating system through interface (8) in exactly the same manner as an application program does through interface (2).

Peripheral Interfaces

Near the bottom of Figure 6 the interaction of the operating system with the various hardware peripherals is shown by interface (11). Local peripheral equipment interface standards should follow the ANSI standards on the channel level and on the minicomputer device level when they are adopted. Communications interface standards are discussed below under distributed data processing.

Recommendations on Systems Integration

(Policy)

1. The DBMS is the key to integration of applications modules. Functional specifications should be prepared for the competitive procurement of a commercially available data base software package to support all near-term ICAM projects. The specification should require the package to be available on all hardware systems that would be considered for CAM applications in the first few years of the program.

2. A Data Base Administrator should be appointed to the staff of the Air Force ICAM Standards Office. This person should remain fully informed of the state-of-the-art in data base software, and evaluate available packages for future ICAM application and standardization. This person should participate in on-going Federal and national standards efforts, including the NBS FIPS Task Group 24.

3. The Air Force should provide staff to support the CODASYL Task Group in future developments.

4. Dependence on host system software should be minimized, with interfaces provided through machine independent ICAM system software written in a high level language.

(Guidelines)

1. The following user guidelines will be required for effective use of the DBMS. If they are not supplied by the vendor, they should be created by the ICAM Data Base Administrator.

   Data Dictionary
   Device Media Control Language Guide
   Data Manipulation Language Guide
   Guide to the user of query package/report generator

2. A Guide to ICAM systems software should be developed and distributed to all ICAM contractors and users.

(Standards)

1. There are no standards on DBMS or operation systems.

2. Local peripheral equipment interface standards should follow the ANSI standards on the channel level interface and the minicomputer device level interface when they are adopted.

DISTRIBUTED DATA PROCESSING

A stated objective of the Air Force ICAM program is compatability of ICAM software with Fourth Generation distributed system concepts. The trend toward linking central processing units together with either local or commercial communications networks is clear, and the implications for improved efficiency and lower costs is also clear.

The scenario for CAM applications for the 1980's is, then, a series of mini and microcomputers, each running dedicated applications programs, networked together in a total CAM system. One computer may be running the warehouse and providing inventory control, another running CAD programs and supporting interactive graphics, another supporting process planning programs, while an entire hierarchy of computers operates the machine tools and materials handling systems in actual manufacturing tasks. Each computer will keep its own local data base, or possibly have a local "back end" processor running a DBMS, and will be able to access relevant data throughout the distributed system through the communications system by using appropriate access protocols.

The potential power of such an array of processors all working in parallel and yet all acting coherently and with distributed data access is tremendous. Even more importantly, the cost reductions of such a system may be remarkable as mini and microcomputers and mass storage devices continue to drop in cost. Furthermore, the architecture of this systems design ideally satisfies the ICAM objective for having individual modules developed and implemented separately to perform useful work in a stand alone mode while allowing for eventual systems integration.

Subcontractors may be able to make use of CAM capability by directly receiving manufacturing data packages through a network in a form that will directly run their machine tools and inspection machines. Procurement costs will drop significantly if procurement practices are changed to reflect the opportunities of this new technology.

Standards for Distributed Systems

The layers of protocol for one application program in one computer to talk to another application program in another computer are diagrammed in Figure 7. The lower three levels are the subject of formal standards, and the combination of EAI RS232, RS XYZ, CCITT X.21, ANSI ADCCP, and CCITT X.25 offer a comprehensive definition of protocols to access commercial packet communications and switched networks. These standards provide a sound basis for a distributed processing system using commercial communications systems.

The higher level protocols are as yet unstandardized. Host to host protocols are defined by each vendor, for example, in IBM's SNA or DEC's DECNET, but these are specific to each vendor. Potentially a project standard could be implemented for each allowed host system. Process level protocols should be the subject of project standards.

The National Bureau of Standards is initiating (1977) a new project in support of the Air Force Rome Air Development Center to define these higher level protocols. This work will provide a basis for developing ICAM project standards.

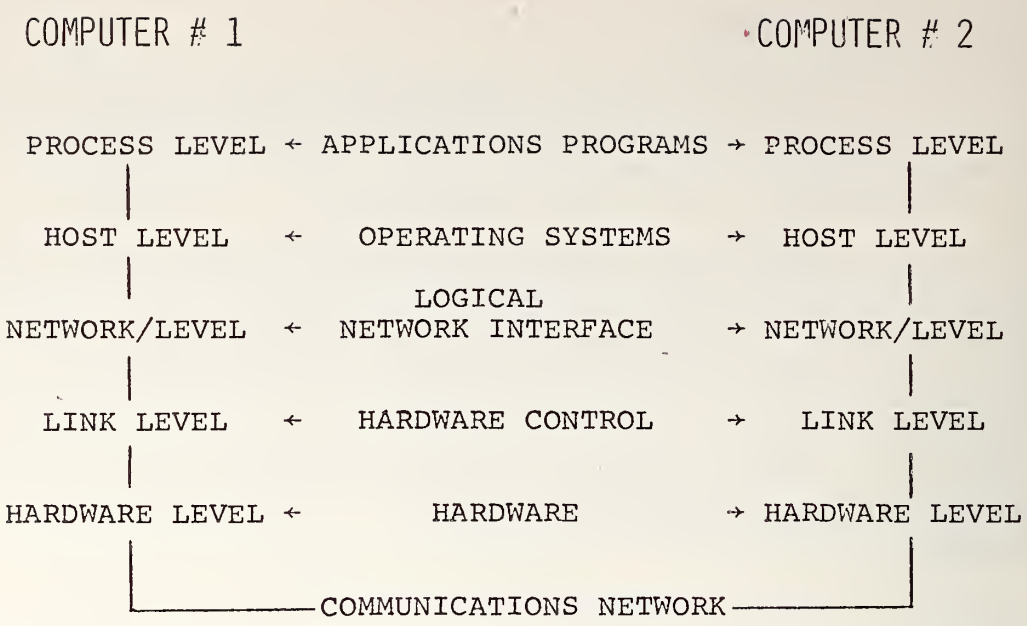

COMPUTER # 1
COMPUTER # 2
PROCESS LEVEL
APPLICATIONS PROGRAMS
PROCESS LEVEL
HOST LEVEL
OPERATING SYSTEMS
HOST LEVEL
LOGICAL
NETWORK/LEVEL
NETWORK INTERFACE
NETWORK/LEVEL
LINK LEVEL
HARDWARE CONTROL
LINK LEVEL
HARDWARE LEVEL
HARDWARE
HARDWARE LEVEL
COMMUNICATIONS NETWORK

Figure 7

RECOMMENDATIONS

(Policy)

1. The ICAM system should be designed and implemented in such a way that it could be run on a highly distributed fourth generation computer system in the 1980's. The development and use of communications software and protocols, programming language subsets for minicomputers and microcomputers, and a network model DBMS follow from this approach.

2. The ICAM Standards Office should task NBS to specify host level and process level protocols for use as ICAM project standards, working from the Rome Air Development Center project as a base.

3. The ICAM Program Manager, with advice from the Industry Review Panel should set security requirements for distributed systems.

4. The ICAM Industry Review Panel should be tasked with considering the impact of this technology on procurement practices in contracts between the Government and a prime contractor and between a prime and various subcontractors.

(Guidelines)

1. A guide to the use of DTE/DCE (Data Terminal Equipment/Data Communication Equipment) interface standards, link level protocol standards and network level protocol standards should be developed and distributed to all contractors and users of ICAM software.

2. A guide to security in a distributed system should be written and distributed to all contractors and systems users.

3. A guide to implementing host level and process level protocols should be developed by NBS and distributed to all contractors and systems users.

(Standards)

1. The following standards are recommended for ICAM use:

   DTE/DCE Interface: RS 232C, RS XYZ, CCITT Recommendation X.21
   Link level protocol: ANSI ADCCP
   Network level protocol: CCITT Recommendation X.25
   Communication Codes: ANSI X3.4 (ASCII)

2. The Air Force should task NBS with maintaining cognizance of developments in the recommended standards, representing Air Force interests in standard committee meetings and providing regular reports to the Air Force for distribution to ICAM contractors and users.

EXCHANGEABLE MANUFACTURING DATA

Too often we view the output of the manufacturing process as simply the delivery of finished goods. However, in government procurement of weapons systems the output of the manufacturing process is seen to also encompass the data which defines and controls the manufacturing of the end products. The difference is significant and reflects the concern of the Department of Defense (DoD) with the life cycle cost of the system. Data must be available which completely defines the manufacturing process of each component part of the end product. More importantly, this data must be presented in such a form as to be meaningful and useful to the different manufacturing systems that may be involved in the production of parts for the system over its entire life span. The importance of this exchange of data is illustrated by the following evolutionary phases of a typical system:

Developmental Phase	Part manufacturing data flows to and from Primes and Subs
System Delivery Phase	Manufacturing data on each piece part is supplied to government
Competitive Remanufacture Phase	Government furnishes complete manufacturing data to bidders. Primes may pass data to subs.
Maintenance and Repair Phase	Manufacturing data flows among various government repair facilities and logistics

Engineering Drawings

Presently the bulk of this manufacturing data is exchanged through the use of engineering drawings. Much time and effort at specifying and standardizing this form of data presentation have made it a useful and universal method. Although time consuming and expensive to create, engineering drawings fulfill their purpose well in being easily exchangeable and readily understandable.

However useful engineering drawings may be, they have been a creation of a manual system. Drawings are developed by hand and are meant to be interpreted by hand. Even though computers are assisting in the preparation of some drafting work, the method of data presentation is still pointed toward the exchange of manufacturing data in a conventional environment.

Digital Data Package

The increasing use of computer technology in manufacturing is creating some interesting problems in the exchange of discrete part data. For one thing part data is taking on more forms than just the traditional dimensioned engineering drawing. Digital data descriptions are now being used by numerically controlled machine tools to manufacture parts directly. These digital part descriptions are exchanged in various codes and formats on paper tape, punched cards or disk files. The most prominent standard in this area which pertains to manufacturing data is for the APT programming language; however, a competing language, COMPACT II, is in the early stages of standardization. COMPACT appears to be more efficient than APT for simple parts but is not as capable as APT when used for very complex parts. It should be mentioned that neither language is sufficient as a manufacturing data package. Nor is the APT language in a sufficiently standardized form to serve Air Force needs. While the new standard on APT to be released in 1977 will do much to alleviate this problem, guidelines are needed on the interpretation and use of the APT language to make it a more flexible

and versatile tool for exchanging NC manufacturing data among functionally equivalent machines.

Promising work is being done for the manufacturing interface in the CAM-I Geometric Modeling Project, by the ANSI subcommittee on Computer Aided Preparation of Product Definition Data, and by NASA's IPAD Program. The Air Force is advised to continue to maintain close liasion with these three projects as a useful and meaningful manufacturing data specification is surely to evolve from the healthful interaction of these efforts.

Whether one talks of integrated CAM systems or simple numerical control machines, much less reliance is placed upon formal dimensioned drawings. Where drawings are used at all, their format is often simplified to convey general ideas rather than specific detail. Exact dimensions are transmitted digitally.

The trend is clear. As the Air Force and industry progress toward complete computer integrated manufacturing, the dimensioned engineering drawing will cease to be an essential component of manufacturing data. It will be supplanted by a digital part data description that will completely describe the manufacturing process to be used. It will also define digitally any computer generated drawings that are needed for informational purposes. The evolution of this digital part data package is inevitable. While it will eventually affect all components of the Department of Defense, its impact will be felt first in the ICAM Program. Techniques must be developed to allow digital data to be exchanged in as versatile, efficient and flexible a manner as engineering drawings are done now. Technical barriers must be overcome to enable this data to be exchanged among various combinations of government agencies, prime contractors and subcontractors. Means must also be found for the data packages to be exchanged between computer based manufacturing systems and the conventional types. Unless these capabilities are developed, the Department of Defense will incur many unnecessary costs in the evolution of the computer integrated manufacturing systems envisioned by the ICAM Program. The groundwork must be developed now for firm policy, effective guidelines and explicit data standards that will collectively enable the needed exchange of digital manufacturing data.

RECOMMENDATIONS

(Policy)

1. Task the ICAM Industry Advisory Group to develop recommendations for policy regarding the specification, purchase, data rights, and exchange of digital data for discrete part manufacturing.

2. Through the ICAM Standards Office maintain close liasion with the NASA IPAD Program, the ANSI Y14.26.1 subcommittee on Computer Aided Preparation of Product Definition Data, and the CAM-I Geometric Modeling Project to avoid potential problems in compatability.

(Guidelines)

1. Support the development of guidelines on the interpretation, processing and use of the APT language for numerical control manufacturing, such that an APT part program can be quickly and easily exchanged among different machine tools, different shops and different contractors. Inconsistencies in the present use of the language inhibit this ability.

(Standards)

1. Where the Air Force specifies a single part programming system for numerical controlled machine tools, it should conform to ANSI 3.37-1977 APT. If two part programming languages can be acceptable COMPACT II is recommended for efficiency at programming less complex parts provided the CLDATA file can be produced to be compatible with APT output.

2. Through the ICAM Standards Office task the National Bureau of Standards to maintain close liasion with the ANSI subcommittee X3J7 on APT Language Standards.

3. Insist on the use of the IPC-D-350A Standard on "End Product Description in Numeric Form for Printed Wiring Products" for future wedges relating to electronic systems.

SUMMARY OF TECHNICAL RECOMMENDATIONS

EVALUATION AND RECOMMENDATIONS ON CAM STANDARDS

EVALUATION AND RECOMMENDATION ON COMPUTER STANDARDS

- Communications
- Codes
- Software
- Documentation
- Media

SUMMARY MATRIX

This report recommends optimal standards for the Air Force ICAM program. Many formal standards that now exist are recommended as relevant to the ICAM program and these are expected to remain so in the future. Furthermore, trends and developments in standardization process are enumerated with key areas identified for monitoring or development. Finally, several areas are cited where formal standards do not exist but where project standards will be necessary.

This report and the recommendations of this report should be considered as a first step in an interactive effort to define the nature, the detailed structure, and the details of implementation of ICAM projects.

Evaluation and Recommendations on CAM Standards

There are few CAM standards that can be evaluated, a result partly of the newness of the field and partly of the way in which CAM systems have been developed primarily by large user industries for their own internal (and hence proprietary) use. In the area of NC programming languages, standards have developed because of the multi-industry development effort and Air Force contractual requirements. The APT language standard is recommended as a minimum, with extensions to cover adequately the post processor area. If two languages can be allowed, the COMPACT II/ACTION/SPLIT family should also be used since it is more efficient on simple parts and can product compatible CL data as an option.

In the CAD/CAM interface area, the ANSI Y14.26.1 effort, NASA's IPAD, and the CAM-I Geometric Modeling Project are considered in relation to the digital representation of physical object shapes. The ANSI approach is recommended, and the Air Force is advised to monitor the other efforts to insure eventual compatibility with the ICAM system. In addition, the Institute for Printed Circuits standard on printed circuit boards is discussed as a tutorial example and recommended where appropriate.

Evaluation and Recommendations on Computer Standards

Communications

In order to construct distributed fourth generation computer systems expected to be in wide use in the 1980's, adequate communications interface standards are a necessity. A suprising amount has been dome on hardware standards and comminucations protocols. A comprehensive set of standards, some of which are only in the formation stages, is recommended on computer peripherals (ANSI proposed channel level and minicomputer device level interfaces), DTE/DCE interfaces (RS 232, RS XYZ, CCITT X.21), and bit oriented link level and packet network protocols (ANSI ADCCP and CCITT Recommendation X.25). Following these standards, a distributed computer system can be developed, using commercial communication services, that will remain relevant into the 1980's.

Codes

The lowest level of information storage and transmission is the character code level. Serious problems may arise in code conversion and in accessing or merging files with different coding schemes. These problems are discussed and the American Standard Code for Information Interchange (ASCII) code is recommended for data crossing any system interface.

Software: Languages, Data Base Management, and Operating Systems

The Air Force has stated that their objectives for the ICAM system include software portability, integration of software modules and, potentially, distributed data processing. These requirements lead to a consistent set of recommendations for programming languages, data base management, and operating systems.

Standardized programming languages offer the key to portable software. Using adequate language standards and requiring validation of compilers against those standards will be required for Air Force ICAM software to be portable. FORTRAN and COBOL will have to be supported to the near term because of the bulk of application programs written in those languages. Eventual conversion to the use of a more modern programming language should be anticipated. Representative of the "modern" languages is PL/I which is the only one that has been submitted for standardization. However, substantial effort remains before PL/I can be termed suitable for ICAM needs.

From the point of view of integration of applications modules, the most critical element is the data base management system (DBMS). The recommendation here is to prepare functional specifications for the competetive procurement of a commercially available data base software package to support all near term ICAM projects. Emphasis should be placed upon obtaining modular architecture, well defined interfaces, portability of applications programs, integration of ICAM modules, and future adaptability to a computer network system with distributed data bases.

In Operating Systems there are no standards. This is a major problem area from the point of view of software portability, but a standard operating system is not feasible for large scale computers because of the differences in architectures. However, a standard operating system for 16 bit or 32 bit byte oriented machines seems at least technically feasible. It may be necessary to implement project standards on file names and library names to avoid problems in portability due to differences in file management conventions.

Documentation, Validation and Testing, and Software Tools

These are some of the most important tools to insure system integration and software portability and maintainability. Detailed recommendations are not possible until the maintenance of ICAM software is better defined, but general requirements and various approaches are discussed and evaluated, and general guidelines are provided. It is recommended that the Air Force use validation and testing procedures, and that general software development tools be developed, used in ICAM development, and then made a part of the ICAM system.

Media

Magnetic Tape and discs and direct communication links are the primary recommended standards for transmitting ICAM data and software within a given installation and between installations. Where punched cards must be used, standards are available. Paper tape, even for NC, is not recommended; instead direct communications links (DNC) should be used.

Summary Matrix

There is no comprehensive set of present day standards that will solve all of the Air Force's needs. However, where today's standards are inadequate, major trends, developments and needs for project standards have been identified that should provide the Air Force ICAM program with sound initial guidance. A summary of recommendations is given in the matrix of Figure 1.

Standard Cases	Major Standards	ICAM Plan	Anticipated Further Efforts Required
CAM STANDARDS			
1. NC Languages	APT COMPACT II	Use both languages, require CLDATA as interface	Standardize postprocessors Monitor X3J5, X3J7 for future developments
2. CAD/CAM	ANSI Physical Object Description (Y14.26.1) NASA IPAD CAM-I IPC Printed Circuit Standard	Use ANSI Y14.26.1, IPC standard where appropriate	Monitor IPAD, CAM-I, resolve any incompatibilities that develop
COMPUTER STANDARDS			
1. Computer and Communications Interface Standards	EIA RS 232, RS XYZ (data terminal to analog communications) CCITT X.21 (data terminal to digital communication) Peripheral Interface Standards ANSI ADCCP Link Level Protocol ANSI X.25 Packet Network Protocol	Use RS XYZ or X.21 for systems in 1980s/use peripheral interface standards as adopted; Work with commercial networks for distributed systems	Monitor and evaluate further development of link level and network level protocols Develop host-to-host level protocols for distributed systems
2. Communication Codes	ANSI X3.4 (ASCII) IBM EBCDIC Encryption Algorithm	Use ASCII at all system interfaces and for collating; use encryption algorithm if security required	
3. Programming Languages	FORTRAN COBOL PL/I BASIC	Support FORTRAN and COBOL for near term	Investigate PL/I and standard subsets of PL/I. Investigate DoD-1. Validate compilers and run time support routines to insure portability.

Standard Cases	Major Standards	ICAM Plan	Anticipated Further Efforts Required
4. Operating Systems	No standards	Develop project standards on certain aspects of operating systems, particularly system calls and file names.	Consider development of standard operating system for 16/32 bit minicomputers and cross system support for micro-computers.
5. Data Base Management Systems	No standards	Use network or relational data structures for 1980s system	Investigate DBMS developed for ICAM based on existing software considering data integrity and security. Develop project standards for DBMS interface and use.
6. Software Testing and Tools	COBOL, FORTRAN validators	Use compiler Validation Use dynamic and static analyzers	Develop Validation for future languages. Develop test for accuracy of mathematical software. Develop acceptance tests for each major software package. Develop and implement software development tools.
7. Documentation	FIPS PUB 30 FIPS PUB 38 CAM-I Standards	Use FIPS PUB 38 and CAM-I to formulate specific ICAM documentation guidelines.	Disseminate project standards for documentation. Appoint a documents administrator to specify and maintain system design, documentation standards, and documented software.
8. Media	Magnetic Tape (X3.40, X3.14, X3.22) Paper Tape (X3.29, X3.1B) Punched Cards (X3.11, X3.21)	Use magnetic tape as primary media	Clarify DNC system interface in cooperation with EIA Committee IE-31

STANDARDS FOR COMPUTER AIDED MANUFACTURING

NC PART PROGRAMMING LANGUAGES

BACKGROUND

An NC machine tool accepts commands from a punched paper tape or from a computer to control the operations of that tool. These control signals are strings of bit patterns that are decoded by the tool into the proper locations, movements, and actions, to produce the desired part. Following a standard code for the different control signals, an operator can punch these values into a paper tape. Simply rerunning the paper tape into the NC tool allows the tool to produce automatically as many parts as desired while the operator is free to do other jobs.

For simple parts, and originally for all parts, the coding of the control tape is carried out directly according to the EIA standard tape formats. (RS-247C with RS-358 character code.)

As the parts to be made become more complicated, the programming becomes much more involved. Higher level NC programming languages have been developed for these more sophisticated cutting operations. These languages are typically made up on a number of English-like commands which are translated by a computer program (processor) into either the proper bit pattern for a particular NC machine tool, or into an intermediate machine-tool-independent data file (Cutter Location Data (CLDATA) file). This CLDATA file will then be fed into another computer program called a postprocessor. It is the function of the postprocessor to translate the cutter location data into the appropriate commands for the selected machine tool necessary to machine the desired part. The postprocessor also checks for various error conditions and produces the printed listing to assist the machine operator.

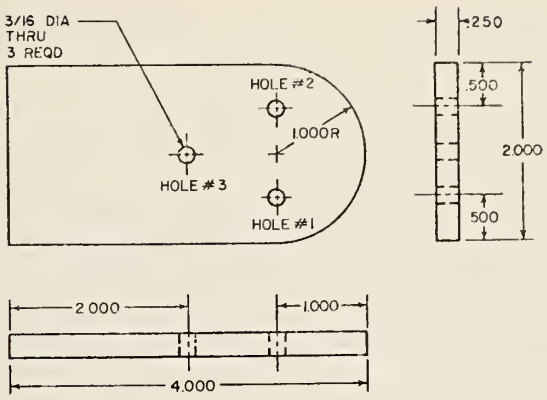
Thus, the CLDATA file is a machine-independent data file that describes in detail the path the cutting tool must follow to make the part. This file is created by a single processor.

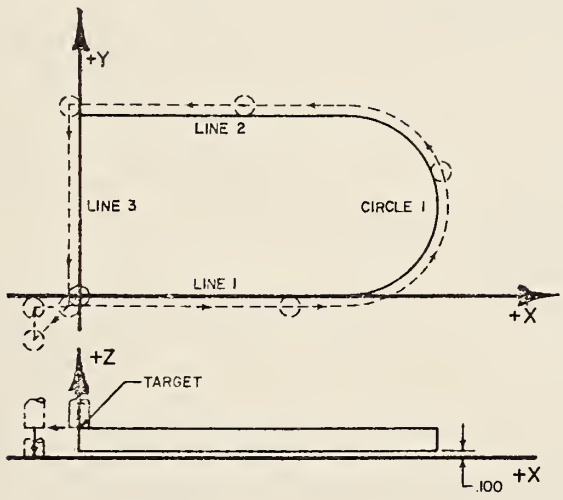
Since the postprocessor is dependent upon both the machine tool and controller, there are as many postprocessors as there are different models of machine tools and controls.

Although there exist over 40 NC programming languages, only two are in widespread, productive use. These are APT (Automatically Programmed Tools), the first of the higher level NC languages, and COMPACT II (COMputer Program for Automatically Controlling Tools). These two languages are representatives of two families of NC language processors. The APT family includes the APT, UNIAPT, and ADAPT processors, while COMPACT represents the COMPACT II, ACTION and SPLIT processors. Both of these language families are proceeding toward formal standardization. An example of each language family is given in the accompanying figures. A simple test part is shown in Figure 1 while the respective part programs are given in Figures 2 and 3.

APT STANDARDIZATION

The revised APT standard (X3.37) (presently undergoing final balloting) is expected in January of 1977. This standard is created and maintained by the American National Standards Institute (ANSI) Committee X3J7 under the Business Equipment Manufacturer's Association (BEMA). The standard is written in a meta-language format which is computer independent. This format gives a complete and vigorous definition of all elements of the language, permissible combinations of these elements, and the meaning of these combinations. While somewhat difficult to read the meta-linguistic format provides a concise and comprehensive technique to itemize all of the combinations and their meanings in a reasonable length document. The new standard will contain the original (X3.37-1974)


3/16 DIA
THRU
3 REQD
HOLE #2
1.000R
HOLE #3
HOLE #1
.250
.500
2.000
.500
2.000
1.000
4.000


+Y
LINE 2
LINE 3
CIRCLE 1
LINE 1
+X
+Z
TARGET
.100
+X

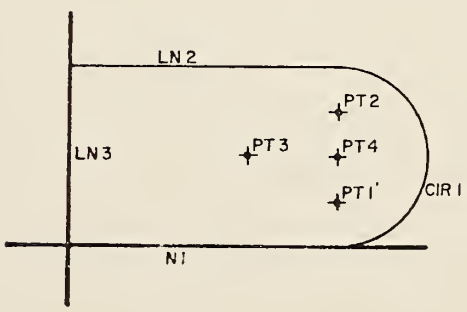

LN 2
PT 2
LN 3
PT 3
PT 4
PT 1'
CIR 1
N 1

Figure 1

```
10 PARTNO TEST1*
15 CLPRNT
20 CUTTER/.25
30 SYN/P,POINT,L,LINE,CI,CIRCLE,GT,GOTO,GF,GOFWD,GD,GODLTA,F,RADIUS,$
40 XA,XAXIS,YA,YAXIS
50 INTOL/.001
60 OUTTOL/.001
70 SP=P/0,0,.35
80 FEDRAT/5
90 L1=L/XA
100 L2=L/PARLEL,L1,YLARGE,2
110 L3=L/YA
120 C1=CI/3,1,1
130 P1=P/3,.5
140 P2=P/3,1.5
150 P3=P/2,1
160 CYCLE/MILL,0,0
170 $$ TARGET AT LEFT BOTTOM CORNER
180 FROM/SP
190 GD/-.5,-.5,0,150
200 GD/0,0,-.35,150
210 GO/TO,L1,150
220 TLRGT,GORGT/L1
230 GF/C1
240 GF/L2
250 GOLFT/L3,PAST,L1
260 $$ CUTTER MOVES TO HOME POSITION AND STOPS. CHANGE TO 3/16 DRILL
270 FEDRAT/.004,IPR
280 CYCLE/DRILL,0,.45
290 GT/P1
300 GT/P2
310 GT/P3
320 CYCLE/OFF,OMIT
330 END
340 FINI
```

Figure 2

```
MACHIN, KTGE7500
IDENT,NCLE TEST #1 COMPACT II
SETUP,EB,INDEX45,-.125LX,.125LY,10LZ,LIMIT(X-10/10,Y-10/10,Z-10/10)
$TARGET AT LEFT BOTTOM CORNER
BASE,XA,YA,ZA
PT1,3XB,.5YB,.35ZB
PT2,3XB,1.5YB,.35ZB
PT3,2XB,1YB,.35ZB
PT4,3XB,1YB,ZB
LN1,YB
LN2,LN1/2YL
LN3,XB
CIR1,PT4,1R
MTCHG,TOOL1,.25TD,5IPM,0GL
MOVE,-.5X,-.5Y
MOVE,-.350Z
MOVE,TOLN1
OCON,CIR1,CCW,S(TANLN1),F(TANLN2)
CUT,PASTLN3
CUT,PASTLN1
HOME,STOP
$CUTTER MOVES TO HOME POSITION AND STOPS. CHANGE TO 3/16 DRILL
MTCHG,TOOL2,(3/16)TD,1800RPM,0GL,.004IPR
DRL,PT1,.25THRU,.1CLEAR
DRL,PT2,.25THRU,.1CLEAR
DRL,PT3,.25THRU,.1CLEAR
HOME,STOP
END
```

Figure 3

standard for the processor, plus updates and corrections in addition to a standard for postprocessor language.

The new standard is unique in its consideration of the postprocessor language. This is the language which enables the control of the non motion functions of a machine tool such as choice of spindle speeds, control of coolant, and selection of cutting tools. Prior to this document guidelines for postprocessor language have been scanty with the result that developers of software programs have had to sometimes choose language syntax themsleves.

As various new hardware or electronic options were developed in the marketplace so were new APT language commands to control them. While the commands for a single function like a tool-change are similar among all postprocessors, minor differences exist in each software implementation. These differences have the effect of forcing a part programmer to choose a specific NC machine tool before he starts to develop the APT language to produce the desired item.

The lack of a fully specified APT language tends to defeat the intended universality of the higher level language concept. The design intent of APT was that a part program could be easily processed for any appropriate machine tool through the use of different postprocessors. Increasingly today one finds that not to be the case. Computer runs are aborted for trivial problems such as a command to postprocessor "A" calling for SPINDL/1000, CLW causing an error in postprocessor "B" which requires SPINDL/CLW, RPM, 1000. The revised APT standard is aimed at correcting this problem.

COMPACT II STANDARDIZATION

The COMPACT II/ACTION/SPLIT Standard proposal is currently under consideration by the X3J5 standards committee of CBEMA. SPLIT is the parent language of a group of languages comprising SPLIT, ACTION, COMPACT II in a father/son/grandson relationship. The languages are very similar, but the processors are quite different. It was decided in developing the standard that a standard CL (Cutter Location) output would be optional since this family of languages does not necessarily generate an intermediate data output medium.

The SPLIT processor is machine dependent and does not create an intermediate cutter location (CL) file. The ACTION and COMPACT II processors, however, are machine independent; but they work in conjunction with their respective postprocessors. In this integrated mode, each statement is processed into a CL file statement and then postprocessed by the selected postprocessor into a machine control format before the program moves to the next statement.

ACTION can be run on a minicomputer and in that situation operates in the re-entrant mode - i.e., all the statements in a program are processed and a CL file is generated; then that file is postprocessed to produce the machine control output.

The ANSI committee feels that to make intermediate output (CLDATA file) the mandatory output of the standard would be to deprive the users of many of the inherent economics of the languages. However, the committee is recommending that the intermediate data output be a user or implementor option, and where offered it should conform to the existing CLDATA Standard. The University Computer Corporation (UCC) COMPACT II processor already produces an intermediate data file in accord with the CLDATA requirement of the APT standard.

The long term objectives of the COMPACT II Standards committee are to provide most of the capabilities already present with APT or under research effort. These include working standards for graphic output, for incorporation of machining technology, for programming sculptured surfaces, and for the interface of the NC language to total CAD/CAM systems.

There are presently 1400 installations using the COMPACT II family of languages, representing 6000 NC machine tools. The five year projection (by 1981) estimates 3500 users (20,000 NC machine tools) in the US and 6000 users (40,000 NC machine tools) worldwide.

At present, about 50% of all NC machine tools are being programmed by computer assist. Of these, about 40% are being programmed by COMPACT II family and about 40% by APT with the remainder using the other 40 languages. The five year prediction is for 75% of all NC tools to use computer assisted programming with close to half in COMPACT II and half in APT.

Thus, even though the COMPACT II family is a late entry, (circa 1967 vs. 1950's for APT) it has quickly found widespread acceptance. The main reasons for this are several. The COMPACT II family is less sophisticated than APT and for that reason many users feel that for their more limited requirements that COMPACT is easier to learn. Lathe (2 axis) programming is much more efficient in COMPACT II because of certain language features not available in APT. Lathes represent 40% to 50% of all of the NC tools being shipped. COMPACT II has also been well provided on a time-shared remote service bureau basis by Manufacturing Data Systems Inc. (MDSI) with excellent support.

COMPARISON OF NC LANGUAGES

In March 1974 the Numerical Control Society submitted a final report on the US Army Electronics Command Numerical Control Language Evaluation. This study analyzed seven general purpose NC programming languages and presented data concerning their performance on ten test parts representative of Department of Defense workload. The test parts all of the milling-drilling-boring variety spanned the entire range from 2 axis to 5 axis complexity.

While no definite conclusions were reached in the study, sufficient data is presented and analysis factors explained that a prospective user can perform benefits analysis in the context of his own shop environment.

One fact is clear - that of the general purpose NC language processors now in widespread productive use only two language families are prevalent, APT and COMPACT. It is again only these two language forms that are being considered in government and national standardization activities. As such both merit the attention of the Air Force ICAM Program.

CRITERIA FOR NC LANGUAGE SELECTION

Several technical factors should be considered in choosing a programming language for numerical control:

Language Programming Capability
Processor Availability
Language Documentation
Processor Maintenance
Programming Time
Processing Costs
Proprietary Nature of Language

Study of NC languages must be placed into the perspective of the final goal of the Air Force program, an integrated computer base manufacturing system. It is expected that when this goal is realized, a designer may sit down at a computer terminal with a CRT, design some object, then allow the computerized manufacturing system to manage the supply of raw materials, schedule machines, decide on cutters, manage inventories and produce a final product while providing management and designers with the appropriate feedback information Critical interfaces in this final system should be identified now and carefully standardized so that a workable system can be developed. In the area of the actual machining and forming of parts, the most important interface is between the CAM (computer aided manufacturing) system and the actual production machines. This interface is defined by the CLDATA file. This is the standardized part description data that describes exactly how to make any part. A postprocessor of any machine tool will convert this standardized data to the specific command statements necessary for that particular tool to make the part. The CLDATA file can also be used by graphics devices to display in visual form information concerning the part.

At the present time this CLDATA file is generated by the NC programming language APT, and is being considered as an optional requirement for COMPACT II. It is the standard for the International Standards Organization (ISO). A designer now gives a part programmer either his own drawings or design drawings made with varying degrees of computer assist. The part programmer then generates the necessary code to make the part. This is passed through a processor to create (in APT) a CLDATA file which should be a totally machine-independent representation of the part. This is customised to the requirements of the individual machine tool by putting the CLDATA file through the postprocessor for that tool. Eventually the part programmer should be eleminated with the CAM system providing the CLDATA file from the designer's requirements. Thus, while the NC programming language standard is important, indeed crucial during the interim stage, its importance will decrease as the full CAM system is realized. The CLDATA file, however, will become the link between the CAM system and the real world of production. If this CLDATA file can be properly standardized, it can be the common data base between any CAM system and any set of machine tools or any programming language and any CAM system or machine tool. It would make it easy for any machining facility to produce any parts regardless of their own CAM capability, merely by having access to the CLDATA file. It would allow government, for instance, to make replacement parts or additional units from CLDATA files without having to attempt to access contractor CAM systems that might be proprietary. Again this interface, the CLDATA file, is considered one of the most crucial for a truly flexible computer aided manufacturing system.

RECOMMENDATIONS

With this in mind our recommendations considering NC part programming languages follow.

(1) If a single part programming language is desired to cover all applications then this language should be APT. APT produces the CLDATA file standard. It is the most sophisticated including such unique capabilities as producing part programs for milling a non-formula or sculptured surface (a surface defined by a lattice of coordinate points) such as found commonly on aerospace parts. Thus far it is the only language for which there is a formal draft standard. There are several areas of research and development of advanced capabilities such as process planning, geometric modeling, and sculptured surfaces that will be compatible with existing APT processes.

(2) If more than one language can be considered, then it is recommended that both APT and COMPACT II be used. APT provides the sophistication for complex parts. COMPACT II, however, offers significant advantages in speed and ease of programming of simpler parts and expecially lathe work.

(3) If COMPACT II is included as a standard language then the capability to produce a standard CLDATA file must be included. This would allow part programming in either APT or COMPACT II with their CLDATA files to be the common interface to the production machines.

(4) While the current standardization activity with the APT postprocessor language is encouraging, it falls short of the capability truly needed by the Air Force in manufacturing. Even the most recent proposed standard for APT allows too much latitude in the choice of language syntax. Anything short of a complete and comprehensive language specification obviates the possibility of being able to rapidly and easily exchange NC workload among functionally equivalent machines. This capability is central to the concept of integrated and flexible manufacturing. The Air Force can and should provide the impetus to a widespread implementation of a complete government standard on postprocessor language and philosophy of post-processing which would bring about this flexibility. Only with this technique can NC data be made transferable among different machines, different shops and different contractors.

(5) Further work on additional language capabilities for both APT and COMPACT II is being carried out by the relevant ANSI committees on NC part programming languages. It is recommended that the Air Force monitor this work and help provide direction for the implementation in CAM systems. Work is progressing in the areas of a) sculptured surfaces, non-analytical sculptured shapes (shapes arrived at by sculpturing processes), unconventional analytical shapes (e.g. parametric surfaces), and any combination of these two; b) bounded geometry, 3-dimensional modeling capability within the computer. Objects would be represented and manipulated as bounded, closed entities rather than as bounded by a set of possibly infinite faces combined in specific ways; c) lathe language - a study of the various capabilities of several languages in their ability to efficiently program lathes which account for close to half of all NC machine tools.

COMMENTS

The emphasis of the proceeding report on NC Programming Language Standards is the important interface between future CAD/CAM systems and the production tools. The CLDATA file appears to be a good starting point for the development of this crucial interface standard. The recommendations above suggest some important additions necessary if real flexibility is to be obtained at this interface.

There are additional considerations which will be mentioned here.

The CLDATA file is not a totally independent description of the necessary commands and cutter path. When the program is written, certain data such as the diameter of the cutting tool, the length of the tool, etc. are included in the program and these affect the cutting path motions. The CLDATA file with postprocessor commands can be used as a description of the machine tool operations only as long as these additional parameters are kept constant. If a shop does not have the

correct size cutter it would be advantageous if the part program could be modified to accomodate the cutter size availabe. For contouring operations, this implies new geometric calculations and the need for source code modifications. It would thus be advantageous to have the NC system on-line in a DNC configuration. This is a reasonable plan for systems for the 1980's. This would require better identification of relevant statements in the CLDATA file, perhaps through flags, comment statements, etc. to allow for possible editing.

REFERENCES

(1) American National Standard APT - Proposed Revision, March 1975, American National Standards Institute, 1430 Broadway, New York City, New York 10018

(2) Numerical Control Language Evaluation, Numerical Control Society, March 1974, 1201 Waukengan Road, Glenview, Illinois 60025

(3) CAM-I Special Projects 1976, PR-75-ASPP-01, Computer Aided Manufacturing-International, Inc., 611 Ryan Plaza Drive, Suite 1107, Arlington, Texas 76012

(4) Proposal for the creation of an X3 Standards Committee covering the COMPACT II, ACTION, and SPLIT family of Numerically Controlled Machine Languages, 1975; Submitted to: American National Standards Committee X3, Secretary X3, CBEMA; By: Manufacturing Data Systems, Inc., 320 North Main Street, Ann Arbor, Michigan 48104; Sundstrand Machine Tool Compary, 3625 Newburg Road, Bervidere, Illinois 61008.

(5) The letter of August 27, 1975 by Elliot Brebner, Chairman of X3J7, in response to the SPARC committee requesting that X3J7 comment on the COMPACT II, ACTION, SPLIT proposal for the formation of a standards committee.

(6) Computer Software for Numerically Controlled Manufacturing - 1973; By: Bradford Smith, Report # NSRDC 4327, Computer Aided Design Division, Naval Ship Research & Development Center, Bethesda, Maryland 20084

SUMMARY DATA SHEETS

The following Data Sheets summarize these standards which apply to NC Part Programming Languages.

1. Designation: ANSI X3.37-1974

2. Title: Automatically Programmed Tool (APT)

3. Maintenance Authority: ANSI X3J7 and ISO/TC97/SC9

4. Scope: Programming language used for Numerical Control (N/C) machine tools in discrete part manufacturing and tooling manufacturing, for surface definition, and as a subset of graphics and process planning programs for all of the above.

5. Relationship to Other Standards:

   ARELEM - 1971 (subset)
   ARLM1 - 1975 (subset)
   SSX5 - 1975 (extension)
   CASPA - 1975 (pre-subset)
   ADAPT - (subset)
   UNIAPT - (Comparable except for the size of computer and mode of operation)
   ISO/DIS 3592 - 1975 (Pseudo-subset) the ISO CL DATA standard derived from APT.

6. Competitive Standards: SPLIT/ACTION/COMPACT II Languages

7. Standardization Status: In 1963, the Business Equipment Manufacturer's Association (BEMA) sponsored the formation of the American National Standards Institute (ANSI) subcommittee X3J7 to create a standard for the APT language. The standard was submitted for approval in May, 1973. The first APT Language Standard was published by ANSI in June 1974. It is designated ANSI X3.37. This standard was revised in March 1975 and a draft (X3J7/55-80) has been approved with publishing expected by January 1977. The revised standard will be designated ANSI X3.37-1977. The major revision is the addition of the APT N/C post processor language in the standard.

   FIPS Task Group 19 is studying the suitability of this revised form of the APT Standard for use as a Federal Standard. A draft Federal Standard is expected by September 1977.

8. Implementation Status: The APT Language is implemented on IBM 704, 709, 7090, 360, 370, UNIVAC 1108. General Electric's international computer service network. APT is also fully or partially implemented on Fujitsu, Control Data Corporation, Siemens, and English Electric Computers.

9. Known Manufacturing Uses: Approximately 20% of the parts made on N/C tools are programmed in APT.

10. Known Sources of Information:

    Computer and Business Equipment Manufacturers Association (CBEMA)
    Secretariat of ANSI X3
    1828 L Street, NW
    Washington, D.C. 20036

    CAM-I, Inc. (Computer Aided Manufacturing-International, Inc.)
    611 Ryan Plaza Drive, Suite 1107
    Arlington, Texas 76012

11. Probable Sources of Information:

IBM (International Business Machines) Corp.
Data Processing Division
1133 Westchester Avenue
White Plains, New York 10604

12. Bibliography:

ANSI X3.37 - 1974
IBM SYSTEM/370 APT-BP Numerical Control Processor General Information Manual
IBM SYSTEM/370 APT-IC & APT-AC Numerical Control Processor General Information Manual

13. Comments: APT is the first of the N/C Languages, provides the most sophisticated capabilities, and is presently the most widely used, with about 20% of all N/C machined parts being programmed in APT. Development work includes a geometric modeling project for processing a wide variety of engineering shapes, an improved Arithmetic Element, and a sculptured surfaces project to extend the geometric capability of APT to unconventional analytical as well as non-analytical shapes.

Developmental work is primarily carried out by CAM-I, Inc., which has taken over the work of the APT Long Range Program (ALRP) formerly at the Illinois Institute of Technology Research Institute (IITRI).

Major problems still remain with post processor and controller source code incompatibilities that make it impossible to transfer either Cutter Location (CL) file tapes or machine tapes from one machine tool or facility to another without at least some modifications.

1. Designation: N.A.

2. Title: COMPACT II/ACTION/SPLIT (COMputer Program for Automatically Controlling Tools II/ACTION/Sundstrand Processing Language Internally Translated)

3. Maintenance Authority: ANSI X3J5, Manufacturing Data Systems Inc. (MDSI)

4. Scope: Programming Language for describing operations for numerically controlled machines.

5. Relationship to Other Standards: N.A.

6. Competitive Standards: APT is the main competitive language; there are approximately 40 other NC programming languages.

7. Standardization Status: Initial proposal for a standard was reviewed by CBEMA SPARC committee June 17, 1975. This committee recommended that a study group be formed and the X3/SPARC Study Group held its first meeting September 30, 1975. As a result the COMPACT II/ACTION/SPLIT Standard proposal was modified and forwarded to X3 with the recommendation that an X3J* standards committee be formed in order to produce a standard within 12 to 24 months. X3J5 held its first meeting in March, 1976.

8. Implementation Status:

   SPLIT has be implemented on the DEC PDP 11/20 and the IBM 360/30.
   ACTION has been implemented on the DEC PDP 11, DEC PDP 10, IBM 360, IBM 370.
   COMPACT II is available only in the remote time-sharing mode on two world-wide networks maintained by Manufacturing Data Systems Inc. (MDSI)

9. Known Manufacturing Uses: A language used in programming of Numerical Control (N/C) machine tools. At present there are 1400 users of the COMPACT II/ACTION/SPLIT family of languages representing over 6000 N/C machine tools. This results in about 20% of the machine parts made on all N/C tools are programmed in this family of languages.

10. Known Sources of Information:

    Robert F. Guise, Jr.
    Director - New Product Planning
    Manufacturing Data Systems Inc. (MDSI)
    320 N. Main St.
    Ann Arbor, Michigan 48104

11. Probable Sources of Information:

    Mr. Harold Baeverstad
    Vice President Manufacturing
    Sundstrand Machine Tool Division of the Sundstrand Corp.
    Newburg Road
    Belvidere, Illinois 61108

    Mr. Richard A. Stitt
    President
    NCCS-WORD, Inc.
    23500 Merchantile Blvd.
    Cleveland, Ohio 44122

12. Bibliography:

MDSI COMPACT II Programming Manual, March, 1973
Sundstrand Machine Tool OM 3 Omnimill SPLIT Programmers Manual
TS 60030 SPLIT Vocabulary Manual
ACTION Programming Manual for Mill and Drill
ACTION Central, Description of the System
ACTION N/C Time Sharing
5-Axis Action 5A:001

13. Comments: SPLIT is the parent language of a group of languages comprising SPLIT, ACTION, and COMPACT II in a father/son/grandson relationship. The languages are very similar, but the processors are quite different.

It was decided in developing the standard that a standard CL (Cutter Location) Data output would be optional since this family of languages does not necessarily generate an intermediate data output medium.

Approximately 30% to 50% of all N/C machine tools are programmed by computer assist. Of this number approximately 30% to 40% are programmed by COMPACT II/ACTION/ SPLIT family of languages. This high use is mainly a result of the efficiency of programming 2 axis machines (lathes, which account for 40% of the N/C tools) and the ease of programming simple parts as compared to the greater programming effort required with the more sophisticated APT language.

CONAPT is a member of this family, not the APT family.

STANDARDS FOR COMPUTER AIDED MANUFACTURING

CAD/CAM INTERFACE STANDARDS

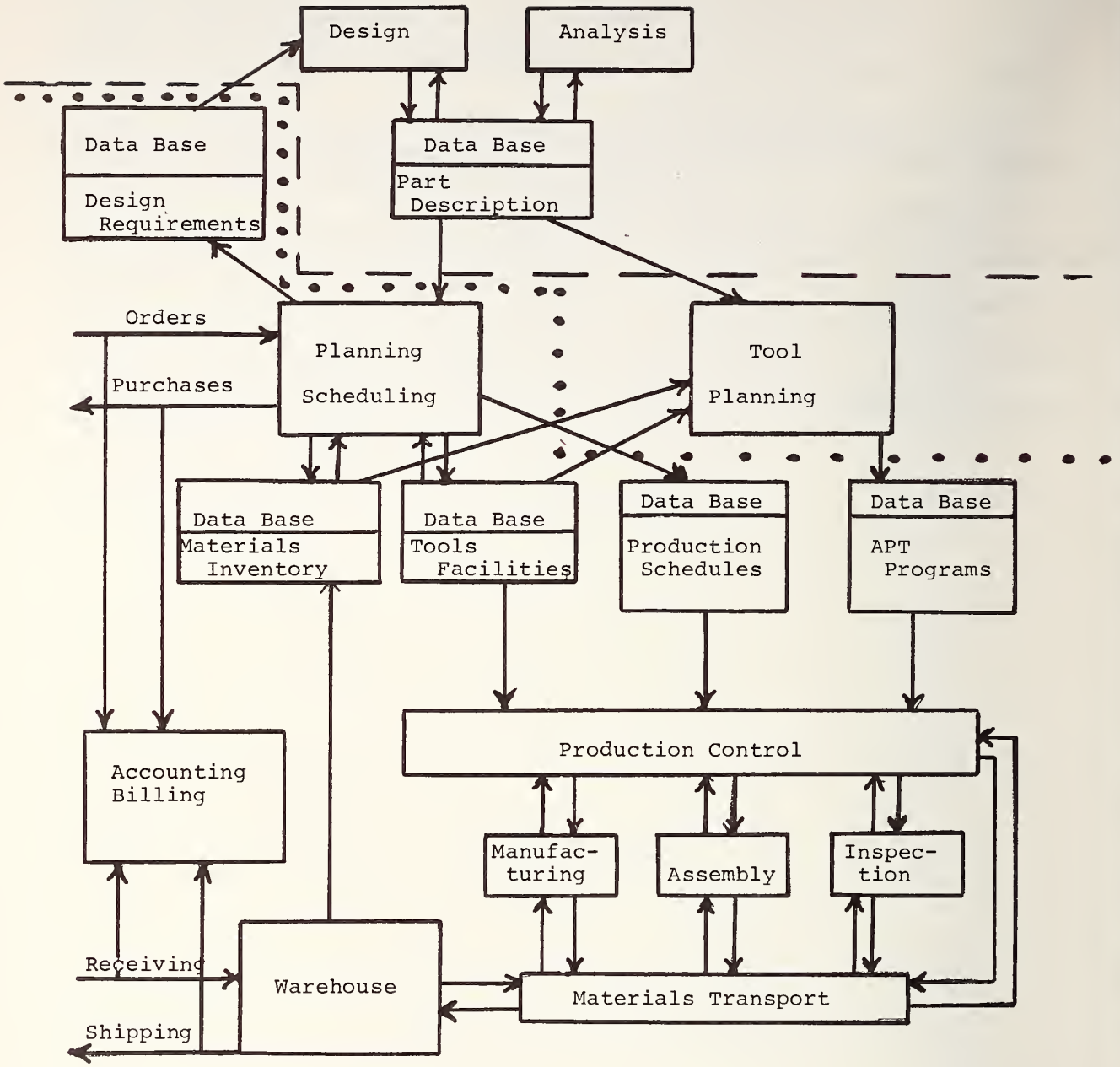


Figure 1

SCHEMATIC LOCATIONS OF THE CAD/CAM INTERFACE

INTRODUCTION

The CAD/CAM interface is a boundary, as yet ill defined, across which information must be communicated. The flow of information is primarily in the CAD→CAM direction, although ideally there is a reverse flow giving the design or process engineer information concerning tool availability, material inventory, etc. The simplest, historical design/manufacturing interface was the set of engineering drawings describing the part to be manufactured. In a CAM system, the interface is the appropriate data base representing the same data as the part drawings.

APT AS A DE FACTO CAD/CAM INTERFACE

There presently exists no consensus as to what the CAD/CAM interface is or precisely when it should be drawn. For example, are Automatically Programmed Tool (APT) programs part of the design process or the manufacturing process? Many small stand-alone interactive CAD systems produce APT source code, APT CL file data or machine tapes as direct output. This data is then carried to a manufacturing installation where it is put through a processor and/or post processor (if necessary) and used to control NC machine tools. The APT part description is thus a direct CAD/CAM interface for small systems.

In larger installations, where CAD/CAM is more integrated, the data base which describes the physical parameters of the parts to be manufactured is usually considered to be the CAD system output. APT tool programming is treated as one part of Process Planning (part of CAM, not CAD). The CAD/CAM interface is thus considered the drawing or data base describing the part. In Figure 1, if APT programming is considered to be a part of manufacturing, the CAD/CAM interface can be drawn as the dashed line. If, however, APT is included in design, then the interface can be drawn as the dotted line in Figure 1. In either case, it is possible, given the structure shown in Figure 1, to draw the CAD/CAM interface such that it cuts only the outputs of data bases. This would appear to be a useful concept in that it makes for clearly defined interfaces both physically and logically.

STANDARDS ACTIVITY IN DIGITAL REPRESENTATION OF PHYSICAL OBJECT SHAPES

Whether any particular CAD/CAM system adopts the configuration of Figure 1 or some other, it is clear that the data base consisting of a numeric description of physical objects is central to the entire CAD/CAM processes. This data base provides the working input to CAD displays and to CAD analysis programs. Indeed, the entire CAD process does nothing more than generate, analyse, and manipulate this data base. Once finalized, the part descriptor data base provides the primary input to APT tool programs, planning and scheduling programs, and eventually to inspection and quality assurance programs.

Thus, the part description data base is central to the entire CAD/CAM concept and, in large measure, will define the CAD/CAM interface. This implies that efforts to develop standard methods for representing part shapes, dimensions, tolerances, materials surface finishes, etc. are prerequisites to developing standards for CAD/CAM interface. The American National Standards Institute (ANSI) Y14.26 subcommittee on Computer Aided Preparation of Product Definition Data is presently working on a Y14.26.1 standard for the Digital Representation of Physical Object Shapes.

The stated aim of this standard is to facilitate the communication of physical object shape descriptions among CAD/CAM programs and data bases of organizations engaged in interfacing activities such as contracting and subcontracting. The approach is to abstract the spatial property of shape

by representing physical objects as geometric solids. The problem then reduces to describing solids.

A solid may be considered to be a geometric structure constructed out of building blocks of simpler geometric entities. The description of that solid is then an information structure constructed out of building blocks of digital data. This leads to a hierarchy of building blocks.

At the lowest level in the geometrical hierarchy is the point. A point moving along a trajectory generates a line, a line moving along a trajectory generates a surface. A surface moving from a start to an end surface generates a solid element. A succession of solid elements can be joined to form a complex solid. An example of this method of generating and describing physical object shapes is shown in Figure 2.

Generating a trajectory requires a rule (or set of rules, procedures, or equations) which describes the motion of the generatrix (point, line, surface, solid element). Each element in the hierarchy depends on the available set of subelements and generating procedures in the lower levels of the hierarchy. A judicious choice of lower level subelements can produce a very broad variety of complex shapes.

This work is proceeding steadily, although rather slowly. But even when this standard is formalized, it will represent only a first step toward solving the larger problem of completely describing physical objects.

A related effort is currently being funded by Computer Aided Manufacturing-International, Inc. (CAM-I). The CAM-I Geometric Modeling Project is attempting to develop 3-dimensional modeling tools based on digital descriptions of geometric shapes. On August 25, 1976, CAM-I acccepted a bid from Sof-Tech, Inc to develop a Geometric Modeling System (GMS). This will be a generic system capable of incorporating software modules for part description languages, geometric modeling mathematics, display and communication technology, and end use applications. GMS is to "conform to ANSI standards and be as computer-independent as possible."

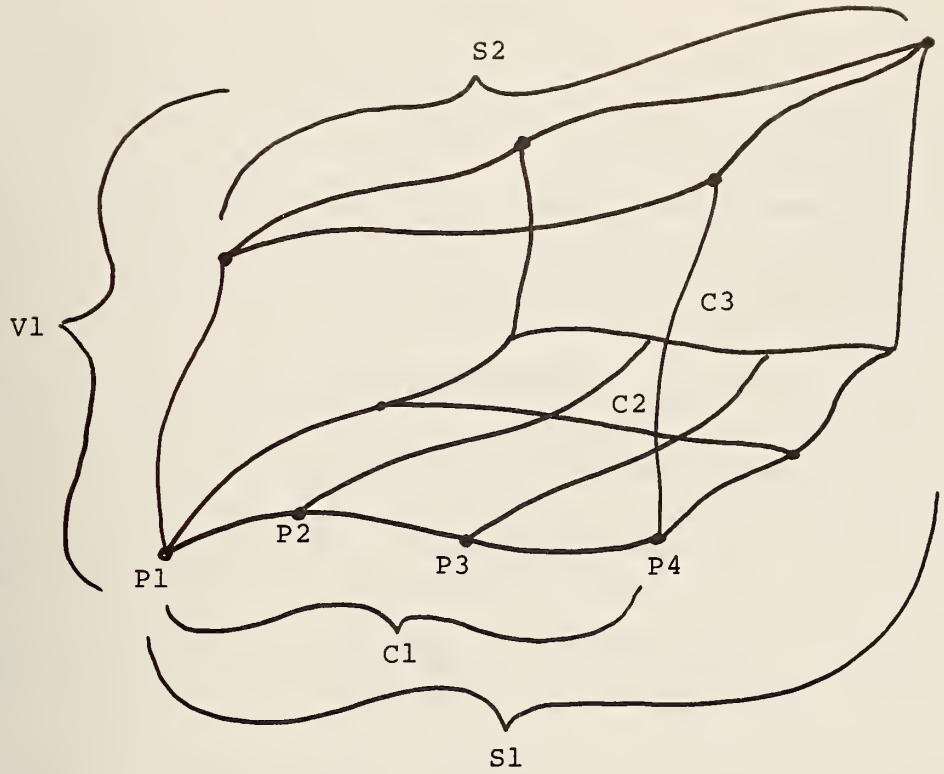
THE CAD/CAM INTERFACE IN PRINTED CIRCUIT BOARD MANUFACTURING

The Institute of Printed Circuits has published a standard entitled "End Product Description in Numeric Form for Printed Wiring Products."

This standard not only defines methods for describing geometric shapes of printed wiring boards but prescribes record formats for describing the end-product in digital form. An example of four records describing four segments of a printed wiring circuit is shown in Figure 3. This digital data, when recorded on punched cards or magnetic tape, contains sufficient information for tooling, manufacturing and continuity testing of printed wiring products. These formats thus may be used for transmitting information between the designer and the manufacturing facility after the design has been completed by a computer-aided process. Such data format standards are particularly useful when the manufacturing process includes numerically controlled machines.

The data records specified in this standard are general, not in any particular machine language, and can be used for both manual and machine interpretation. Thus each facility can produce an end-product from the data by the most efficient method available.

Unfortunately, this standard addresses only a tiny fraction of the set of manufactured products, namely two dimensional printed circuit boards. Nevertheless, it is complete, is presently in use, and does deal with the problem of describing a physical object with sufficient completness to define not only the manufacturing process, but the inspection and acceptance testing process as well.



C1 = G06 (P1, P2, P3, P4)

The curve C1 is generated by the parametric cubic operator G06 operating on the points P1, P2, P3, P4.

S1 = G05 (C1, C2, C3)

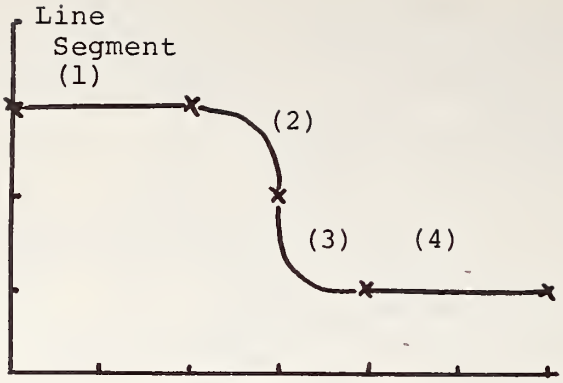
The surface S1 is generated by the operator G05 operating on the curves C1, C2, C3.

V1 - G04 (S1, S2)

The solid V1 is generated by the operator G04 operating on surfaces S1, S2.

Figure 2

ANSI Y14.26 METHOD OF DESCRIBING PHYSICAL OBJECT SHAPES



Op. Code	Data Field						Record #
	Start Point		End Point				
111	X+00	Y+03	X+02	Y+03			(1)
	Circle Center		Start Angle	Finish Angle	Radius	Direc-tion	
021	X+02	Y+02	X+90	Y+00	X+01	Y+01	(2)
021	X+04	Y+02	X+180	Y+270	X+01	Y-01	(3)
011	X+04	Y+01	X+06	Y+01			(4)

Set size of XY fields
1 = linear 2 = circular interpolation
1 = begin line 0 = continue line

Figure 3

EXAMPLE OF IPC STANDARD REPRESENTATION OF PRINTED WIRING CIRCUIT

POTENTIAL IMPACT OF NASA'S IPAD PROJECT

The work on an Integrated Program for Aerospace-Vehicle Design (IPAD) being funded by NASA Langley Research Center is not a standard by the common definition. It is merely one more integrated software system which attempts to computerize, in so far as possible, company-wide design information processing. IPAD will be composed of 1) executive software that will control user-directed processes through interactive interfaces with a large number of terminals in simultaneous use by engineering and management personnel, 2) a large number of utility software packages for information manipulation and display functions, and 3) data management software to store, track, and retrieve large quantities of data in multiple storage devices.

However, IPAD is different from other integrated software design systems in that it is scheduled to be released by NASA to become public domain under NASA's FEDD (For Early Domestic Dissemination) policy. If IPAD is a successful system it will undoubtedly be widely used by many industries, especially those which are too small to afford to develop their own internal CAD systems. The formats used by IPAD for digital description of physical objects shapes, and even for describing end-products, will thus become common usage in many CAD/CAM systems in the future.

The result will be that even though IPAD does not pretend to be a standards setting project, it nevertheless will set precedents which are almost certain to become de facto standards for data base formats, man-machine interfaces, and eventually CAD/CAM interfaces.

There will probably arise many situations where IPAD data bases will not conveniently conform to standards being developed under ANSI Y14.26.1. The temptation will be to ignore the ANSI standards since they have not yet been formally adopted. Every effort should be made to resolve such conflicts whenever they arise for otherwise the general applicability and usefulness of both IPAD and Y14.26.1 will be reduced. The result will be that future CAD/CAM systems such as the ICAM system of the US Air Force will be adversely impacted.

The Air Force should take every effort to avoid such conflicts, working closely with NASA in the manner outlined in the existing Memorandum of Agreement between NASA and the Air Force.

SUMMARY

To date, all operational CAD/CAM systems have adopted ad hoc techniques custom tailored to specific applications. To some extent this is acceptable as long as a CAD/CAM installation is confined to a single plant or a single company where local custom can serve as an ad hoc standard. It is, however, completely unacceptable in a wider context where many different contractors and subcontractors will be required to use the same numeric descriptors for competitive bidding and for manufacturing operations. For a project such as the Air Force is presently contemplating, it is critical that efforts to achieve systematic set of numerical product descriptors be given top priority. Full cooperation and support should be given to the ANSI Y14.26 subcommittee as well as to the CAM-I Geometric Modeling Project. Close liaison should be maintained with the NASA's IPAD and every effort made to see that conflicting and competing standards do not proliferate.

RECOMMENDATIONS

It is recommended that the Air Force:

1. Maintain close liaison with the ANSI Y14.26 subcommittee.

2. Insist that all of its contractors adhere to the ANSI proposed standards whenever possible.

3. Maintain its close liaison with the IPAD project as outlined in its present Memorandum of Agreement with NASA.

4. Be aware of potential conflicts with IPAD and take whatever steps possible to prevent serious incompatabilities from developing.

5. Monitor the CAM-I Geometric Modeling Project to identify any compatability problems that may develop.

6. Insist on the use of the IPC-D-350A standard in future wedges relating to electronics or systems including printed wiring products.

REFERENCES

(1) Statement of Work - Development of Integrated Program for Aerospace Vehicle Design (IPAD), Ap 15, 1976, Langley Research Center, Langley, VA.

(2) Product Manufacturing Interface, October 1976 D6-IPAD-70011-D, NASA Langley Research Center

(3) End Product Description in Numeric Form for Printed Wiring Products, IPC-D-350A, September 1974, Institute for Printed Circuits

(4) Digital Representation of Physical Object Shapes, ANSI Y14.26.1 Draft Report, June 1976, American National Standards Institute, New York City, 10018

(5) CAM-I Special Projects, 1977, PR-76-ASPP-01, Computer Aided Manufacturing-International, Inc., Arlington Texas, 76012

(6) Minutes of Geometric Modeling Project Meeting, M-76-GM-01 held August 24-26, Rochester, N.Y., CAM-I, Arlington, Texas 76012

STANDARDS DATA SHEETS

The following Data Sheets summarize the standards which apply to the CAD-CAM Interface.

1. Designation: IPAD

2. Title: Integrated Program for Aerospace-Vehicle Design

3. Maintenance Authority:

Boeing Commercial Aircraft Co.
P.O. Box 3707
Seattle, Washington 98124

NASA Langley Research Center
Hampton, Virginia

4. Scope: IPAD is not a standard by the common definition. It is an integrated software system to computerize, insofar as possible, company-wide design-information processing. IPAD will be composed of 1) executive software that will control user-directed processes through interactive interfaces with a large number of terminals in simultaneous use by engineering and management personnel, 2) a large number of utility software packages for information manipulation and display functions, and 3) data management software to store, track, and retrieve large quantities of data in multiple storage devices.

IPAD is scheduled to be released by NASA to become public domain under NASA's For Early Domestic Dissemination (FEDD) policy. If it is widely used by industry IPAD may set de facto standards for data base formats and for the man/machine interfaces.

5. Relationship to Other Standards: N/A

6. Competitive Standards: N/A

7. Standardization Status: N/A

8. Implementation Status: IPAD is now being implemented. It will be released in three stages on two different host computer systems.

Release	1	Host	1	June 1978
"	1	"	2	Dec 1978
"	2	"	1	May 1979
"	2	"	2	Nov 1979
"	3	"	1	June 1980
"	3	"	2	Dec 1980

9. Known Manufacturing Uses: IPAD will be used in aerospace design to provide executive control, data management, and display utilities for engineering and management programs.

10. Known Sources of Information:

Robert Fulton, or Susan Voigt
NASA Langley Research Center
Hampton, Virginia 23665
804/827-2887, x3401

R. E. Miller, Jr.
IPAD Program Manager
Boeing Commercial Airplane Co.
P.O. Box 3707
Seattle, Washington 98124
206/237-8223

11. Probable Sources of Information: N/A

12. Bibliography:

Feasibility Study on an Integrated Program for Aerospace-Vehicle Design (IPAD), The Boeing Company, Contract NAS1-11441, 1973

Feasibility Study on an Integrated Program for Aerosapce-Vehicle Design (IPAD), General Dynamics/Convair, Contract NAS1-11431, 1973, NASA CR 132401-06.

IPAD Prospectus, NASA Langley Research Center, February 10, 1975.

NASA Request for Proposal 1-15-4934 Development of Integrated Programs for Aerospace-Vehicle Design (IPAD) May 16, 1975.

Boeing Technical Plan-Review D6-IPAD 70002-PS, May 24, 1976.

13. Comments: The Air Force has an memorandum of agreement with NASA to insure the compatability of the IPAD and CAM systems.

1. Designation: Institute for Printed Circuits Standard IPC-D-350A

2. Title: End Product Description in Numeric Form for Printed Wiring Products

3. Maintenance Authority:

   Institute for Printed Circuits
   1717 Howard St.
   Evanston, Illinois 60202

4. Scope: Describes record formats for defining end-product description data in digital form. This digital data, when recorded on punched cards or magnetic tape, contains sufficient information for tooling, manufacturing, and continuity testing of printed wiring products. These formats may be used for transmitting information between the designer and the manufacturing facility when the design has been formed by a computer-aided processes. These formats are also useful when the manufacturing process includes numerically-controlled machines. The data record is not in any particular machine language and can be used for both manual and computer interpretation.

5. Relationship to Other Standards: IPC-D-350A contains the following standards:

   Institute of Printed Circuits:
   - IPC-T-50 Terms and Definitions
   - IPC-D-310 Suggested Guidelines for Artwork Generation and Measurement Techniques for Printed Circuits
   - IPC-D-390 Guidelines for Design Layout and Artwork Generation on Computer Automated Equipment for Printed Wiring

   Department of Defense
   - MIL-STD-429 Printed-Wiring and Printed-Circuit Terms and Definitions

   American National Standards Institute
   - ANSI X3.22 Recorded Magnetic Tape for Information Interchange
   - ANSI X3.26 Hollerith Punched Card Code

   American Society for Testing and Materials
   - E380-74 Metric Practice Guide

6. Competitive Standards: None

7. Standardization Status: IPC-D-350 released August, 1972; IPC-D-350A revised and enhanced, released September, 1975.

   The ANSI Y14.26.2 Subcommittee is presently considering revising IPC-D-350A and reissuing it as a joint ANSI/IPC standard under the designation, ANSI Y14.26.2/IPC-D-350B. This revision will not make any change in the data formats; the only changes will be in the text of the descriptive narrative. Expected release data of this new standard is September, 1976.

8. Implementation Status: Computer-Vision, Inc. has implemented a translator for the standard to sell with their printed wiring manufacturing equipment.

   Bendix and Sandia are negotiating with Applicon for a similar translator.

9. Known Manufacturing Uses: The National Security Agency (NSA) has made IPC-D-350A a requirement for all suppliers of printed circuits and printed circuit equipment.

10. Known Sources of Information:

Timothy Ristine, Chairman of ANSI Y14.26.2/IPC-D-350B
Multiwire-New England
491 Amherst St.
Nashua, N.H. 03060
603/889-0083

11. Probable Sources of Information: N/A

12. Bibliography: N/A

13. Comments:

1. Designation: ANSI Y14.26.1

2. Title: Digital representation of Physical Object Shapes

3. Maintenance Authority:

   American National Standards
   Y14 Committee on Engineering Drawing and Related Documentation

   Subcommittee Y26 on Computer Aided Preparation of Product Definition Data

4. Scope: To establish a standard method of describing physical object shapes to facilitate communication of physical descriptions among computer users.

5. Relationship to Other Standards: Not yet defined. However, the proposed Y14.26.1 standard for a computer-readable format undoubtedly must subsume such currently used standards as MIL-D-1000, Military Specification for Engineering Drawings, and MIL-D-100A, Military Specifications for Engineering Drawing Practices.

   In addition, Y14.26.1 must be compatible with other ANSI Y14, Y10, Y32, and Z32 standards on drafting practices, graphical symbols and letter symbols.

6. Competitive Standards: None

7. Standardization Status: A technical report defining the geometrical foundations of Y14.26.1 was released in June, 1976. A draft of the Y14.26.1 standard will be submitted to the committee for a vote by Spring 1977 and will be released some months thereafter.

8. Implementation Status: N/A

9. Known Manufacturing Uses: Computer aided preparation of engineering drawings.

10. Known Sources of Information:

    S. Hori
    Leader of Y14.26.1 and 2
    McDonnell Douglas Corp.
    Dept. H213, Bldg. 107, Rm. 227
    P.O. Box 516
    St. Louis, MO 63166
    314/232-7286

11. Probable Sources of Information: N/A

12. Bibliography:

    Informational Report on Digital Representation of Physical Object Shapes, American National Technical Report ANSI Y14.26, 1 June, 1976
    Design/Manufacturing Interface, Aerospace Industries Association Report on Project MC 75.4, October, 1975.

13. Comments: This work does not yet represent a formal standard and has had only limited testing. The concepts promise to be extremely useful in constructing and transferring data on geometric objects.

STANDARDS FOR COMPUTER SYSTEMS

COMPUTER AND COMMUNICATIONS INTERFACE STANDARDS

INTRODUCTION

For purposes of the following discussion, an interface is defined to be the point of interconnection between two logically and physically separate components to enable the interchange of information. Depending upon the operational capabilities and functional complexities of the components, specification of an interface may require the definition of parameters and performance characteristics at several levels.

At the most basic level, for example, the physical interconnection of two components requires that they be electrically and mechanically compatible at the interface point, i.e., the signalling voltages and currents presented at the interface by each component must be compatible with the impedances and receiving circuit sensitivities of the other and the two interconnection plugs must mate. In addition, also at the basic level of interface definition, it is essential for information interchange that the components be functionally compatible, i.e., every function required by one component must be generated and presented at the interface in proper sequence by the other.

For some kinds of relatively unsophisticated equipment, conformance to the basic electrical, mechanical, and functional interface characteristics is sufficient to ensure operation. Complex systems also require that higher level operational and procedural definitions be provided. At the highest level where the components being interconnected have a range of operating capabilities, the formats and information transfer sequences must be also defined to ensure component interoperability.

There are generally three different kinds of interfaces that have been established for ADP systems that govern the interconnection of these systems with external devices and facilities and which enable the input/output interchange of internally stored information with the data collection, storage, or distribution environment external to the ADP system. The three kinds of interfaces are for:

(1) Computer peripheral devices, such as magnetic tape or disk that may serve both as intermediate or long term storage as well as a means for the direct input and output of data.

(2) Instrumentation and control devices that may be employed in a laboratory experiment or process control environment where the ADP system collects data produced by environmental or positional sensors and as a result of processing this data generates correctional control sequences to operate other machinery or equipment involved with the performance of the process.

(3) Communications, where the ADP system is to be interconnected with analog or digital telecommunication facilities in a teleprocessing environment.

For each of these three different kinds of interfaces, industry or national standards are being developed, or in some cases have already been approved, that specify the interfaces sufficient to ensure that components furnished by different suppliers can be interconnected.

COMPUTER PERIPHERAL DEVICE INTERFACES

Standards for this ADP system interface have proved to be the most difficult to accomplish, not because of their technical complexity but rather due to competitive pressures and fundamental differences in the architectural structure employed by different ADP system manufacturers. However, several draft proposed American National Standards are presently close to completion.

Large Scale Computer System Peripheral Interfaces

The first set of these computer peripheral device interface standards deal with the large scale ADP system and is based upon the IBM 370 type I/O channel-to-peripheral controller interface; the set consists of three kinds of specifications: (1) a document that prescribes the interface electrical, mechanical, and functional characteristics, (2) an interface power control specification, and (3) a series of device-specific (e.g., tape, disk, etc.) operational specifications.

Figure 1 illustrates the architectural structure for a large scale computer system that contains an I/O channel and shows the point in this structure that is defined as the I/O channel-to-controller interface.

Figure 2 provides a listing of functions presented on the two sides of the I/O channel-to-controller interface and indicates the direction of signalling. In general, a command initiating an action (e.g., Select Out) is issued by the channel, while the response, indicating the action has been completed is issued by the controller.

It is anticipated that an I/O channel-to-peripheral interface standard including operational specifications for both magnetic disk and tape devices will be completed and approved by the American National Standards Institute by late 1977.

Minicomputer System Peripheral Interfaces

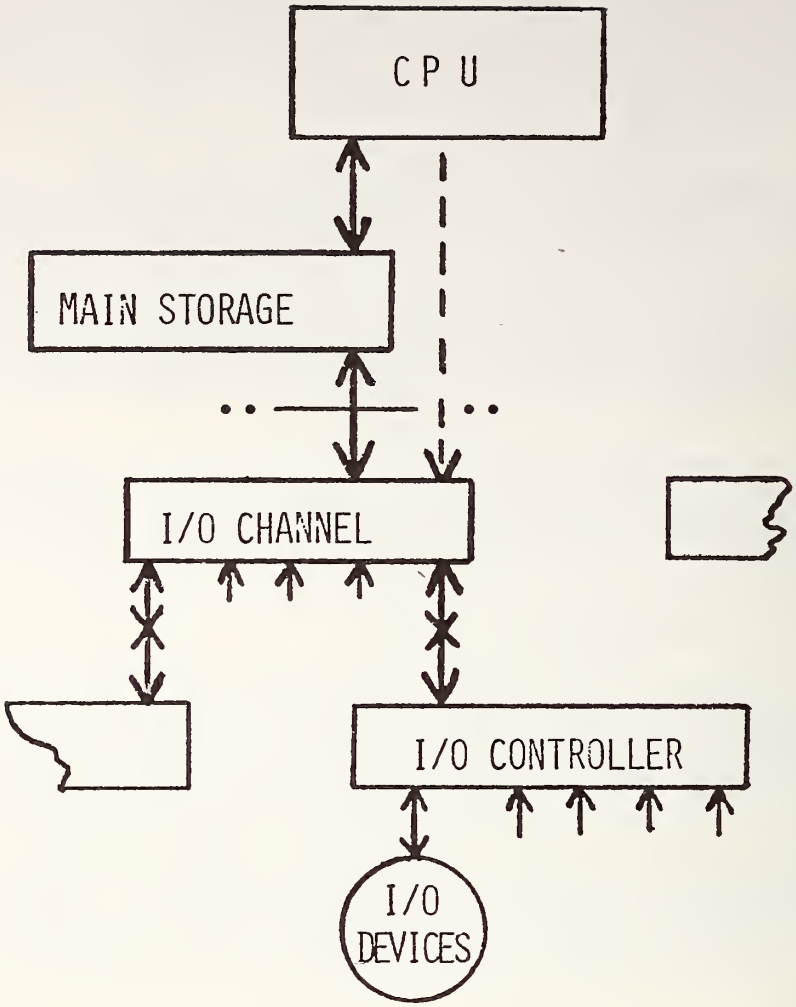
A different kind of device level, but device-specific computer peripheral interface standard is being developed for minicomputer systems. Figure 3 indicates the arrangement of processing logic, control, storage, and I/O components in a typical minicomputer system employing a common bus structure. It also shows the interface point for connecting peripheral devices. In the minicomputer case, a general purpose standard peripheral device interface is being perscribed that contains a total of some 40 functions--all of which would be presented on the CPU side of the interface; devices conforming to this interface, however, will only employ the functions they actually require, elg., a printer cannot perform the function "read media" and thus would not implement this function.

Figure 4 lists the kinds of functions and indicates the signalling directions for this general purpose interface as it would be implemented between a magnetic tape transport and a controller.

It is anticipated that this general purpose device-level minicomputer interface standard will be completed and approved by the American National Standards Institute by the End of 1977. Furthermore, it is planned that in conjunction with the final stages of processing by ANSI these computer peripheral interface standards will also be processed for adoption and implementation as Federal Information Processing Standards.

Recommendation for Computer Peripheral Device Interfaces

The General Services Administration has established a number of Mandatory Requirement Contracts dealing with the procurement of "plug compatible replacement" peripheral devices for the product lines furnished by several of the major manufacturers. These contracts cover magnetic tape and magnetic disk subsystems (including the respective controllers), add-on memory, and input/output punched card facilities. All agencies are obligated to use these GSA Mandatory Requirement Contracts whenever practical to do so. It is recommended, however, that the Air Force



X - I/O Channel to Controller
Interface Points

FIGURE 1: LARGE SCALE COMPUTER SYSTEM ARCHITECTURE

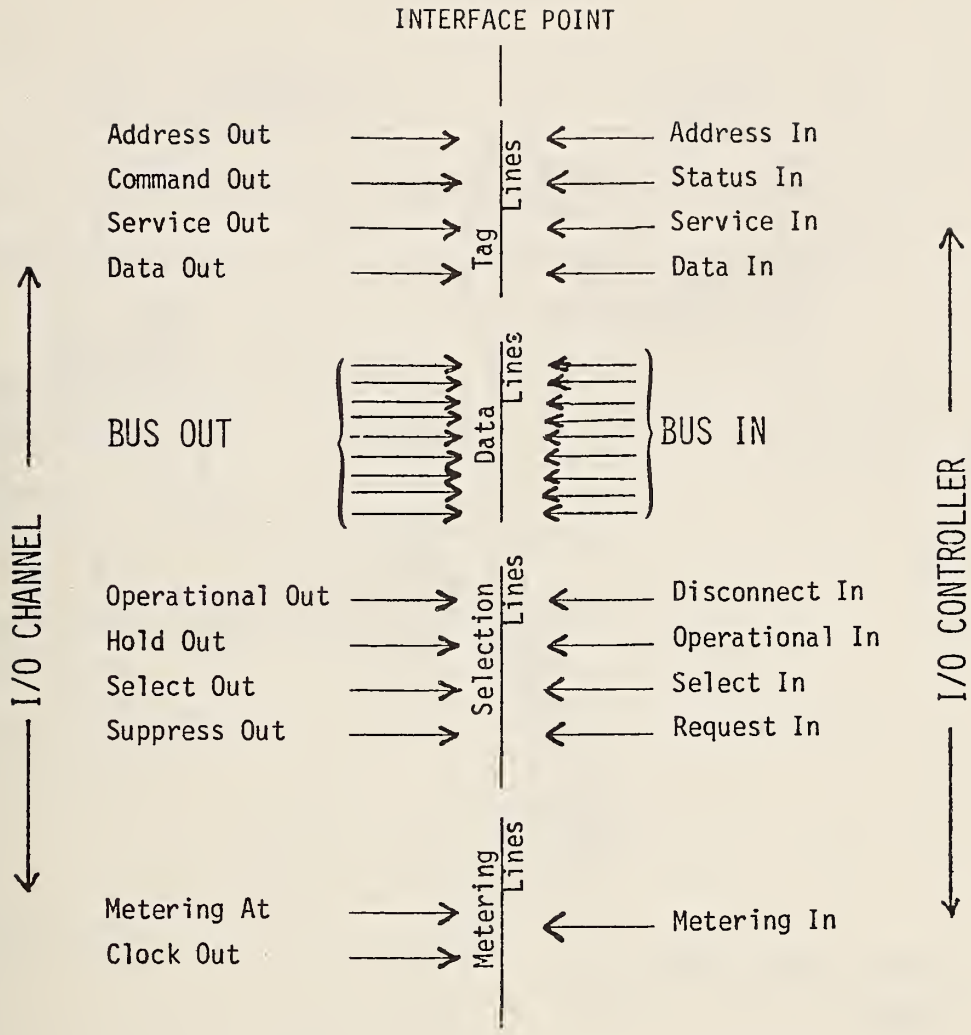
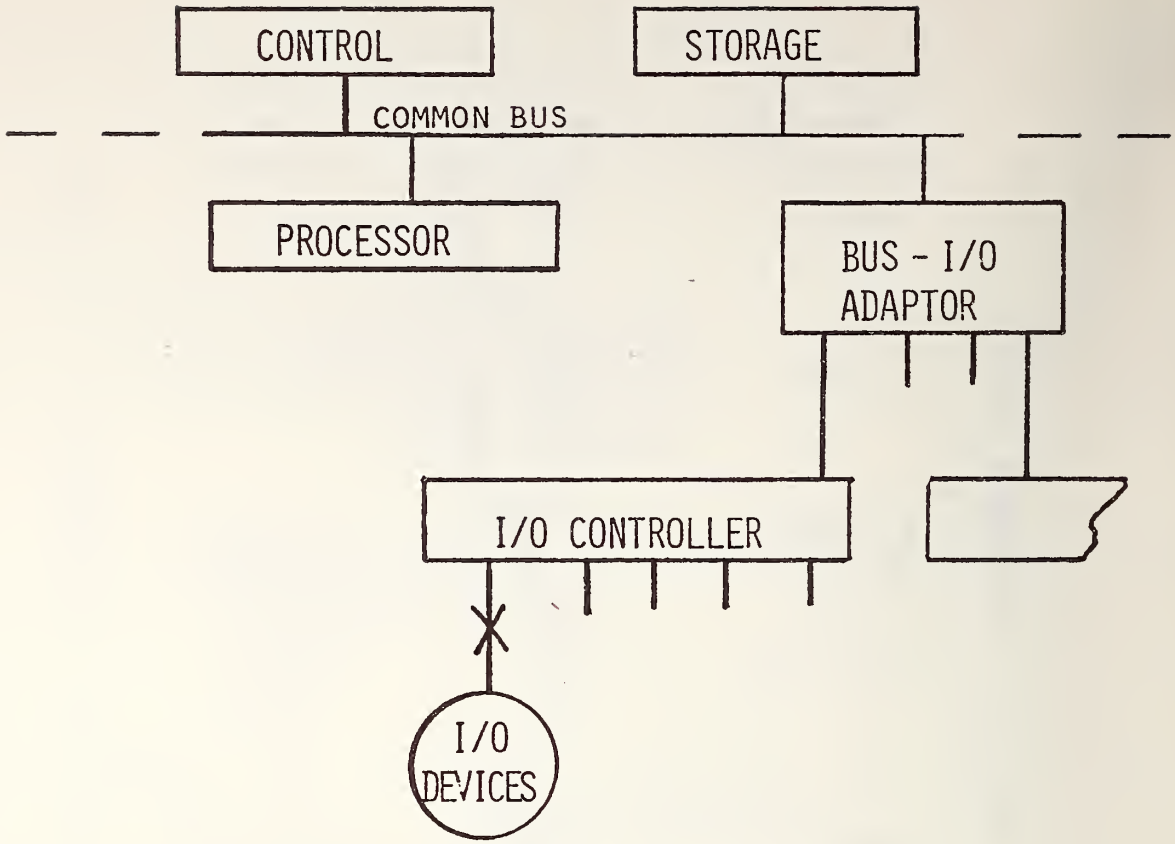


FIGURE 2: I/O CHANNEL TO CONTROLLER INTERFACE



X - Device Level Interface
Point

FIGURE 3: MINICOMPUTER SYSTEM ARCHITECTURE

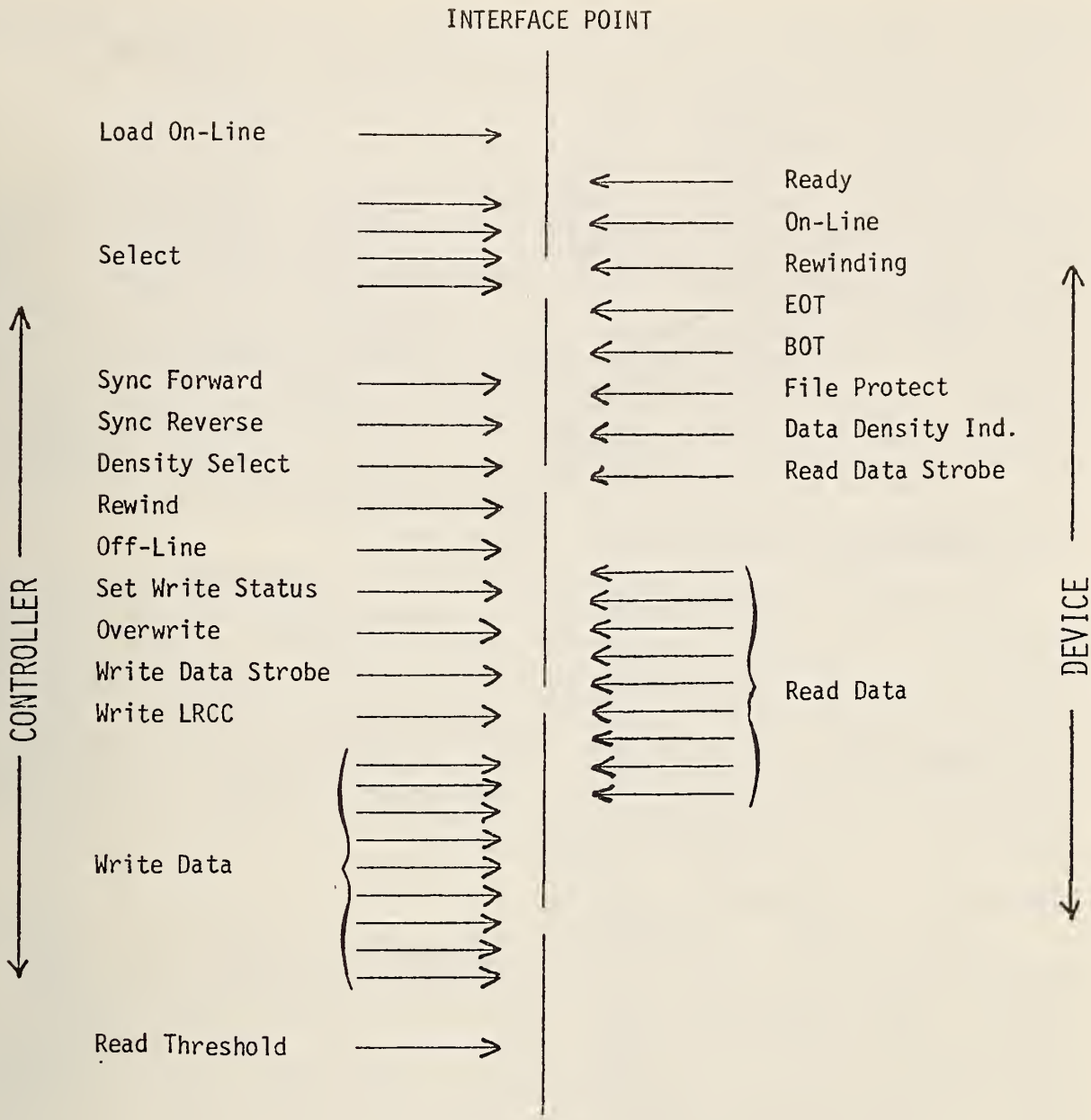


FIGURE 4: I/O CONTROLLER TO MAGNETIC TAPE DEVICE INTERFACE

carefully follow the standards being developed for the computer peripheral device interface and be prepared to implement these in CAM applications as soon as these standards are proposed for Federal adoption.

INSTRUMENTATION INTERFACES

The IEEE has developed and approved (as of July 1974) an industry standard for instrumentation applications entitled "IEEE Standard 488--Digital Interface for Programmable Instrumentation." This standard has also been approved by the American National Standards Institute as ANSI MC 1.1-1975. Although this standard is not limited by its scope, it appears that its principal application is concerned with minicomputers instrumented in close proximity for limited process control functions such as in a laboratory type environment. This standard deals with systems and components that employ byte-serial, bit-parallel data transfer. Figure 5 illustrates the 16 wire bus structure of the IEEE 488 programmable instrumentation interface and indicates some of its characteristics as well as the functional properties (talker, listener, etc.) of some of the components that may be interconnected by it.

Recommendation for Instrumentation Interfaces

It is anticipated that implementation of the IEEE Standard 488 will probably be contrained to minicomputers interconnected in close proximity with digital instruments and devices normally employed in laboratory type experimental situations, e.g., temperature, signals various form sensors, or positional measurements with the processing of these measured data being employed to correct and control their future values. While this interface is not considered to be of general purpose utility for data processing, some of the instruments and devices that are available as "off-the-shelf" items for use in CAM applications are designed to the IEEE Standard 488 interface. For this reason, it is recommended that the Air Force be aware of the existence of IEEE Standard 488.

COMMUNICATIONS INTERFACES

Perhaps the most dynamic areas of computer utilization are currently those concerned with teleprocessing and computer networking that are dependent upon advances in data communication technology. Within the past few years, there have been a number of significant developments in establishing standards for data communications and computer networking. New standards that are in various stages of development include replacements for such widely accepted and implemented standards as RS-232 at the physical interconnection level and binary synchronous (bisync) link control at the link protocol level as well as for higher levels not previously covered, such as for packet switching. These standards are being developed both on a national and international scale by such groups as the American National Standards Institute (ANSI), the International Standards Organization (ISO), and the Consultative Committee on International Telegraph and Telephone (CCITT). Most of these standards eventually will be adopted for mandatory use within the Federal Government by either or both the National Bureau of Standards (NBS) and the National Communications System (NCS). Because most of these standards pertain to the interconnection of computers or data terminal equipment with data communication or telecommunication facilities, they all may be characterized as interface standards dealing with the computer communications interface.

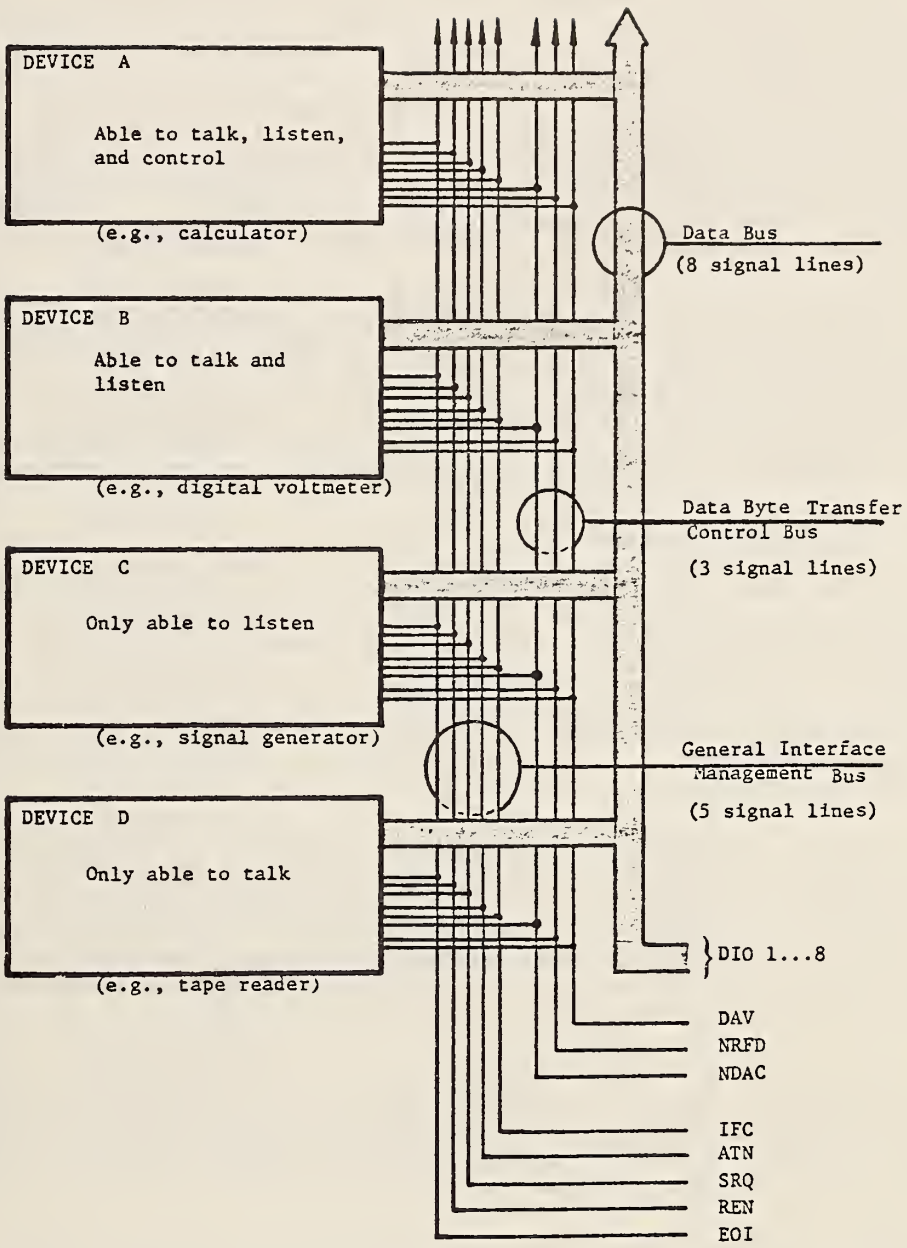


FIGURE 5: THE BUS STRUCTURE FOR THE IEEE STANDARD 488--
DIGITAL INTERFACE FOR PROGRAMMABLE INSTRUMENTATION

Hardware Interconnection Level Interfaces

With the data networks of the future expected to be digital from end to end, standards are being developed to interface terminals to such networks. This includes replacement for RS-232, presently being developed by EIA and known as RS-XYZ, as well as the addition of a signalling scheme to initiate and terminate calls (replacing manual dialing). Internationally, CCITT Recommendation X.21 is being proposed for synchronous terminals and provides a means to initiate calls, exchange call progress signals, transmit data, and finally terminate calls on new public data networks. This CCITT Recommendation is also under consideration for adoption as American National and Federal standards.

Figure 6 and 7 show the respective functional properties of the RS-XYZ and X.21 interfaces. Note that while the RS-XYZ interface provides a separate interchange circuit for each function, the X.21 interface accomplishes essentially the same functional interchanges by combinations of signals presented on the circuit pairs TRANSMIT (data) with CONTROL and RECEIVE (data) with INDICATION, i.e., when CONTROL is "on" the information on the TRANSMIT circuit is interpreted as control--otherwise it is data.

Data Link Level Interfaces

At the data link level, ISO has been working for the past few years to complete the details of a new, bit-oriented High Level Data Link Control Procedure (HDLC). The American National Standard version of this procedure is called the Advanced Data Communications Control Procedure (ADCCP). The concept of a data link to which these control procedural standards apply is defined as an assembly of two or more data terminals and the interconnecting line operated according to a particular method or protocol that permits information to be exchanged.

So far, international agreement has been achieved for both the HDLC frame structure and the elements of procedure (definition of the command and response repertoire). International arguments are still going on regarding the way in which these commands and responses are to be used for various applications involving different terminal and link configurations. Consensus is slow to achieve because of the many different interests that must all be satisfied with the proposed standard.

IBM, because of its ability to act unilaterally in product announcements, has announced a product implementing its own Synchronous Data Link Control (SDLC) procedure which is quite similar to HDLC. Some other vendors such as Burroughs have announced products that they claim will be fully compatible with SDLC, ADCCP, and HDLC.

DEC, on the other hand, has continued to pursue its own link control procedure (DDCMP) which is quite different from any of the proposed standards. As strong vendor participation continues in the final development of both HDLC and ADCCP, it seems likely that both an international and compatible national standard will eventually emerge that will be implemented by most of the major vendors.

Network Level Interfaces

One of the most dramatic standards developments in the last few years has been the adoption of Recommendation X.25 by the CCITT at its quadrennial plenary assembly this past September. Recommendation X.25 is a standard for interfacing host computers to public packet switching networks. It includes both X.21 and HDLC in the appropriate portion of the standard, and adds a set of packet formats and commands and responses for setting up "virtual calls" and transferring data through the network.

CIRCUIT MNEMONIC	CIRCUIT NAME	CIRCUIT DIRECTION	CIRCUIT TYPE	
SG	SIGNAL GROUND	–	COMMON	
SC	SEND COMMON	TO DCE		
RC	RECEIVE COMMON	FROM DCE		
IS	TERMINAL IN SERVICE	TO DCE	CONTROL	
IC	INCOMING CALL	FROM DCE		
TR	TERMINAL READY	TO DCE		
DM	DATA MODE	FROM DCE		
SD	SEND DATA	TO DCE	DATA	PRIMARY CHANNEL
RD	RECEIVE DATA	FROM DCE		
TT	TERMINAL TIMING	TO DCE	TIMING	
ST	SEND TIMING	FROM DCE		
RT	RECEIVE TIMING	FROM DCE		
RS	REQUEST TO SEND	TO DCE	CONTROL	
CS	CLEAR TO SEND	FROM DCE		
RR	RECEIVER READY	FROM DCE		
SQ	SIGNAL QUALITY	FROM DCE		
NS	NEW SIGNAL	TO DCE		
SR	SIGNALING RATE	TO DCE		
SSD	SECONDARY SEND DATA	TO DCE	DATA	SECONDARY CHANNEL
SRD	SECONDARY RECEIVE DATA	FROM DCE		
SRS	SECONDARY REQUEST TO SEND	TO DCE	CONTROL	
SCS	SECONDARY CLEAR TO SEND	FROM DCE		
SRR	SECONDARY RECEIVER READY	FROM DCE		
LL	LOCAL LOOPBACK	TO DCE	CONTROL	
RL	REMOTE LOOPBACK	TO DCE		
TM	TEST MODE	FROM DCE		
SS	SELECT STANDBY	TO DCE	CONTROL	
SB	STANDBY INDICATOR	FROM DCE		

FIGURE 6: INTERCHANGE CIRCUITS DEFINED FOR RS-XYZ
THE DTE/DCE INTERFACE FOR ANALOG NETWORKS

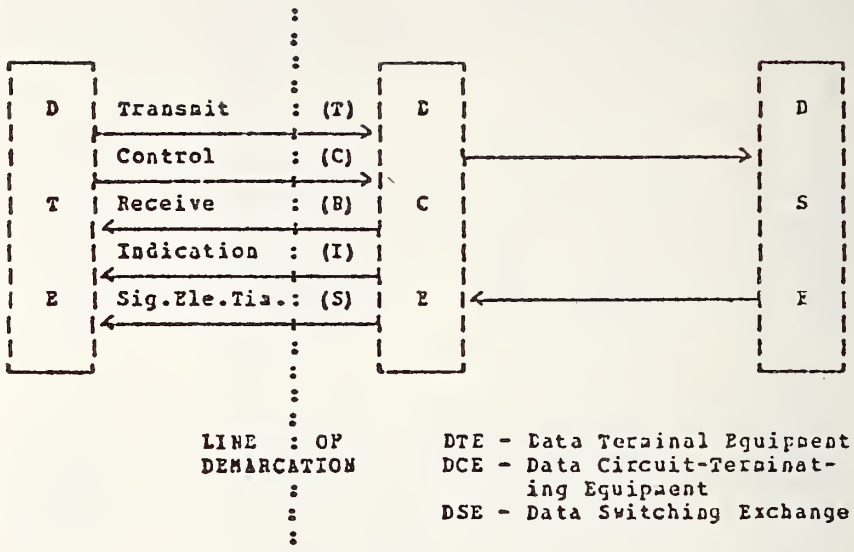


FIGURE 7: INTERCHANGE CIRCUITS DEFINED FOR CCITT RECOMMENDATION X.21 THE DTE/DCE INTERFACE FOR DIGITAL NETWORKS

Figure 8 shows the enveloping format prescribed by X.25 for the interchange of information between a host computer and a packet switched network. A packet consists of data to be transferred between two users. This data is preceded by a packet header that identifies the sender and intended recipient; the network uses information contained in the header for routing, billing, and network control purposes. The packet is then enveloped by an HDLC frame for transmission between the host computer and the network. The HDLC frame provides for link level control and consists of bracketing opening and closing flag octets, a link address octet and a control octet identifying the type of command or response frame; the frame is ended with the two octet Frame Check Sequence provided for error detection just prior to the closing flag octet.

Agreement on such a worldwide standard seemed quite remote only a few years ago, and it was not until the major packet switching carriers around the world got together privately that a consensus emerged. The standard has been criticized by some as lacking in certain features--notably the standard does not presently provide for the "datagram" type of service--but this particular deficiency is already being addressed by proposals to add to the standard.

The key point to note about X.25 is that a workable solution has been adopted which averts the situation of multiple incompatible interfaces being implemented by carriers in each country. Thus, it will be possible for computer manufacturers and software houses to build and support only one interface for packet switching.

Recommendations for Computer Communications Interfaces

The data communications area is a very dynamic one at present, and standards will continue to evolve to keep pace with the state of the art. Significant developments to look for over the next few years are the completion and large scale implementation of work already begun, such as HDLC, additions and modification to recently adopted standards, such as X.25, and the initiation of new work in areas not currently addressed, such as end-to-end protocols between host computers.

Designers of networks within the Department of Defense, such as AUTODIN-II and SATIN-IV are generally cognizant of these standards developments and insofar as practical most of these new standards are being implemented as the network design specifications are finalized.

For this reason, it is recommended that the Air Force advise ICAM contractors to confer with commercial data communication carriers concerning alternative network design characteristics and particularly with regard to specific user-to-network interfacing requirements rather than unilaterally prescribing communication interface standards that might subsequently prove incompatible with existing or planned networks.

SUMMARY OF INTERFACE STANDARDS

Figure 9 provides a system level overview of the typical locations of the several standard interfaces that have been described for a system that consists of one large scale computer connected to two remotely located minicomputers via a packet switched public data network. While the interfaces described are not the only interface points in a processing system such as this standardization of these particular interfaces provides the consumer of ADP products and services with a large degree of freedom in the acquisition and interconnection of components furnished by competitive sources. It should be noted, however, that although the various interface standards that have been described do make possible the physical interconnection of independently supplied components as well as

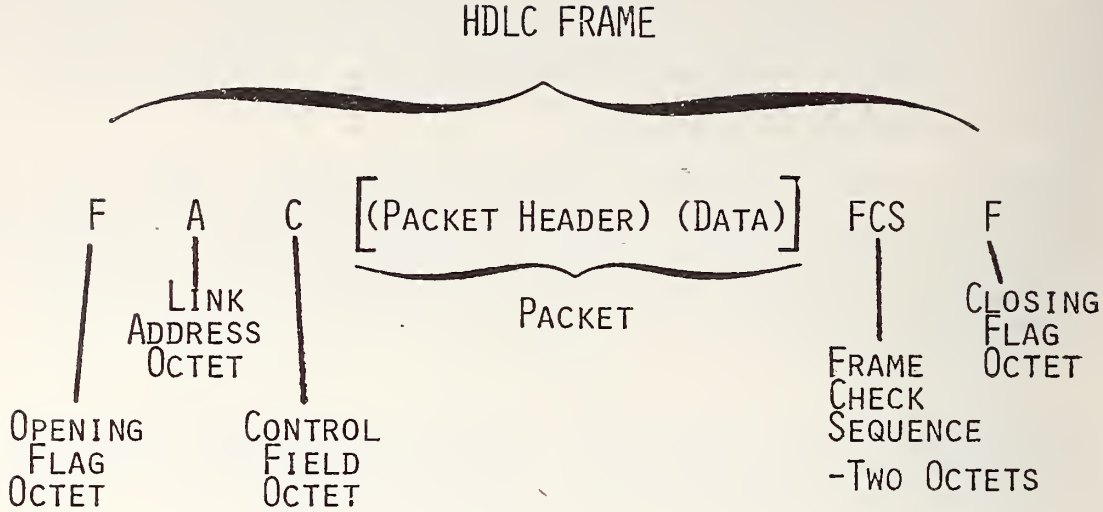
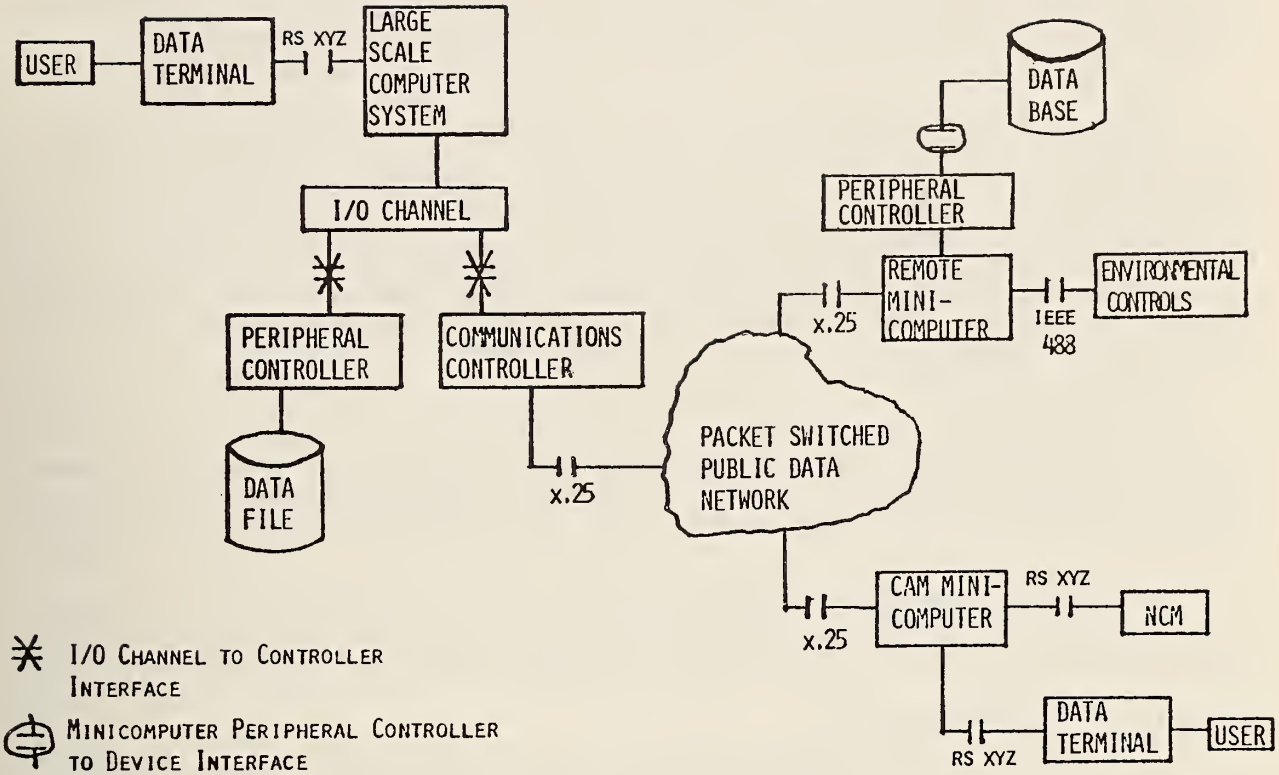


FIGURE 8: THE TRANSMISSION FORMAT PRESCRIBED BY CCITT RECOMMENDATION X.25



NOTE: X.25 PRESCRIBES THREE INTERFACE LEVELS: (1) THE PHYSICAL CIRCUIT LEVEL EMPLOYING X.21, (2) THE LINK CONTROL LEVEL EMPLOYING THE HDLC PROTOCOL, AND (3) THE PACKET LEVEL PROTOCOL INCLUDING SOME 14 PACKET FORMATS.

FIGURE 9: SYSTEM LEVEL PERSPECTIVE OF INTERFACES DESCRIBED

enable the interchange of data among these components it must be emphasized that these standards are not sufficient to ensure that meaningful end-to-end information interchange can occur. End-to-end communication between users in a system such as this also requires that both ends employ a common protocol involving a standard language that is represented with an agreed upon alphabet with characters encoded in a standard manner.

While a number of different multi-computer networking systems have been designed and successfully implemented that are incompatible among themeselves with regard to user protocols, languages, and codes, development of standards for many of these higher level problems has not yet been satisfactorily addressed. Partly, this is because some of these higher level problems are not yet sufficiently well defined that a standard solution can be prescribed--even though the need for standardization is generally recognized; partly, it is because in other cases a number of alternative competing solutions have been proposed, none of which appear optimal.

An an interim alternative to standardization for some of these higher level problems, NBS has designed and implemented a Network Access Machine (see NBS Technical Note 917) that employs a minicomputer to translate from a common user protocol to that required for accessing a variety of services provided by different remote host computer systems.

It is anticipated that these higher level areas of standardization will receive increasingly urgent attention in the near future and it is recommended that the Air Force monitor these activities closely.

SUMMARY OF RECOMMENDATIONS

a) It is recommended that the Air Force carefully follow the standards being developed for the computer peripheral device interface and be prepared to implement these in CAM applications as soon as these standards are proposed for Federal adoption.

b) It is recommended that the Air Force be aware of the existence of the IEEE Standard 488 that prescribes a Digital Interface for Programmable Instrumentation.

c) It is recommended that the Air Force advise ICAM contractors to confer with commercial data communication carriers concerning alternative network design characteristics and particularly with regard to specific user-to-network interfacing requirements rather than unilaterally prescribing communication interface standards that might subsequently prove incompatible with existing or planned networks.

d) It is recommended that the Air Force closely monitor standardization activities in the area of establishing common user, network access, and other higher level standards that will help ensure end-to-end communications in a heterogeneous computer networking environment.

REFERENCES

(1) ANSI X3S34/589 (Fifth Draft), 4/9/76, Advanced Data Communications Control Procedures, American National Standards Institute.

(2) Donnan, R. A., and J. Ray Kersey, "Synchronous Data Link Control: A Perspective", IBM Systems J., 13, 2, 1974.

(3) IBM Corp., "Binary Synchronous Communications", Order No. GA27-3004, IBM Corp., White Plains, N.Y. 10604.

(4) ANSI X3.28-1971, "Procedures for the use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links", American National Standards Institute, Inc., New York, N.Y., 10018

(5) Metcalf, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for local computer networks", CACM, 19, 9, 7/76, pp. 395-403.

(6) Farber, D. J., & K. C. Larson, "The System Architecture of the Distributed Computer System - The Communications System", Presented at the Symposium on Computer Networks, Polytechnic Institute of Brooklyn, 4/72.

(7) C.C.I.T.T., "Recommendation X.25 - Interface between Data Terminal Equipment and Data Circuit-Termination Equipment for Terminals Operating in the Packet Mode on Public Data Networks". See also ANSI documents X3S37-76-]4 and X3S33-76-6.

(8) Pouzin, L., "Virtual Circuits vs. Datagrams - Technical and Political Problems", Proc. NCC, 1976 (V. 45), pp. 483-495.

(9) ANSI X3S37-75-54/4 (Fourth Draft - "ANSI X.21") Proposed American National Standard--"General Purpose Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment for Synchronous Operation on Public Data Networks.

(10) Electronic Industries Association Draft Standard--"Functional and Mechanical Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange"--(Temporarily Labeled RS-XYZ), Twelfth Draft--May 7, 1976, Amended--July 30, 1096, Prepared by EIA Subcommittee RS 30.2.

STANDARDS DATA SHEETS

The following Data Sheets summarize those standards which apply to the Computer and Communications Interface.

1. Designation: EIA RS-232-C, August 1969

2. Title: Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange, August 1969

3. Maintenance Authority: Electronic Industries Association, Subcommittee TR-30.2

4. Scope: Hardware Standard. "This standard is applicable to the interconnection of data terminal equipment (DTE) and data communication equipment (DCE) employing serial binary data interchange." It defines: (1) electrical signal characteristics, (2) interface mechanical characteristics, (3) functional description of interchange circuits, (4) standard interfaces for selected communication system configurations.

5. Relationship to Other Standards:

   EIA RS-334 (ANSI X3.24-1968) Signal quality for EIA 232-C interface
   EIA RS-422 and EIA RS-423, April 1975 (revised electrical signal characteristics)
   EIA SP-1194, October 1975 (revised functional description)

6. Competitive Standards: CCITT V.24 (functional) and V.28 or V.31 for electrical characteristics. CCITT X.21 corresponding interface for public data (in contrast to public telephone) networks.

7. Standardization Status: RS-232, May 1960; RS-232-A, October 1963; RS-232-B, October 1965. RS-232-C is expected to be gradually (ten years) replaced by EIA SP-1194A (see writeup immediately below).

8. Implementation Status: Commerically, RS-232-C has enjoyed universal acceptance as the data terminal-to-modem de facto interface. Although MIL STD 188C prescribes 232-C functions, it employs different (lower voltage and lower impedance) electrical characteristics, primarily for security and privacy purposes.

9. Known Manufacturing Uses: RS-232-C is primarily a communcations (serial) interface specification.

10. Known Sources of Information: Mr. A. M. Wilson, Electronic Industries Association, 2001 Eye Street, N.W., Washington, D.C. 20006, (202) 659-2200.

11. Probable Sources of Information: Mr. George E. Clark, National Bureau of Standards, Building 225, Room B210, Washington, D.C. 20234, (301) 921-3723.

12. Bibliography: EIA RS-232-C, August 1969

13. Comments: Equipment conforming to RS-232-C will gradually be replaced with that conforming to RS-422 and 423 (employing integrated circuit components) that will also operate over much greater distances (up to 400 ft.) and at much higher speeds (up to 10 mega bits/sec.). (Note that 232-C is constrained to 20 kilobits/sec and 50 ft.)

1. Designation: EIA SP-1194A/Proposed Federal Standard 1031/Proposed FIPS PUB

2. Title: Functional and Mechanical Interface Between Data Terminal Equipment and Data Communication Equipment

3. Maintenance Authority: Electronic Industries Association Subcommittee TR-30.2.[1]

4. Scope: SP-1194A, together with EIA RS-422 and RS-423, is intended to supersede EIA RS-232-C.

5. Relationship to Other Standards: SP-1194 is presently being revised and may result in two or more standards specifying different interface functional and mechanical characteristics.

6. Competitive Standards:

7. Standardization Status: EIA SP-1194A is presently (September 1976) under EIA ballot. This Standard, together with EIA Standard RS-423, is intended to gradually replace EIA Standard RS-232-C as the specification for the nonengineered interface between data terminal equipment (DTE) and data communication equipment (DCE) employing serial binary data interchange at data signaling rates up to 60,000 bits per second. With a few additional provisions for interoperability, equipment conforming to this standard can interoperate with equipment designed to RS-232-C. This standard is intended primarily for data applications using analog telecommunications networks.

8. Implementation Status: None

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. A. M. Wilson, Electronic Industries Association, (202) 659-2200

11. Probable Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723

12. Bibliography: EIA SP-1194A

13. Comments: A notice of an earlier version (EIA SP-1194) of this standard as a proposed Federal Standard (1031) and a proposed FIPS PUB appeared in the Federal Register on December 5, 1975, page 56938.

---

[1] As a Federal Standard, it would be maintained by the National Communications System (NCS-TS), Washington, D.C. 20305.

1. Designation: ANSI X3.24-1968 (EIA RS-334, March 1967)

2. Title: Signal Quality at Interface Between Data Processing Terminal Equipment and Synchronous Data Communication Equipment for Serial Data Transmission

3. Maintenance Authority: Electronic Industries Association Subcommittee TR-30.1

4. Scope: "This standard is applicable to the exchange of serial binary data signals and timing signals across the interface between data processing terminal equipment and synchronous data communication equipment, as defined in EIA Standard RS-232-C. The data communication equipment is considered to be synchronous if the timing signal circuits are at the transmitting terminal or the receiving terminal, or both . . . This standard does not describe any requirements for error performance, either for a complete system or any system components."

5. Relationship to Other Standards: EIA RS-232-C (the interface)

6. Competitive Standards: None

7. Standardization Status: First approved by EIA in March 1967. Approved as an ANSI Standard on September 27, 1968. Revision of this standard according to the newly approved electrical characteristics is awaiting final EIA actions on recent (July 1976) revisions of EIA Standards RS-422 and 423.

8. Implementation Status: Most equipment conforming to RS-232-C exceeds the provisons of X3.24.

9. Known Manufacturing Uses: Used in conjunction with RS-232-C for specifying the DTE/DCE communications interface.

10. Known Sources of Information: Mr. A. M. Wilson, Electronic Industries Association, 2001 Eye Street, N.W., Washington, D.C. 20006, (202) 659-2200

11. Probable Sources of Information: Mr. George E. Clark, National Bureau of Standards, Building, 225, Room B210, Washington, D.C. 20234, (301) 921-3723

12. Bibliography: ANSI X3.24-1968 (EIA RS-334, March 1967)

13. Comments: Conformance to X3.24 assures that the signal amplitude and timing relationships will be compatible for equipment furnished by different suppliers -- and providing that the RS-232-C functions are consistently implemented, this conformance insures that equipment will interoperate.

1. Designation: EIA RS-408

2. Title: Interface Between Numerical Control Equipment and Data Terminal Equipment Employing Parallel Binary Data Interchange

3. Maintenance Authority: EIA EI-31

4. Scope: Hardware Standard. This standard applies to the interconnection of data terminal equipment and numerical control equipment at the tape reader interface. It provides electrical signal characteristics, interface mechanical characteristics, and a functional description of the interface.

5. Relationship to Other Standards: This standard is for parallel-by-bit, serial-by-byte data, such as that generated by a perforated tape reader.

6. Competitive Standards: IEEE Standard 488-1975

7. Standardization Status: Approved by EIA in March 1973

8. Implementation Status: Widely implemented in numerical control equipment.

9. Known Manufacturing Uses: Used in machines employing numerical control.

10. Known Sources of Information: Mr. A. M. Wilson, EIA, (202) 659-2200; Dr. John Evans, NBS, (301) 921-2381.

11. Probable Sources of Information: NMTBA

12. Bibliography: EIA RS-408, March 1973

13. Comments: The data terminal equipment (DTE) typically includes a serial-to-parallel converter. This standard is employed on the parallel-by-bit side of the DTE. Other standards, such as EIA RS-232-C, apply at the serial-by-bit side of the DTE.

1. Designation: IEEE Standard 488-1975

2. Title: IEEE Standard Digital Interface for Programmable Instrumentation

3. Maintenance Authority: IEEE Instrumentation and Measurement Group

4. Scope: Hardware Standard. This standard applies to interface systems used to interconnect both programmable and non-programmable (digital) electronic measuring apparatus with other apparatus and accessories necessary to assemble instrumentation systems. It is a parallel-by-bit, serial-by-byte standard.

5. Relationship to Other Standards: The character coding is based upon ISO 646-1973, similar to ASCII, FIPS PUB 1, ANSI X3.4-1968.

6. Competitive Standards: EIA RS-408, IEEE Standard 583-1975 (CAMAC)

7. Standardization Status: Approved by the IEEE Standards Board on December 19, 1974. Development of this standard was coordinated with IEC/TC66/WG3. It may become an IEC standard.

8. Implementation Status: Implemented in electronic instruments, such as those made by the Hewlett-Packard Co.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. Robert A. Soderman, General Radio Co., (617) 396-4400 x608; Mr. Donald C. Loughry, Hewlett-Packard Co., (408) 735-1550; Mr. Robert G. Fulks, Omnicomp, 71 N. 12th Place, Phoenix, (602) 997-5456.

11. Probable Sources of Information: IEEE

12. Bibliography: IEEE Standard 488-1975

13. Comments: Up to 15 devices may be interconnected on one "party-line" configuration. Cable length is up to 20 meters. Maximum data rate on any signal line is one megabit per second. This standard is optimized for devices in close proximity (up to 20 meters).

1. Designation: IEEE Standard 583-1975

2. Title: IEEE Standard Modular Instrumentation and Digital Interface Systems (CAMAC)[1]

3. Maintenance Authority: IEEE Nuclear Instruments and Detectors Committee

4. Scope: Hardware Standard. "This standard is intended to serve as a basis for a range of modular instrumentation capable of interfacing transducers and other devices to digital controllers for data and control. The standard fully specifies a data bus by means of which instruments and other functional modules can communicate with each other, with peripherals, with computers, and with other external controllers. Data may be transferred either bit-serial or byte-serial."

5. Relationship to Other Standards: Identical in many respects to IEC 482 and IEC 516.

6. Competitive Standards: EIA RS-408, IEEE Standard 488, EIA RS-232-C

7. Standardization Status: Approved by the IEEE Standards Board on February 27, 1975.

8. Implementation Status: Increasingly implemented in laboratory digital instrumentation equipment, especially that related to nuclear physics and testing.

9. Known Manufacturing Uses: Aluminum Furnace Control (ALCOA), Steel Process Control (Inland Steel Co.), Diesel Locomotive Testing (GM), Large Power Semiconductor Testing (GE), Telescope Control and Data Gathering (Kitt Peak)

10. Known Sources of Information: Mr. Dale W. Zobrist, Eldec Corporation, (206) 743-1313; Mr. Louis Costrell, NBS, (301) 921-2518; Mr. Lowell A. Klaisner, Kinetic Systems Corporation, (815) 838-0005

11. Probable Sources of Information: IEEE, ERDA, Stanford Linear Accelerator Center, Lawrence Radiation Lab, Berkeley, California.

12. Bibliography: IEEE Standard 583-1975; "CAMAC, A Modular Standard," IEEE Spectrum, April 1976, pp. 50-55.

13. Comments: This standard was developed by the ESONE Committee of European Laboratories and the NIM Committee of ERDA. Data may be transferred byte-serial for high speeds and bit-serial for long distances.

---

[1] Computer Automated Measurement and Control

1. Designation: FIPS PUB 22-1 (1976)/ANSI X3.1-1976

2. Title: Synchronous Signaling Rates Between Data Terminal and Data Communication Equipment

3. Maintenance Authority: ANSI X3S36

4. Scope: This standard provides a group of specific signaling rates for synchronous serial or parallel binary data transmission. These rates exist on the received data and transmitted data circuits of the interface between data terminal equipment and data communications equipment which operate over nominal 4kHz voice bandwidth channels.

5. Relationship to Other Standards: FIPS PUB 37/ANSI X3.36-1975 (wide band synchronous signaling rates); EIA RS-334 is referenced by ANSI X3.1 for tolerances on the prescribed rates.

6. Competitive Standards: None

7. Standardization Status: First approved by ANSI in 1962 and revised slightly in 1966, 1969, and 1976. The most recent revision (1976) eliminated the "interim-speed of 2000 bits/second."

8. Implementation Status: FIPS PUB 22-1 differs from X3.1 only in that it specifies the tolerance as follows: "The deviation from any specified rate shall not exceed 0.01 percent."

9. Known Manufacturing Uses: Applicable to data terminal and data processing equipment employed with synchronous data communication designed to operate on binary encoded information over voice grade lines.

10. Known Sources of Information: Mr. William F. Hanrahan, Secretary, ANSI X3, (202) 466-2288; H. J. Crowley, Chairman X3S36, (315) 330-2355

11. Probable Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723

12. Bibliography: FIPS PUB 22-1 (1976)/ANSI X3.1-1976

13. Comments: None

1. Designation: FIPS PUB 16-1971/ANSI X3.15-1966

2. Title: Bit Sequencing of ASCII in Serial-by-Bit Data Transmission

3. Maintenance Authority: NBS/ANSI X3S33

4. Scope: This standard specifies the bit sequencing of ASCII (ANSI X3.4-1968) for serial-by-bit, serial-by-character data transmission. It applies at the interface between data processing terminal equipment and data communications equipment.

5. Relationship to Other Standards: This is an implementation standard for CCITT Recommendation V.4-1972 and for ASCII (FIPS PUB 1, ANSI X3.4-1968) and the character structure standards for serial-by-bit data (FIPS PUB 17, ANSI X3.16-1966). EIA RS-232-C uses this standard.

6. Competitive Standards: All bit-oriented, code-independent data transmission standards, such as HDLC, SDLC, ADCCP, BDLC, etc; parallel-by-bit standards, such as FIPS PUB 18, ANSI X3.25-1968.

7. Standardization Status: Approved as an ANSI standard on August 19, 1966. FIPS PUB 16 adopted RS-232-C and ASCII (FIPS PUB 1, ANSI X3.4-1968).

8. Implementation Status: Widely implemented in terminal equipment conforming to EIA standard RS-232-C and ASCII (FIPS PUB 1, ANSI X3.4-1968).

9. Known Manufacturing Uses: Virtually all ASCII data transmitted in serial-by-bit, serial-by-character form, conforms to the conventions of this standard.

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. William F. Hanrahan, Secretary of ANSI X3, (202) 466-2288.

11. Probable Sources of Information: Teletype Corporation

12. Bibliography: FIPS PUB 16-1971, ANSI X3.15-1966. CCITT "Green Book," Vol. VIII, Recommendation V.4 on pp. 61-62, 1973.

13. Comments: This standard specifies that the ASCII bits for each character be transmitted low-order bit (b1) first. Character-oriented data, such as decimal digits, are usually transmitted high-order character first, and are stored in computer memories with the high-order characters at the high order end of words or blocks. Hence, each character transmitted according to this standard may be subjected to bit inversion for transmission and further bit inversion for re-assembly of a computer-oriented character stream or data structure. For this reason, IBM and others opposed this standard, which was a highly controversial proposal until it was approved in 1966.

1. Designation: FIPS PUB 17-1971/ANSI X3.16-1966

2. Title: Character Structure and Character Parity Sense for Serial-by-Bit Data Communication in ASCII (FIPS PUB 1/ANSI X3.4-1968)

3. Maintenance Authority: NBS/ANSI X3S33

4. Scope: Hardware Standard. This standard specifies the character structure and sense of character parity for serial-by-bit, serial-by-character synchronous and asynchronous data communication in ASCII (FIPS PUB 1, ANSI X3.4-1968). This standard applies to general information interchange at the interface between data processing terminal equipment and the data communication equipment.

5. Relationship to Other Standards: This standard is an implementation of the 7-bit code of ASCII (FIPS PUB 1/ANSI X3.4-1968). It is used at interfaces such as EIA RS-232-C. The companion standard FIPS PUB 18/ANSI X3.25-1968 is for character structures using parallel-by-bit data communication. Subsets, such as EIA RS-358, can use the structure of this standard.

6. Competitive Standards: Proprietary structures for communicating non-ASCII codes, such as 6-bit Teletypesetter or 8-bit EBCDIC

7. Standardization Status: The ANSI standard X3.16 was approved on August 19, 1966; FIPS PUB 17, adopting in its entirety that ANSI standard, was approved on October 1, 1971.

8. Implementation Status: Widely implemented in communication systems and ADP terminal devices.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. George E. Clark, NBS, (301) 921-3723.

11. Probable Sources of Information: Teletype Corporation

12. Bibliography: FIPS PUB 17-1971/ANSI X3.16-1966

13. Comments: This standard specifies odd parity for synchronous data communication and even parity for asynchronous data communication. It does not specify the bit sequence, which is given in FIPS PUB 16/ANSI X3.15-1966.

1. Designation: FIPS PUB 18-1971/ANSI X3.25-1968

2. Title: Character Structure and Character Parity Sense for Parallel-by-Bit Data Communication in ASCII (FIPS PUB 1/ANSI X3.4-1968)

3. Maintenance Authority: NBS/ANSI X3S33

4. Scope: Hardware Standard. This standard specifies the character structure and sense of character parity for parallel-by-bit, serial-by-character, data communication in ASCII (FIPS PUB 1/ANSI X3.4-1968). This standard applies to general information interchange at the interface between data processing terminal equipment and data communication equipment.

5. Relationship to Other Standards: This standard is an implementation of the 7-bit code for ASCII (FIPS PUB 1/ANSI X3.4-1968). It is used at parallel-by-bit interfaces, such as EIA RS-408. The companion standard FIPS PUB 17/ANSI X3.16-1966 is for character structures using serial-by-bit data communication. Subsets, such as EIA RS-358 can use the structure of this standard.

6. Competitive Standards: Proprietary incompatible structures for communicating non-ASCII codes, such as 8-bit EBCDIC.

7. Standardization Status: The ANSI standard X3.25 was approved on September 27, 1968; FIPS PUB 18, adopting in its entirety that ANSI standard, was approved on October 1, 1971.

8. Implementation Status: Implemented in most parallel-by-bit data communication devices.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723; Mr. John L. Little, NBS, (301) 921-3723.

11. Probable Sources of Information:

12. Bibliography: FIPS PUB 18-1971, ANSI X3.25-1968

13. Comments: This standard specifies an 8-bit character structure including the 7 bits of ASCII and an odd parity bit where the character timing is not separately signaled. Where the character timing is on a separate timing channel, the parity sense is even.

1. Designation: FIPS PUB 37 (1975)/FED-STD-1001/ANSI X3.36-1975

2. Title: Synchronous High Speed Data Signaling Rates Between Data Terminal Equipment and Data Communication Equipment

3. Maintenance Authority: ANSI X3S36

4. Scope: "This standard provides a group of specific signaling rates for synchronous high speed serial data transfer. These rates exist on the received data and the transmitted data circuits of the interface between data terminal equipment and data communication equipment that operate over high speed channels."

5. Relationship to Other Standards: FIPS PUB 22-1 (1976)/ANSI X3.1-1976

6. Competitive Standards: None

7. Standardization Status: The Federal standard (FED-STD-1001) adopts the ANSI standard (X3.36) with two exceptions, as follows: "a. The note alluding to certain unspecified coding restrictions on the data stream of users operating at 1544 kbits/sec is not applicable," and "b. A signaling rate of 64 kbit/sec may also be utilized by Federal agencies having requirements to interface directly with point-to-point transmission facilities of foreign communication carriers."

8. Implementation Status:

9. Known Manufacturing Uses: None

10. Known Sources of Information: Mr. William F. Hanrahan, Secretary, ANSI X3, (202) 466-2288; H. J. Crowley, Chairman X3S36, (315) 330-2355

11. Probable Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723

12. Bibliography: FIPS PUB 37 (1975)/FED-STD-1001/ANSI X3.36-1975

13. Comments: ATT has only recently applied for a tariff (#269) proposing to offer the rates prescribed by this standard as part of the Dataphone Switched Digital Service (DSDS). This tariff is based on 56 kbit/sec subscriber services. This and other speeds prescribed by X3.36 are not presently in wide usage.

1. Designation: FED-STD-1020/EIA RS-422 (1975)

2. Title: Electrical Characteristics of Balanced Voltage Digital Interface Circuits

3. Maintenance Authority: Electronic Industries Association Committee TR-30.1

4. Scope: This standard specifies the electrical characteristics of the balanced voltage digital interface circuit, normally implemented in integrated circuit technology, that may be employed for the interchange of serial binary signals between data terminal and data communication equipment.

5. Relationship to Other Standards: FED-STD-1030/EIA RS-423 (Unbalanced Voltage Digital Interface Circuits); EIA RS-232-C (only the electrical characteristics).

6. Competitive Standards: None

7. Standardization Status: RS-422 (and also RS-423) may be employed as an evolutiontary replacement for the electrical characteristics of RS-232-C.

8. Implementation Status: Not widely implemented at present.

9. Known Manufacturing Uses: Although primarily designed for communication interface applications, the integrated circuit components implementing RS-422 can be employed in many other data interchange environments.

10. Known Sources of Information: Mr. A. M. Wilson, Electronic Industries Association, (202) 659-2200

11. Probable Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723

12. Bibliography: FED-STD-1020/EIA RS-422 (1975)

13. Comments: None

1. Designation: FED-STD-1030/EIA RS-423 (1975)

2. Title: Electrical Characteristics of Unbalanced Voltage Digital Interface Circuits

3. Maintenance Authority: Electronic Industries Association Committee TR-30.1

4. Scope: This standard specifies the electrical characteristics of the unbalanced voltage digital interface circuit, normally implemented in integrated circuit technology, that may be employed for the interchange of serial binary signals between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE).

5. Relationship to Other Standards: FED-STD-1020/EIA RS-422 (Balanced Voltage Digital Interface Circuits); EIA RS-232-C (only the electrical characteristics)

6. Competitive Standards: None

7. Standardization Status: RS-423 (and also RS-422) may be employed as an evolutionary replacement for the electrical characteristics of RS-232-C.

8. Implementation Status: Not widely implemented at present.

9. Known Manufacturing Uses: Although primarily designed for communication interface applications, the integrated circuit components implementing RS-423 can be employed in many other data interchange environments.

10. Known Sources of Information: Mr. A. M. Wilson, Electronic Industries Association, (202) 659-2200

11. Probable Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723

12. Bibliography: FED-STD-1030/EIA RS-423 (1975)

13. Comments: None

1. Designation: C.C.I.T.T. Recommendation V.28

2. Title: Electrical Characteristics for Unbalanced Double-Current Interchange Circuits

3. Maintenance Authority: C.C.I.T.T.

4. Scope: Hardware Standard. "The electrical characteristics specified in this Recommendation apply generally to interchange circuits operating with data signalling rates below the limit of 20,000 bits per second."

5. Relationship to Other Standards: C.C.I.T.T. Recommendation V.31 is for the lower speed circuits up to 75 bits per second.

6. Competitive Standards: C.C.I.T.T. V.28 is an alternative to the electrical characteristics of EIA RS-232-C.

7. Standardization Status: Approved by the C.C.I.T.T. plenary session at Geneva, Switzerland in 1972.

8. Implementation Status: Implemented primarily in Europe.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. Ira W. Cotton, NBS, (301) 921-2601; Mr. George E. Clark, NBS, (301) 921-3723; Mr. Arthur Freeman, U.S. Dept. of State, (202) 632-1007

11. Probable Sources of Information: Teletype Corporation

12. Bibliography: C.C.I.T.T. Recommendation V.28, "Green Book," Vol. VIII, Data Transmission, pp. 132-135.

13. Comments: C.C.I.T.T. is the French abbreviation for International Consultative Committee on Telegraph and Telephone. In most nations of the world (but not in the U.S.), its recommendations are given the force of law. The U.S. is represented on the CCITT by the U.S. Department of State. By way of contrast, the U.S. is represented on ISO and IEC by ANSI. The CCITT is an organ of the International Telecommunications Union (ITU) which is reported to be the oldest international standardizing body in the world. The ITU is now an organ of the United Nations.

1. Designation: C.C.I.T.T. Recommendation V.31

2. Title: Electrical Characteristics for Single-Current Interchange Circuits Controlled by Contact Closure

3. Maintenance Authority: C.C.I.T.T.

4. Scope: Hardware Standard. "In general, the electrical characteristics specified in this Recommendation apply to interchange circuits operating at data signalling rates up to 75 bits per second."

5. Relationship to Other Standards: C.C.I.T.T. Recommendation V.28 is for higher speed circuits up to 20,000 bits per second.

6. Competitive Standards:

7. Standardization Status: Approved by the CCITT plenary session at Geneva, Switzerland, in 1972.

8. Implementation Status: Implemented primarily in Europe.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. Ira W. Cotton, NBS, (301) 921-2601; Mr. George E. Clark, NBS, (301) 921-3723; Mr. Arthur Freeman, U.S. Department of State, (202) 632-1007

11. Probable Sources of Information: Teletype Corporation

12. Bibliography: C.C.I.T.T. Recommendation V.31, "Green Book," Vol. VIII, Data Transmission, pp. 140-142

13. Comments: C.C.I.T.T. is the French abbreviation for International Consultative Committee on Telegraph and Telephone. In most nations of the world (but not in the U.S.), its recommendations are given the force of law. The U.S. is represented on the CCITT by the U.S. Department of State. By way of contrast, the U.S. is represented on ISO and IEC by ANSI. The CCITT is an organ of the International Telecommunications Union (ITU) which is reported to be the oldest international standardizing body in the world. The ITU is now an organ of the United Nations.

1. Designation: ANSI X3.42-1975

2. Title: American National Standard for the Representation of Numeric Values in Character Strings for Information Interchange

3. Maintenance Authority: ANSI X3L5

4. Scope: This standard specifies the syntax of the elements of three sets of character strings which are decimal positional representations of numeric values for use in the interchange of numeric values between independent data processing systems and products. This standard also provides guidance for developers of programming standards and implementors of programming products.

5. Relationship to Other Standards: When used to represent all numeric values recorded on storage media or transmitted on data channels conforming to the appropriate American National Standards, would ensure that any recipient of a representation of a number attributes the same value to it as the originator, whether or not they are operating in the same system, programming language, or architecture.

6. Competitive Standards: None

7. Standardization Status: Standard published Aug. 4, 1975, by ANSI. This standard is in the process of international standardization by ISO.

8. Implementation Status: All standard programming languages (PL/I, FORTRAN, COBOL, BASIC, MUMPS) either conform to this standard or are being revised during their current revision cycle to conform to it.

9. Known Manufacturing Uses: All data files that are used by a different programming language than produced them and all data files that are applied on a different computer system or computer architecture.

10. Known Sources of Information: Mrs. Frances E. Holberton, NBS, (301) 921-3491; Mr. William F. Hanrahan, X3 Secretary.

11. Probable Sources of Information:

12. Bibliography: ANSI X3.42-1975

13. Comments:

1. Designation: ANSI X3.28-1976 Communication Protocol (Link Level) Standards - Character Oriented

2. Title: Procedures for the Use of the Communication Control Characters of ASCII in Specified Data Communication Links

3. Maintenance Authority: ANSI X3S3, Task Group 3

4. Scope: Protocols for Link Level Data Communication

5. Relationship to Other Standards:

   ANSI X3.4 (ASCII Character Set, Control Characters Used to Format Transmission)
   ISO R1745-1971 (Dialect)
   ECMA-16, 1973 (Dialect)[1]
   ISO R2111-1972 (Extension to Base Mode for Code-Independent Information Transfer)
   ISO R2629-1973 (Extension to Basic Mode for Conversational Information Transfer)
   ECMA-24, 1969 (Extension to Basic Mode for Code-Independent Information Transfer)
   ECMA-26, 1971 (Extension to Basic Mode for Recovery Procedures)
   ECMA-27, 1971 (Extension to Basic Mode for Abort and Interrupt Procedures)
   ECMA-28, 1971 (Extension to Basic Mode for Multiple Station Selection)
   ECMA-29, 1971 (Extension to Basic Mode for Conversational Information Transfer)
   ECMA-37, 1972 (Extension to Basic Mode for Supplementary Transmission Functions)

6. Competitive Standards: IBM's Binary Synchronous Communications, BISYNC/IBM Order No. GA27-3004- 2/10/70 (more extensive than X3.28, and utilizing EBCDIC Character Set).

7. Standardization Status: Revised Standard Issued in 1976.

8. Implementation Status: No known implementations adhering strictly to the standard classes of procedures. Each computer manufacturer has implemented a different part of X3.28. The standard specifies approximately 140 different system configurations that can be implemented conforming to the standard.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723.

11. Probable Sources of Information:

12. Bibliography: ANSI X3.28-1971, "Procedures for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links", American National Standards Institute, Inc., New York, NY, 10018.

13. Comments: Not a FIPS standard because does not provide for compatibility and data interchange among different systems.

---

[1] ECMA is European Computer Manufacturers Association

1. Designation: ANSI X3S34/589 (Draft 5) Communication Protocol (Link Level) Standards - Bit Oriented

2. Title: Proposed ANS for Advanced Data Communication Control Procedures (ADCCP) (Draft 5, 4/9/76)

3. Maintenance Authority: ANSI X3S3, Task Group 4

4. Scope: Hardware/Software. The (proposed) standard establishes procedures to be used on synchronous communications links.

5. Relationship to Other Standards: IBM Synchronous Data Link Control (SDLC) is a subset of ADCCP. IBM Document GA27-3093-1

6. Competitive Standards: ANSI X3.28 (character oriented), IBM BISYNC (Character oriented), DEC DDCMP

7. Standardization Status: Draft 5 being circulated for letter ballot.

8. Implementation Status: No known implementation operational. IBM's SDLC may be functional at this time. A number of microprocessor chips are being developed to be used as ADCCP link controllers; one known effort is Motorola.

9. Known Manufacturing Uses:

10. Known Sources of Information: ANSI Committee X3, Tech. Committee X3S3, Task Group 4; Mr. George E. Clark, NBS, (301) 921-3723.

11. Probable Sources of Information: IBM, Honeywell

12. Bibliography:

ANSI X3S34/589 (Fifth Draft) 4/9/76, Advanced Data Communication Control Procedures, American National Standards Institute.

Donnan, R. A., and J. Ray Kersey, "Synchronous Data Link Control: A Perspective", IBM Systems J., 13, 2, 197.

Sanders, R. W., and V. G. Cerf, "Compatibility or Chaos in Communications", Datamation, 3/76, pp. 50-55.

13. Comments: IBM is known to be basing most of its networking efforts on the use of SDLC as a link-level protocol.

1. Designation: DDCMP Communication Protocol (Link Level) - Bit Oriented

2. Title: DDCMP - Digital Data Communications Message Protocol, Ed. 3, 12/10/74

3. Maintenance Authority: Digital Equipment Corp., Maynard, Mass., 01754

4. Scope: Hardware/Software protocol to support message communication between (processes running on) computers.

5. Relationship to Other Standards:

6. Competitive Standards:

   Bit-oriented: ADCCP (ANSI X3S34/589, Draft 5); SDLC (IBM Document GA27-3093-1)
   Character-oriented: ANSI X3.28-1971; IBM Binary Synchronous Communications Protocol (IBM Document GA27-3004-2)

7. Standardization Status: Apparently an internal corporate standard for network implementations utilizing DEC computer equipment.

8. Implementation Status: In limited use in-house at DEC/Maynard.

9. Known Manufacturing Uses: None

10. Known Sources of Information: Mr. Stu Wecker, DEC, Maynard, Mass., 01754

11. Probable Sources of Information:

12. Bibliography: DDCMP: Digital Data Communications Message Protocol, Specification Document, Ed. 3, 12/10/74.

13. Comments: DDCMP is the (physical) link-level protocol supporting DNA, the Digital Network Architecture. The latter is the philosophical basis for the DECNET network systems yet to appear.

1. Designation: SNA - Systems Network Architecture

2. Title: Systems Network Architecture

3. Maintenance Authority: International Business Machines Corp., White Plains, N.Y. 10604

4. Scope: System (hardware and software) architecture for data communications and teleprocessing applications development and implementation. Includes the specification of link and higher levels of protocol.

5. Relationship to Other Standards: Similar in intent to CCITT X.25 and DEC's DNA. Specifics of various levels differ.

6. Competitive Standards: CCITT Recommendation X.25; Digital Equipment Corp. Digital Network Architecture (DNA) ARPANET Imp/Host protocol

7. Standardization Status: Not intended as a standard. SNA is being used in the development of several IBM product lines.

8. Implementation Status: Several IBM product lines, including point-of-sale terminal systems, are in various stages of implementation.

9. Known Manufacturing Uses: None

10. Known Sources of Information: IBM Systems Development Division, Advanced Systems Architecture, Dept. E97, P.O. Box 12195, Research Triangle Park, North Carolina, 27709

11. Probable Sources of Information:

12. Bibliography:

"Systems Network Architecture - General Information", IBM Doc. GA27-3102-0.
McFadyen, J. H. "Systems Network Architecture: An Overview", IBM Systems J., 15, 1, 1976, pp. 4-23.
Cullen, P. G. "The Transmission Subsystem in Systems Network Architecture", IBM Systems J., 15, 1, 1976, pp. 24-38.
Hobgood, W. S., "The Role of the Network Control Program in Systems Network Architecture", IBM Systems J., 15. 1, 1976, pp. 39-52.

13. Comments:

1. Designation: Digital Network Architecture

2. Title: Digital Network Architecture

3. Maintenance Authority: Digital Equipment Corp., Maynard, Mass. 01754

4. Scope: DNA is an architecture designed to permit the implementation of networks for data communication. It encompasses link (DDCMP), host/host (NSP), and process/process (DAP) level protocols for data and control communication.

5. Relationship to Other Standards: Similar in concept to IBM's SNA, CCITT Recommendation X.25, ARPANET Imp/Host Protocol

6. Competitive Standards: IBM's SNA, CCITT X.25, ARPANET Imp/Host Protocol

7. Standardization Status: Unknown

8. Implementation Status: Partial, primarily in-house.

9. Known Manufacturing Uses: Unknown

10. Known Sources of Information: Stu Wecker, Digital Equipment Corp., Maynard, Mass.

11. Probable Sources of Information:

12. Bibliography:

Wecker, S. "The Design of DECNET - A General Purpose Network Base" Presented at ELECTRO/76, Boston Mass., 5/76.
Digital Equipment Corp., "Digital Network Architecture - Design Specification for Network Services Protocol (NSP)", 7/10/75.
Digital Equipment Corp., "Digital Network Architecture - Design Specification for Digital Data Communications Message Protocol (DDCMP)", 12/10/75.

13. Comments:

1. Designation: CCITT Recommendation X.25 Communication Protocol (Network Level) Standards - Packet Oriented

2. Title: CCITT Recommendation X.25 - Interface Between Data Terminal Equipment and Data Circuit-Termination Equipment for Terminals Operating in the Packet Mode on Public Data Networks

3. Maintenance Authority: CCITT

4. Scope: Hardware/Software standard covering protocols relating to packet-switched computer networks. Covers three levels: Physical (Level 1) - X21 (or RS-232); Link Access (Level 2) - HDLC Procedures; Packet format and control (Level 3) - Virtual Calls, Circuits.

5. Relationship to Other Standards:
CCITT Recommendation X.1 - Classes of service for DTE in packet mode.
CCITT Recommendation X.2 - User facilities in packet mode.
CCITT Recommendation X.21, X.21bis - DTE/DEC Interface Characteristics.
CCITT Recommendation X.92 - Logical control links.
CCITT Recommendation X.95 - Network parameters
CCITT Recommendation X.96 - Call progress signals.

6. Competitive Standards: IBM's System Network Architecture (SNA), IBM Document GA27-3102-0; DEC's Digital Network Architecture (DNA).

7. Standardization Status: Draft standard. Not yet official.

8. Implementation Status: Unknown

9. Known Manufacturing Uses: None

10. Known Sources of Information: Mr. George E. Clark, NBS, (301) 921-3723.

11. Probable Sources of Information:

12. Bibliography: Sanders, Ray W., Vincton G. Cerf, "Compatibility or Chaos in Communications", Datamation, 3/76, pp. 50-55; CCITT Recommendation X.25 (Doc. X3S37-76-14 and ANSI X3S33-76-6)

13. Comments: This recommendation provides the electrical, link, and packet level procedures to implement a data network. The first two are based on existing standards or draft standards, and the third level is defined in the X.25 document.

1. Designation: ARPANET Imp/Host Protocol

2. Title: ARPANET Imp/Host Protocol

3. Maintenance Authority: Bolt, Beranek and Newman, Inc., 50 Moulton St. Cambridge, Mass. 02138

4. Scope: Software/Hardware specification of protocol governing the interface between a DTE (Host) and the ARPANET (i.e., Imp).

5. Relationship to Other Standards: Similar to CCITT X.25, DEC's DNA and IBM's SNA.

6. Competitive Standards:

7. Standardization Status: None

8. Implementation Status: Operational

9. Known Manufacturing Uses: None

10. Known Sources of Information: Network Control Center, BBN, Cambridge, Mass.

11. Probable Sources of Information:

12. Bibliography:

Bolt Beranek and Newman, "Specifications for the Interconnection of a Host and an Imp", BBN Report 1822, Rev. 4/73.

Heart, F. E., et al., "The Interface Message Processor for the ARPA Computer Network", Proc. SJCC, 5/7/70, pp. 551-567.

13. Comments:

1. Designation: Fourth Draft Proposed Standard "ANSI X.21" (Document No. X3S37-75-54/4)

2. Title: General Purpose Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment for Synchronous Operation on Public Data Networks

3. Maintenance Authority: ANSI Task Group X3S37

4. Scope: This standard defines the interface characteristics, interface procedures and timing of events, signal formats, and failure detection and isolation for a general purpose interface between data terminal equipment (DTE) and data circuit terminating equipment (DCE) for synchronous operation on public data networks.

5. Relationship to Other Standards: (1) Essentially the same as CCITT Recommendation X.21. (2) An alternative to EIA Standards RS-232-C and RS-XYZ that are both intended for DTE interconnections to analog network facilities. (3) A portion of CCITT Recommendation X.25 on packet switching.

6. Competitive Standards: See 5. (2) above.

7. Standardization Status: Fourth draft completed by X3S37 as a result of a letter ballot. This draft will probably be forwarded to X3.

8. Implementation Status: None

9. Known Manufacturing Uses: None

10. Known Sources of Information: Mr. George E. Clark, Jr., NBS (301) 921-3723 and Mr. J. G. Griffis, DCA (703) 437-2247.

11. Probable Sources of Information: Mr. S. M. Harris, Mitre Corporation (617) 271-3587; Mr. B. D. Wessler, Telenet Communications Corporation (202) 637-7925; and Mr. Vincent Dovydaitis, AFCS/LO (617) 861-4801.

12. Bibliography: None

13. Comments: Telephone systems in the future will be based on digital transmission systems. A/D and D/A converters will be placed in the hardset. A 4000HZ audio bandwidth will be achieved by digitizing at an 8KHZ sampling rate with 7 bits resolution. This means that each phone will use a 56 Kbit digital line. X.21 is the appropriate DTE/DCE protocol for such a transmission system. Limited implementation now; widespread implementation in the 1980's.

1. Designation: CCITT Recommendation X.21

2. Title: Interface Between Data Terminal Equipment and Data Circuit Terminating Equipment for Synchronous Operation on Public Data Networks

3. Maintenance Authority: CCITT

4. Scope: This standard defines the interface characteristics, interface procedures and timing of events, signal formats, and failure detection and isolation for a general purpose interface between data terminal equipment (DTE) and data circuit terminating equipment (DCE) for synchronous operation on public data networks.

5. Relationship to Other Standards:
   (1) Essentially the same as a fourth draft ANSI proposal (Document No. X3S37-75-54/4)
   (2) An alternative to EIA standards RS-232-C and RS-XYZ that are both intended for DTE interconnections to analog network facilities.
   (3) A portion of CCITT Recommendation X.25 on packet switching.

6. Competitive Standards: See 5. (2) above.

7. Standardization Status:

8. Implementation Status: None.

9. Known Manufacturing Uses: None.

10. Known Sources of Information: Mr. George E. Clark, Jr., NBS (301) 921-3723 and Mr. J. G. Griffis, DCA (703) 437-2247.

11. Probable Sources of Information: Mr. S. M. Harris, Mitre Corporation (617) 271-3587; Mr. B. D. Wessler, Telenet Communications Corporation (202) 637-7925; and Mr. Vincent Dovydaitis, AFCS/LO (617) 861-4801.

12. Bibliography: None.

13. Comments: Telephone systems in the future will be based on digital transmission systems. A/D and D/A converters will be placed in the hardset. A 4000HZ audio bandwidth will be achieved by digitizing at an 8KHZ sampling rate with 7 bits resolution. This means that each phone will use a 56 bit digital line. X.21 is the appropriate DTE/DCE protocol for such a transmission system. Limited implementation now; widespread implementation in the 1980's.

1. Designation: Draft Proposed Standard ANSI X3T92/064

2. Title: Class B Device Level Interface (for minicomputer systems)

3. Maintenance Authority: ANSI X3T92

4. Scope: These specifications define an interconnection between a peripheral device and a controller in which:
   - The data exchanged between interconnected defices is digital.
   - The number of devices interconnected via the interface is limited to four.
   - The total transmission path length of the interconnecting cables does not exceed 100 feet (30 meters).
   - The data rate across the interface does not exceed the maximum rate designated for each device class.

5. Relationship to Other Standards: None

6. Competitive Standards: None

7. Standardization Status: Draft proposed standard has not yet been completed by the Task Group to include electrical characteristics and device specifications.

8. Implementation Status: Similar to PERTEC magnetic tape drive interface.

9. Known Manufacturing Uses: None

10. Known Sources of Information: Mr. George E. Clark, Jr., NBS (301) 921-3723 and Mr. J. M. Bakshi, NBS (301) 921-3723.

11. Probable Sources of Information: Mr. Delbert Showmaker, GSA (202) 566-1180 and Mr. G. S. Robinson, Inforex, Inc. (617) 272-6470.

12. Bibliography:

13. Comments:

1. Designation: Draft Proposed Standard ANSI X3T9/600, Revision 2

2. Title: Working Paper for a Draft Proposed American National Standard for I/O Channel Interface

3. Maintenance Authority: ANSI X3T9

4. Scope: This American National Standard specifies functional, electrical, and mechanical characteristics of the interface between I/O control units and channels in general purpose computer systems.

5. Relationship to Other Standards: None

6. Competitive Standards: None

7. Standardization Status: Draft proposed standard (revision 2) has been revised to reflect results of X3T9 letter ballot and has been forwarded to X3.

8. Implementation Status: Implemented by Amdahl, Itel, and IBM central processors and in the I/O controllers built by STC, Telex, and other independent peripheral suppliers.

9. Known Manufacturing Uses: None

10. Known Sources of Information: Mr. George E. Clark, Jr., NBS (301) 921-3723 and Mr. J. M. Bakshi, NBS (301) 921-3723.

11. Probable Sources of Information: Mr. Delbert Showmaker, GSA (202) 566-1180 and Mr. Richard Guyette, IBM (914) 463-8153.

12. Bibliography: None

13. Comments: None

STANDARDS FOR COMPUTER SYSTEMS

COMMUNICATION CODE STANDARDS

CHARACTER CODE SETS

- ASCII
- Hollerith
- EBCDIC
- Numerical Control

CODE CONVERSION PROBLEMS

COLLATING SEQUENCE PROBLEMS

- Relevance to CAM Systems
- Relevance to Software Portability

RECOMMENDATIONS ON CODING

PROTECTION OF CAM DATA BY ENCRYPTION

RECOMMENDATION ON ENCRYPTION

REFERENCES

SUMMARY DATA SHEETS

CHARACTER CODE SETS

Successful implementation of modular computer/communications equipment requires well-defined interface specifications to accomplish the successful interchange of control signals and data between the various modules.

For adjacent equipment, the interface may signal each control function on a separate wire, and the data may appear as parallel signalling of bits on many wires. Thus, the interface may contain many wires. Some microcomputer bus-type interfaces employ 100 wires.

Where great distances are involved, the entire interface is reduced to two or four wires or a single microwave beam, and the control and data are accomplished by a stream of bits. Groups of successive bits may represent characters, so that the bit stream groups (usually 5 to 11 bits) represent a character stream. It has been shown that a stream of characters coded according to a recognized standard is the most certain of achieving successful interchange among dissimilar computers.(1)

ASCII Standard Code

In the United States, the standard coded character set is the American Standard Code for Information Interchange (ASCII), ANSI Standard X3.4-1968, also adopted as FIPS PUB 1. Internationally the standard is similar to ASCII and is ISO-646 or CCITT V.3, International Alphabet No. 5. ASCII and its international counterparts are defined as 7-bit codes, having 128 characters. An 8-bit version, having 256 characters, is being developed along the lines described in code extension standards, such as ANSI X3.14-1974, FIPS PUB 35, and ISO 2022 as well as ECMA-35. All of these code extension standards are similar, having been coordinated internationally.

In the numerical control (NC) or computer-aided manufacturing (CAM) areas, the 128 characters of ASCII appear to be adequate. The EIA standard RS-358 is a subset of ASCII for numerical control employing less than half of the ASCII characters. However, many computers represent characters as 8-bit "bytes." In 8-bit environments, ASCII characters should be represented in a standard manner, according to FIPS PUB 35 (ANSI X3.41-1974).

Well-defined ANSI standard interfaces exist for the reading, writing, or representation of ASCII characters on paper tapes, magnetic tapes on reels, cassettes or cartridges, and Hollerith punched cards. In all of these media, the ASCII code should be used as prescribed in these various ANSI standards or pending ANSI standards.

Hollerith Standard Code

The Hollerith Punched Card Code Standard, ANSI X3.26 (FIPS PUB 14) was adopted in 1970 and specifies 256 different hole patterns for twelve row punched cards. Hole patterns include the 128 characters of the ASCII Code, ANSI X3.4-1968 (FIPS PUB 1) plus 128 additional patterns.

EBCDIC Standard Code

ECBDIC is the Extended Binary Coded Decimal Interchange Code defined in IBM Corporate Systems Standard 3-3320-022. The standard specifies the BCD coded representation of up to 256 characters used on IBM 360, 370, System 3 and System 32 computers.

Numerical Control Codes

In the area of character codes for numerical control of machine tools two coding conventions are in popular and widespread use. The older "EIA" code defined by EIA RS-244A of January 1967 is an odd parity code of 52 identifiable characters. This code was that used by Flexowriters in common use in the early days of NC for the preparation of NC control tapes. The newer "ASCII" code is defined by EIA RS-358 of July 1968. It specifies an even parity code for the same character set which is a subset of the full ASCII code.

Originally, both of these standards were recognized and in conflict. More recently the older "EIA" code has been rescinded. Still there exist many numerically controlled machine tools capable of interpreting only the "EIA" code. Newer control units are generally supplied with the ability to read either input coding option.

One slight variation of the "ASCII" coding scheme rapidly gaining acceptance is described in EIA Standards Proposal 1177-A. Recognizing that there is a need for two distinct types of data at the machine tool site, the standards proposal defines a Type 1 and Type 2 data on the input media. Type 1 is the traditional machine program data codes in accordance with EIA RS-358 as above. Type 2 data contains machine set-up instructions, initialization and operational parameter data coded in the full ASCII code. Thus there are three coding schemes prevalent on the input media for numerical machine controllers.

It is expected that future systems operating in a CNC or DNC environment will implement whatever final version of EIA SP1177A is adopted. The Air Force should use this standard for command of any NC tools involved in the ICAM program.

CODE CONVERSION PROBLEMS

Conversion of non-standard codes to and from ASCII is not always a trivial matter. There is supposedly a defined correspondence between the 256 character positions of 8-bit EBCDIC and the 128 character positions of 7-bit ASCII or the 256 character positions in 8-bit ASCII.

The entire basis for correspondence is made possible by the Hollerith Code. That is, both ASCII and EBCDIC representations on a punched card are well defined.

The Hollerith Punched Card Code Standard, ANSI X3.26-1970 (FIPS PUB 14) provides 256 hole patterns mapped into 8-bit ASCII in Table 1 of the Hollerith Standard. These same 256 hole patterns are shown mapped into EBCDIC in Appendix B, which is not part of the Hollerith Standards, but is included there for information. There is thus established a 1 to 1 to 1 correspondence between 256 card hole patterns, 256 ASCII bit patterns, and 256 EBCDIC bit patterns. However, IBM practice does not adhere fully to this correspondence somewhat spoiling the 1 to 1 mapping between EBCDIC and ASCII.

Some EBCDIC control and graphic characters are not contained in ASCII. However, IBM chooses to map these, via the Hollerith Punched Card Code, into ASCII character positions, rather than into the 128 available non-ASCII character positions. EBCDIC equivalent characters to those displaced are mapped elsewhere, and the correspondence is thus spoiled. Selected examples are shown in Figure 1.

There are four interchange separators in EBCDIC which correspond to the four information separators of ASCII. Only IFS and IRS are shown because of the possible confusion of EBCDIC FS (Field Separator) with ASCII FS (File Separator), and of EBCDIC RS (Reader Stop) with ASCII RS (Record Separator).

The EBCDIC square brackets are not shown in the principal defining table (Table IV of the CSS) but are shown as publishing and printing graphic options (in Table VII). There is no Cent Sign in ASCII. IBM has chosen to displace the ASCII Opening Bracket by the EBCDIC Cent Sign. However, in representing the ISO 7-Bit Code of ISO 646-1973, IBM drops the Cent Sign and uses the Hollerith hole pattern 12-8-2 to represent the ISO Left Square Bracket, as shown in Table X of the IBM CSS, as a displacement of a national use symbol.

The substitutions in ASCII of the symbols for Logical Or and Logical Not are permitted in the ASCII standard (FIPS 1/ANSI X3.4-1968). This was done by IBM in order to get all 60 of the PL/I language symbols into the 64 character subset of FIPS PUB 15. The ASCII Exclamation Point is displaced. The EBCDIC Exclamation Point then displaces an ASCII Bracket, and a ripple of confusion follows, as was shown in Figure 1.

It is important to note that these problems in code conversion only occur whenever characters are required to cross an interface. In these cases the coding of characters should adhere strictly to the ASCII Standard. This is true regardless of when characters are enveloped in a "code independent frame" or are represented in serial-by-bit form or in parallel-by-bit form. Internal computer codes, if different than ASCII, such as EBCDIC, should not be allowed to cross such interfaces. In this way the Air Force should not encounter problems with character coding in the multi vender, distributed, integrated computer system that is envisioned for the 1980's.

COLLATING SEQUENCE PROBLEMS

An even more serious problem than code conversion arises from differences in the collating sequence embedded in various coded character sets. The collating sequence in a computer determines

(1) the order of the records in a data file according to the relative binary values of the entries in a "sort key" (does W32 come before or after 37N?)

(2) the results of inequality comparison operations (is ZaQ3 smaller or larger than Z24J?)

As long as one keeps to a single computer system or a network of similar equipment no problems are caused by collating sequence. However, the advent of distributed manufacturing systems opens the prospects of a variety of computer hardware being linked together, of data files on one system being queried by another, and of data files and programs being freely transported between different sites. In this type of environment collating sequence can lead to differing results obtained from identical programs operating on identical data files.

The ASCII standard, ANSI X3.4-1968 (FIPS 1) section 6.3 states; "The relative sequence of any two characters, when used as a basis for collation, is defined by their binary values." The IBM Corporate Systems Standard 3-3220-002 for EBCDIC states in Section 1.1 that it defines a collating sequence. ANSI Standard X3.27-1969, Magnetic Tape Labels for Information Interchange also provides guidance on structuring data files.

CONTROL AND GRAPHIC CHARACTERS OF IBM EBCDIC WHICH MAP VIA HOLLERITH HOLE PATTERNS INTO 8-BIT ASCII IN POSSIBLY CONFUSING WAYS

IBM Name of Character	IBM EBCDIC Character	IBM EBCDIC Position Hex. Col. Row	Standard Hollerith Hole Pattern	Corresponding ASCII Position Column/Row	ASCII Symbol	ASCII Name
Interchange File Separator	IFS	1C	11-9-8-4	01/12	FS	File Separator
Field Separator	FS	22	0-9-2	08/2	None	None
Interchange Record Separator	IRS	1E	11-9-8-6	01/14	RS	Record Separator
Reader Stop	RS	35	9-5	09/5	None	None
Tape Mark	TM	13	11-9-3	01/3	DC3	Device Control 3
Cent Sign	¢	4A	12-8-2	05/11	[	Opening Bracket
Open Square Bracket	[	AD	11-0-8-5	13/5	None	None
Close Square Bracket	]	BD	12-11-0-8-5	14/5	None	None
Exclamation Point	!	5A	12-8-2	05/13	]	Closing Bracket
Logical Or	|	4F	12-8-7	02/1	!	Exclamation Point
Logical Not	¬	5F	11-8-7	05/14	^	Circumflex

Figure 1

ASCII and EBCDIC define different collating sequences. The ASCII collating sequence, in general terms is "Space," Special Symbols, Numbers, Capital Letters, Small Letters. The EBCDIC collating sequence in general terms is "Space," Special Symbols, Small Letters, Capital Letters, Numbers.

ASCII collating sequence is defined in FIPS PUB 1 and 7, according to the ASCII standard, ANSI X3.4-1968. EBCDIC is defined only in IBM Corporate Systems Standard 3-3220-002. A draft revision to FIPS PUB 1 and 7, published in the Federal Register on December 29, 1975, pages 59607-08, "Revised Instructions for Implementing Standard Character Codes and Collating Sequence," strengthens the requirements for the use of ASCII collating sequence.

ASCII is the standard collating sequence in most minicomputers and microprocessors. EBCDIC is the de facto collating sequence in IBM 360, 370, System 3, System 32, and directly compatible computers. ASCII is the de facto collating sequence in all DEC computers, most NCR computers and some UNIVAC and Honeywell computers.

A data file in a computer system usually encompasses a well-defined area of interest, such as a "Payroll File," an "Inventory File," and the like. A "file" contains many"records." In a payroll file, there is a record for each item kept in inventory. The records in a file are kept in a specified sequence, usually determined by a "sort key." The various records are arranged according to the sort key, usually in ascending numerical order or in 26-letter alphabetical order. For simple sort keys, the order is the same no matter what kind of computer is used. Some sort keys may be pure binary numbers having any number of bits. Other sort keys can contain more complex arrays of characters, such as mixed upper and lower case letters, punctuation marks, special symbols as well as decimal digits. For complex sort keys, the order of the records is usually a "default" sequence determined by the native character code of the computer.

Two principal character codes are presently used in computers. One is ASCII (American Standard Code for Information Interchange) as specified by FIPS PUB 1, ANSI X3.4-1968, or ISO 646-1973. The other code is EBCDIC (Extended Binary Coded Decimal Interchange Code) as specified in IBM Corporate Systems Standard 3-3220-002 or variations by other mainframe vendors. The collating sequences of ASCII and EBCDIC are the same for simple sort keys, such as numerics or the 26 capital letters. But for more complex sort keys, the collating sequences are radically different. Computer control function characters are, of course, not used in sort keys. For the graphic characters, the collating sequence of ASCII is from low to high value as follows: "Space," punctuation and special symbols, numbers, capital letters, lower case letters, with some special symbols between these major groups. In EBCDIC, the collating sequence of the graphic characters is: "Space," punctuation and special symbols, lower case letters, capital letters, and numbers, with some special symbols between these major groups.

For the same data file, a sort key using most of the graphic characters of ASCII or EBCDIC would produce a record sequence based upon the collating sequence of ASCII or EBCDIC, unless otherwise specified. These two record sequences would be considerably different. A clerk could learn to use either record sequence as an index, but would have great difficulty transferring from one sequence to the other. This has occurred, for example, in the case of large catalogs arranged by Federal Stock Number (FSN, alphanumeric) ordered according to two different collating sequences. Introducing new Federal Stock Number items into the two "master" files would require that the new records be sorted by FSN according to the collating sequence of each file, and then merged into the master file. Data transferred from one "master" file to the other would require a re-sort of the selected records into the sort sequence of the other before the data merge could occur efficiently.

Relevance to CAM Systems

In the CAM arena, it is not apparent whether there are any difficulties that might result from the use of computers having two different collating sequences. It is possible to postulate some. For example, suppose it were desired to generate an index list of all APT part programs. Should "CAP∆37" come before or after "CAP∆FOR∆COVER" where "∆" represents the character "Space"? Since "CAP∆" is the same for both titles, the sequence would be resolved by whether "3" is smaller (lower in the collating sequence) or larger (higher in the collating sequence) than "F". In ASCII collating sequence, numbers are lower than letters, so that "CAP∆37" would precede "CAP∆FOR∆COVER." In EBCDIC collating sequence, numbers are higher than letters and hence "CAP∆37" would follow "CAP∆FOR∆COVER." The point is that, in a large index, a programmer might miss the existence of a desired program if the index had been collated on one machine and was being searched on another.

A standard collating sequence in the CAM area would be preferable to a mixture, sometimes ASCII and sometimes EBCDIC.

The following simple example illustrates the differences between ASCII and EBCDIC collating sequence. A sort key contains only two character positions and the complete character set is comparised of the four characters 1, 9, A, Z. The complete collating sequences are:

Sequence Number	ASCII	EBCDIC
1	11	AA
2	19	AZ
3	1A	A1
4	1Z	A9
5	91	ZA
6	99	ZZ
7	9A	Z1
8	9Z	Z9
9	A1	1A
10	A9	1Z
11	AA	11
12	AZ	19
13	Z1	9A
14	Z9	9Z
15	ZA	91
16	ZZ	99

It can be seen that in a large file index, a clerk would have difficulty locating a particular item without knowledge of the collating sequence.

If both capital letters and small letters are allowed in a sort key, then the confusion would be even greater, since in ASCII capital "Z" collates ahead of small "a," while in EBCDIC small "z" collates ahead of capital "A".

In an alphanumeric sort key, if certain positions are always numeric and other positions are always alphabetic (capital letters or small letters but not both), then the collating sequence will be the same in ASCII or EBCDIC. Thus in the example above, if the first position is always numeric and the second position always alphabetic, the complete sequence will be:

Sequence Number	ASCII or EBCDIC
1	1A
2	1Z
3	9A
4	9Z

It can be seen from the 16-sequence table that the sequence of four does appear in the same sequence in the ASCII or the EBCDIC column. This uniformity in collating sequence is achieved by a constraint on the sort key which greatly reduces the number of keys (records) that can be represented by a given number of positions in the sort key. Consistent use of the ASCII collating sequence will remove the need for such simplifying constraints on sort keys, and will eliminate variations in the sequencing of complex sort keys, and will also give consistent results for computer program comparison operations.

Revelance to Software Portability

Comparison operations in computer programs generally compare one group of characters with another group of characters. If the groups of characters are "simple," such as numerics or 26 letters, then the results of the comparisons will be the same whether the character coding is in ASCII or in EBCDIC. However, if the character groups to be compared are more complex, then the inequality of the two groups can indicate that the former is "larger" in ASCII but "smaller" in EBCDIC. Computer programs in high-level languages, employing such comparisons, can thus give different results in ASCII or EBCDIC, because of the difference in the collating sequences of ASCII or EBCDIC. A standard collating sequence would eliminate this complication along with the sort key sequencing inconsistencies.

In the original COBOL programming language standard, the collating sequence was indicated to be whatever the computer vendor specified. As a consequence, some COBOL programs could, and did, give different results on different computer systems. This had the effect of spoiling the transferability of COBOL programs among various computers, although such transferability was claimed to be one of the advantages of using high-level programming languages. To overcome this disadvantage, the COBOL standard (FIPS PUB 21-1, ANSI X3.23-1974) has been modified to allow the programmer to specify the collating sequence.

SUMMARY OF RECOMMENDATIONS ON CODING

a) It is recommended that the USAF use the FIPS 1 ASCII coding of character set data wherever information crosses an interface between a CAM module and any other CAM, computer or communications module or device.

b) It is recommended that the USAF use the ASCII subset of EIA Standard RS-358 for Numerical Control applications and adopt the "type 1"/"type"2 data conventions of SP177A before it becomes a standard.

c) It is recommended that the USAF use the recognized FIPS/ANSI standard representations of ASCII in media, such as paper tapes, magnetic tapes, punched cards, cassettes, and cartridges.

d) It is recommended that the USAF represent 7-bit ASCII in a standard manner in 8-bit environments, according to FIPS PUB 35/ANSI X3.41-1974.

e) It is recommended that the USAF represent any extensions of ASCII in a standard manner in accordance with FIPS PUB 35/ANSI X3.41-1974.

f) It is recommended that the USAF use the ASCII collating sequence for sequencing file records according to sort keys.

g) It is recommended that the USAF use the ASCII collating sequence for determining the results of comparison operations in computer programs.

PROTECTION OF CAM DATA BY ENCRYPTION

In most CAM applications, no special protection of the data will be required and none should be used. In some cases, protection may be deemed important. If CAM data is to be transmitted by military communications, then military data encryption techniques should suffice. If CAM data protection is desired but military communications are not involved, then such protection can, and should be, accomplished by means of the NBS Data Encryption Standard.

The NBS Data Encryption Standards (DES) algorithm specifies the encryption of 64 bits of data into a 64 bit cipher based on a 64 bit key and the decryption of a 64 bit cipher block into a 64 bit data block based on a 64 bit key. The steps and the tables of the algorithm are completely specified and no options are left in the algorithm itself. Variations in implementing and using the algorithm provide flexibility as to the application of the algorithm in various places in a computer system or network, how the input if formatted, whether the data itself or some other source of input is used for the algorithm, how the key is generated and distributed, how often the key is changed, etc. These issues are covered in a separate NBS guideline.

Basic implementation of the algorithm is most easily done in special purpose electronic devices. Overall security is based on two primary requirements when using the DES algorithm: secrecy of the encryption key and reliable functioning of the algorithm. Implementation of the algorithm in dedicated electronic devices provides the following economic and security benefits:

1) Efficiency of algorithm operation is much higher in specialized electronic devices.

2) Basic implementation of the algorithm in specialized LSI electronic devices which can be used in many applications and environments will result in cost savings through high volume production.

3) Functional operation of the device may be tested and validated independent of the environment.

4) The encryption key may be entered (or entered and decrypted) into the device and stored there and hence never need appear elsewhere in the computer system.

5) The paths of data to and from the device may be controlled and monitored.

6) Unauthorized modification of the algorithm is very difficult in such a device.

7) Redundant devices may simultaneously perform the algorithm independently and the output may be tested before cipher is transmitted.

8) The device can be controlled externally in accordance with the requirements and environment of the application.

9) Implementation in special purpose devices (electronic devices or dedicated micro processing computers) will satisfy Government requirements for compliance with the standard.

RECOMMENDATION ON ENCRYPTION

Wherever security is needed in interchange of CAM information, the NBS Data Encryption Standard algorithm should be applied, unless its use is superseded by military communications requirements.

REFERENCES

(1) "Information Interchange Between Dissimilar Systems" by H.S. Meltzer and H.F. Ickes, Modern Data, Vol. 4, No. 4, April 1971, pp. 56-67.

(2) Hollerith Punched Card Code Federal Informat Processing Standards FIPS PUB 14 1970 U.S. Department of Commerce, National Bureau of Standards.

(3) Extended Binary Coded Decimal Interchange Code IBM Corporate Systems Standard CSS 3-3220-002, November 1970, IBM Systems Standards Department, Poughkeepsie, New York.

(4) Proposed Federal Information Processing Data Encryption Standard 1 Aug. 1975 Federal Register.

(5) Guidelines for Implementing and Using the NBS Data Encryption Standard, Draft Document, 10 Nov. 1975 Institute for Computer Sciences and Technology, National Bureau of Standards.

SUMMARY DATA SHEETS

The following Data Sheets summarize those interface standards which apply to Communications Codes.

1. Designation: FIPS PUB 1-1968/ANSI X3.4-1968 (ASCII)[1]

2. Title: American National Standard Code for Information Interchange

3. Maintenance Authority: ANSI X3L2

4. Scope: Hardware Standard. "This coded character set is to be used for the general interchange of information among information processing systems, communication systems, and associated equipment."

5. Relationship to Other Standards:

ISO 646-1973, (dialect)
CCITT V.3-1972 (dialect)
FIPS PUB 2-1968/ANSI X3.6-1965 (implementation, perforated tape)
FIPS PUB 3-1, 1973/ANSI X3.22-1973 (implementation, magnetic tape, 800 cpi)
FIPS PUB 7-1969 (implementation instructions, Presidential memo)
FIPS PUB 14-1971/ANSI X3.26-1970 (implementation, Hollerith punched card)
FIPS PUB 15-1971 (subsets: 95, 64, 16 graphic characters)
FIPS PUB 16-1971/ANSI X3.15-1966 (implementation, bit sequencing)
FIPS PUB 17-1971/ANSI X3.16-1966 (implementation, character structure, serial)
FIPS PUB 18-1971/ANSI X3.25-1968 (implementation, character structure, parallel)
FIPS PUB 25-1973/ANSI X3.39-1973 (implementation, magnetic tape, 1600 cpi)
FIPS PUB 35-1975/ANSI X3.41-1974 (code extension, 7 or 8 bits)
FIPS PUB 36-1975/ANSI X3.32-1973 (graphical representation of controls)
ANSI X3.14-1973 (implementation, magnetic tape, 200 cpi)
ANSI X3.28-1971 (implementation, communication control characters)
ANSI Z39.2-1971 (superset, magnetic tape, bibliographic interchange)
EIA RS-358 (1968) (subset, numerical machine control)
ECMA-6 (dialect)

6. Competitive Standards: EBCDIC, FIELDATA, TELETYPESETTER, CORRESPONDENCE Code (Some IBM Selectric Typewriter Terminals), EIA RS-RS-244A, and all other coded character sets in use prior to 1963.

7. Standardization Status: First approved in 1963. Revised in a major sense in 1967, with a further minor revision in 1968 into the standard now in effect. Another minor revision is expected in 1977.

8. Implementation Status: ASCII is the most widely implemented code in ADP terminals and in communication systems. It is implemented as the internal code of most minicomputers and microprocessors, all DEC computers, NCR Century and Criterion computers, and the newer large UNIVAC computers and many European computers. It would be more widely implemented in American computers except for the severe competition from EBCDIC in IBM 360, 370, and other computers compatible therewith.

9. Known Manufacturing Uses: Data and message communications; direct numerical control (EIA RS-358 subset); code used in minicomputers and microprocessors.

10. Known Sources of Information: Mr. Charles D. Card, UNIVAC, Chairman of ANSI X3L2, (215) 542-3675; Mr. John L. Little, NBS, member of ANSI X3L2, (301) 921-3723; Mr. William F. Hanrahan, Secretary of ANSI X3, (202) 466-2288

11. Probable Sources of Information: DEC, NCR, Honeywell, Teletype Corporation

12. Bibliography: FIPS PUB 1-1968/ANSI X3.4-1968 (base standard); FIPS PUB 7-1969 (implementation instructions); FIPS PUB 35-1975/ANSI X3.41-1974 (extension in 7 or 8 bits); Western Union Technical Bulletin 71-1, "The ASCII Codes (1963, 1967, and 1968 Versions)."

13. Comments: ASCII is a fundamental standard upon which many other hardware and software standards are based. It has counterparts in international and Federal standards, as well as in many foreign national standards.

[1]The standard also allows the abbreviation "USASCII," but ASCII is much more prevalent.

1. Designation: EIA RS-244A

2. Title: Character Code for Numerical Machine Control Perforated Tape

3. Maintenance Authority: None

4. Scope: Hardware Standards. This standard specifies the representation of the EIA Paper Tape Code used for the numerical control of machine tools.

5. Relationship to Other Standards: Extension of original RS-244 standard.

6. Competitive Standards: EIA RS-358 is a subset of FIPS PUB 2/ANSI X3.6-1965 used for numerical machine control perforated tape.

7. Standardization Status: EIA RS-244A approved in January 1967 but has been rescinded in favor of EIA RS-358.

8. Implementation Status: Widely used in perforated tape equipment used in the numerical control of machine tools.

9. Known Manufacturing Uses: Code is still being supplied on new equipment at users request to maintain compatibility with older equipment. On new equipment older EIA RS-244A and newer EIA RS-358 are often switch selectable.

10. Known Sources of Information: Mr. John L. Little, NBS (301) 921-3723.

11. Probable Sources of Information: Bendix Industrial Controls, General Electric Control Systems Division.

12. Bibliography: EIA RS-244A

13. Comments: Some users note that the odd parity of EIA RS-244A results in fewer holes per character and therefore a stronger paper tape when compared with the even parity EIA RS-358.

1. Designation: ISO 646-1973

2. Title: Seven-Bit Coded Character for Information Processing Interchange

3. Maintenance Authority: ISO/TC97/SC2

4. Scope: Hardware Standard. "This character set (of 128 characters) is primarily intended for the interchange of information among data processing systems and associated equipment, and within message transmission systems. This character set is applicable to all Latin alphabets."

5. Relationship to Other Standards: CCITT Recommendation V.3-1972 is essentially identical to this standard. Similar to ASCII (ANSI X3.4-1968) and the national standards of many other countries, e.g., GOST 13052-1967 (USSR). Similar to ECMA-6, ISO 2022-1973 (extension in 7 or 8 bits).

6. Competitive Standards: EBCDIC in IBM 360, 370 computers, and all coded character sets that were in use prior to 1963.

7. Standardization Status: First approved in December 1967 as 6 and 7-bit coded character sets. The 6-bit set was derived from an ECMA 6-bit code and the 7-bit one was derived from ASCII (X3.4-1967), each of which influenced the other. The current 1973 version relegates the 6-bit set to an appendix which is no longer part of the standard. The 1973 standard contains a "Basic Code Table" and an "International Reference Version" (IRV). The basic code table allows national options while the IRV does not.

8. Implementation Status: CCITT Recommendation V.3-1968 is identical to this standard except for a few restrictions governing international communication. National implementations, such as ASCII (ANSI X3.4-1968) can be considered implementations of this standard.

9. Known Manufacturing Uses: A subset of this standard, in EIA RS-358 (1968), is used as a perforated tape code for numerical control of machine tools.

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. Robert M. Brown, CBEMA, (202) 466-2288.

11. Probable Sources of Information: IBM, Sperry UNIVAC, Honeywell, DEC, NCR, Teletype Corporation

12. Bibliography: ISO 646-1973, available from ANSI.

13. Comments: Extension techniques for this basic code are given in ISO 2022-1973. A draft "ISO International Register of Character Sets to be used with Escape Sequences for Information Interchange in Data Processing" is published in document ISO/TC97/SC2 N1000, maintained by AFNOR in Paris, France.

1. Designation: CCITT V.3-1972

2. Title: International Alphabet No. 5

3. Maintenance Authority: CCITT

4. Scope: Hardware Standard. "This character set (of 128 characters) is primarily intended for the interchange of information among data processing systems and associated equipment, and within message transmission systems. This character set is applicable to all Latin alphabets."

5. Relationship to Other Standards: ISO 646-1973 is essentially identical to this standard. This standard is similar to ASCII (ANSI X3.4-1968) and to the national standards of many other countries, as well as to ECMA-6. See FIPS PUB1.

6. Competitive Standards: EBCDIC in IBM 360, 370 computers, and all coded character sets in use prior to 1963.

7. Standardization Status: First approved in 1968 at the CCITT plenary session at Mar del Plata, Argentina, and amended in 1972 at Geneva.

8. Implementation Status: National implementations, such as ASCII (ANSI X3.4-1968) can be considered implementations of this standard. It is implemented in many international communication networks.

9. Known Manufacturing Uses: A subset of this standard, in EIA RS-358 (1968), is used as a perforated tape code for numerical control of machine tools.

10. Known Sources of Information: Mr. Ira W. Cotton, NBS, (301) 921-2601; Mr. Arthur Freeman, U.S. Department of State, (202) 632-1007.

11. Probable Sources of Information: Teletype Corporation, ITT, RCA, Western Union

12. Bibliography: The International Telegraph and Telephone Consultative Committee (C.C.I.T.T.) Fifth Plenary Assembly, Geneva, 4-15 December 1972, "Green Book" Vol. VIII, Data Transmission, published by the International Telecommunications Union, 1973.

13. Comments: This standard was developed jointly with ISO 646-1973 and is virtually identical to that standard. C.C.I.T.T. is the French abbreviation for International Consultative Committee on Telegraph and Telephone. In most nations of the world (but not the U.S.A.), its recommendations are given the force of law. The U.S. is represented on the CCITT by the U.S. Department of State. By way of contrast, the U.S. is represented on ISO and IEC by ANSI. The CCITT is an organ of the International Telecommunications Union (ITU) which is reported to be the oldest international standardizing body in the world. The ITU is now an organ of the United Nations.

1. Designation: IBM CSS 3-3220-002, EBCDIC[1]

2. Title: Extended Binary Coded Decimal Interchange Code (LATIN ALPHABETS)

3. Maintenance Authority: IBM Systems Standards Department, Poughkeepsie, New York

4. Scope: "This standard defines for the IBM Corporation the BCD coded representation for up to 256 graphics and controls in punched cards, in magnetic tape, on data transmission lines, and in 8-bit BCD CPU's. It also defines a collating sequence."

5. Relationship to Other Standards: This is a commercial standard. In 1970 IBM distributed copies through the European Computer Manufacturers Association (ECMA) and encouraged its adoption as a standard. No formal recognition has been granted to EBCDIC as a national or international standard. It is used in various forms by some other computer vendors who wish to provide easy transition from IBM products to competitive ones.

6. Competitive Standards: ISO 646-1973, CCITT V.3-1972, ANSI X3.4-1968 (ASCII), FIELDATA, TELETYPESETTER.

7. Standardization Status: None, except as an IBM corporate systems standard.

8. Implementation Status: Implemented in IBM 360 and 370, System 3, System 32 computers, Amdahl computers and with variations in certain Burroughs, Honeywell and Univac (RCA) computers. Used in the RYAD series of computers built in the Soviet bloc countries in order to provide compatibility with IBM.

9. Known Manufacturing Uses: Used in IBM computer-aided design systems.

10. Known Sources of Information: Mr. Hubert F. Ickes, IBM, (914) 463-9779; Mr. Robert H. Follett, IBM, (301) 897-3471; Mr. John L. Little, NBS, (301) 921-3723.

11. Probable Sources of Information: Mr. Robert M. Brown, Vice Chairman of ANSI X3, CBEMA, (202) 466-2288.

12. Bibliography: IBM Corporate Systems Standard CSS 3-3220-002, November 1970, Extended BCD Interchange Code, Latin Alphabets. IBM System 370, Principles of Operation, GA 22-7000-4, File No. S/370-01, Appendix H.

13. Comments: Representation of the 128 ASCII characters in EBCDIC coding is shown in an Appendix (not part of the standard) to the Hollerith Punched Card Code, ANSI X3.26-1970 (FIPS PUB 14).

---

[1] Supersedes IBM CSS 2-8015-002, EBCDIC.

1. Designation: Encryption Algorithm

2. Title: Encryption Algorithm for Computer Data Protection

3. Maintenance Authority: U.S. Department of Commerce, NBS

4. Scope: This algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key.

5. Relationship to Other Standards: Identical to an IBM encryption algorithm.

6. Competitive Standards: None

7. Standardization Status: Published as a proposed Federal standard in the Federal Register, Vol. 40, No. 52, March 17, 1975, pp. 12134-12139.

8. Implementation Status: Implemented in IBM encryption/decryption equipment, and in integrated circuit chips supplied by several vendors.

9. Known Manufacturing Uses: IBM, NSA.

10. Known Sources of Information: Dr. Dennis K. Branstad, NBS, (301) 921-3861.

11. Probable Sources of Information: IBM, NSA.

12. Bibliography: Federal Register, Vol. 40, No. 52, March 17, 1975, pp. 12134-12139. U.S. Patent Nos. 3,796,830 and 3,798,359.

13. Comments: The Federal standard is identical to an IBM algorithm. IBM will grant a royalty-free license, as stated in the Federal Register cited above.

1. Designation: EIA RS-358 (1968)/ISO 840-1973

2. Title: Subset of USA Standard Code for Information Interchange (ASCII) for Numerical Machine Control Perforated Tape

3. Maintenance Authority: EIA EI-31

4. Scope: Hardware Standard. "This standard describes a subset of USAS X3.4-1967 (ANSI X3.4-1968) for numerically controlled machines and associated perforated tape preparation equipment."

5. Relationship to Other Standards: ANSI X3.4-1968 (base); FIPS PUB 2-1968/ANSI X3.6-1965 (implementation, perforated tape). ISO 840-1973 is the same coding. ISO 1113 is a compatible implementation in perforated tape.

6. Competitive Standards: EIA RS-244A-1967

7. Standardization Status: Approved by EIA in July 1968

8. Implementation Status: Implemented in all newer numerically-controlled machine tools. Some have a switch that allows either this standard or RS-244A coding to be used in the same machine.

9. Known Manufacturing Uses: Widely used as the perforated tape coding for numerically controlled machine tools.

10. Known Sources of Information: Mr. A. M. Wilson, EIA, (202) 659-2200; Mr. John L. Little, NBS, (301) 921-3723.

11. Probable Sources of Information: NMTBA

12. Bibliography: EIA RS-358, EIA RS-244A, ISO 840, ISO 1113

13. Comments: The original perforated tape code standard for numerical control was approved in July 1961 (before ASCII) as EIA RS-244, a BCD coding as used in Flexowriters. A revision, RS-244A was approved in January 1967 and the conflicting RS-358 was approved in July 1968. RS-244A will be rescinded in order to resolve the conflict. The character sets of EIA RS-244A and RS-358 are the same, but the coding is very different.

1. Designation: EIA RS-274-C

2. Title: Interchangeable Perforated Tape Variable Block Format for Positioning, Contouring and Contouring/Positioning Numerically Controlled Machines.

3. Maintenance Authority: EIA EI-31

4. Scope: This standard applies wherever a variable block format is used on perforated tape to control numerically controlled machines.

5. Relationship to Other Standards: This standard has replaced EIA RS-273-B (rescinded), variable block for positioning and straight cut, EIA RS-326-A (rescinded), fixed block for positioning and straight cut, and EIA RS-274-B, variable block for contouring and contouring/positioning. It may in turn be superseded by EIA SP-1177A, command and data format for advanced contouring and positioning.

6. Competitive Standards:

7. Standardization Status: RS-274 was approved in January 1963. It became ANSI standard X3.8-1965. RS-274-A was a revision and RS-274-B became ANSI standard X8.2-1968. RS-274-C was approved (SP-1147) in April 1974.

8. Implementation Status: Widely implemented in numerical control equipment.

9. Known Manufacturing Uses: Widely used in production where numerical control is involved.

10. Known Sources of Information: Mr. A. M. Wilson, EIA, (202) 659-2200; Dr. John M. Evans, Jr., NBS, (301) 921-2381

11. Probable Sources of Information: NMTBA

12. Bibliography: EIA RS-274-C

13. Comments: This was formerly known as EIA Standards Proposal No. 1147 (SP-1147), March 13, 1973. RS-274-C may be superseded by EIA SP-1177A.

1. Designation: EIA SP-1177A (a proposal currently under revision)

2. Title: Recommended Command and Data Format for Advanced Contouring and Positioning Numerically Controlled Machines

3. Maintenance Authority: EIA EI-31

4. Scope: Hardware Standard. "This standard is intended to serve as a guide in the coordination of system design to promote uniformity in part programming and operating techniques for inputting extended machine set-up, initialization, and/or operational parameter data."

5. Relationship to Other Standards: Machine program data are formatted in accordance with EIA RS-274-C. This standard, when approved, will supersede EIA Automation Bulletin No. 4, March 1969. It will also probably supersede EIA RS-274-C, which in turn superseded RS-273-A, RS-274-B, and RS-326.

6. Competitive Standards:

7. Standardization Status: This revision of SP-1177 was circulated for EIA ballot on April 23, 1975.

8. Implementation Status: Used in all new US computer numerical control (CNC) equipment.

9. Known Manufacturing Uses: Will be widely used in production where CNC is involved.

10. Known Sources of Information: Mr. A. M. Wilson, EIA, (202) 659-2200; Dr. John M. Evans, Jr., NBS, (301) 921-2381.

11. Probable Sources of Information: NMTBA

12. Bibliography: EIA Standards Proposal No. 1177-A (Revision of SP-1177), April 23, 1975.

13. Comments: This SP-1177A will become an EIA Recommended Standard (RS) when the various points of view and controversy are resolved. SP-1177A contains three key concepts: the overall architecture of an NC system, specially identifying interfaces to external components; an escape code to go from RS-358 to full ASCII ("type 1 and type 2 data"); and phonetic abbreviations for CNC control functions. The proposed standard is essentially a guideline for the development of advanced systems by NC control manufacturers.

1. Designation: FIPS PUB 36-1975/ANSI X3.32-1973

2. Title: Graphic Representation of the Control Characters of ASCII

3. Maintenance Authority: NBS/ANSI X3L2

4. Scope: Hardware Standard. This standard provides a graphic representation of the control characters of ASCII, including SPACE and DELETE. The standard contains two alternative sets of representations: a pictorial representation and an alphanumeric representation.

5. Relationship to Other Standards: Gives single or dual symbol representation of the 32 controls as well as SPACE and DELETE of the ASCII code standard, FIPS PUB 1/ANSI X3.4-1968.

6. Competitive Standards: None

7. Standardization Status: The ANSI standard X3.32-1973 was first approved on July 3, 1973. FIPS PUB 36, adopting the ANSI standard in its entirety, was approved on June 1, 1975.

8. Implementation Status: Has been implemented on paper tape equipment that prints one symbol per tape character frame. Has also been implemented in some display terminals to display character streams, including controls. See comments below.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. C. D. Card, UNIVAC, Chairman ANSI X3L2, (215) 542-3675; Mr. John L. Little, NBS, (301) 921-3723; Mr. Robert M. Brown, Vice Chairman of ANSI X3, CBEMA, (202) 466-2288.

11. Probable Sources of Information: Honeywell, Teletype Corporation, Omron.

12. Bibliography: FIPS PUB 36, ANSI X3.32-1973

13. Comments: Implemented (except for the Backspace character representation) in Teletype Model 40 display/printing terminals. Implemented (except for the NULL character representation) in Omron Model 8025 keyboard display terminals.

1. Designation: FIPS PUB 14/ANSI X3.26-1970

2. Title: Hollerith Punched Card Code

3. Maintenance Authority: ANSI X3L2

4. Scope: Hardware Standard. This standard specifies 256 hole patterns in twelve-row punched cards. Hole patterns are assigned to the 128 characters of ASCII (FIPS PUB 1/ANSI X3.4-1968).

5. Relationship to Other Standards: This standard gives the implementation of FIPS PUB 1/ANSI X3.4-1968 (ASCII) in twelve-row punched cards. FIPS PUB 13/ANSI X3.21-1967, Rectangular Holes in Twelve-Row Punched Cards, gives dimensions and dimensional tolerances of the cards and holes. ANSI X3.11-1969, Specifications for General Purpose Paper Cards for Information Processing, gives properties of the card stock.

6. Competitive Standards: The round hole "90 column" cards formerly marketed by UNIVAC are obsolete. The 96-column round hole cards introduced with the IBM System 3 are not compatible with this standard.

7. Standardization Status: FIPS PUBs 13 and 14 were both approved on October 1, 1971. ANSI Standard X3.21-1967 was first approved in 1967. ANSI Standard X3.26-1970 was first approved in 1970.

8. Implementation Status: These standards are widely implemented in punched card accounting machines and as input media for computers.

9. Known Manufacturing Uses: Used in many applications where programs or data are entered into computers via rectangular-hole punched cards.

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. H. F. Ickes, IBM, (914) 463-9779.

11. Probable Sources of Information: Honeywell, UNIVAC

12. Bibliography: FIPS PUB 14/ANSI X3.26-1970, Hollerith Punched Card Code.

13. Comments: The Hollerith Punched Card Code has 256 hole patterns which map into 8-bit ASCII (see Table 1 of the ANSI Standard X3.26-1970) and also map into 8-bit EBCDIC, establishing the basis of code conversion between EBCDIC and ASCII, which is spoiled somewhat by IBM practice. See the writeup on "Code Conversion."

    The Hollerith Punched Card Code specified in ANSI Standard X3.26-1970 is based upon and earlier "Business" version having the symbols Ampersand, Commercial At, Number Sign, and Percent Sign included in a 48-character set. Several other versions, most notably a "Scientific" version which replaced some of the business symbols with FORTRAN algebra symbols Plus Sign, Apostrophe, Equals Sign, and Parentheses in a 48-character set are still in de facto use. When extended beyond 48 characters, these two versions employ the same set of Hollerith hole patterns in different ways, compatible with the earlier 48-character versions. The NBS UNIVAC 1108 "scientific" installation, for example, employs the standard hole patterns for some of the punctuation marks and special symbols (Period, Comma, Semicolon, Asterisk, Slant, Minus Sign, Dollar Sign) but deviates in the following hole pattern coding:

Hollerith Hole Pattern	ANSI X3.26-1970	NBS UNIVAC 1108 (Note 1)
12	Ampersand &	Plus Sign +
12-0 (Plus Zero)	Opening Brace {	Question Mark ?
11-0 (Minus Zero)	Closing Brace }	Exclamation Point !
8-2	Colon :	Ampersand &
8-3	Number Sign #	Equals Sign =
8-4	Commercial At @	Apostrophe '
8-5	Apostrophe '	Colon :
8-6	Equals Sign =	Greater Than >
8-7	Quotation Marks "	Commercial At @ (Master Space)
12-8-2	Opening Bracket {	(Not Used)
12-8-4	Less Than <	Closing Parenthesis)
12-8-5	Opening Parenthesis (	Opening Bracket {
12-8-6	Plus Sign +	Less Than <
12-8-7	Exclamation Point !	Number Sign #
11-8-2	Closing Bracket }	(Not Used)
11-8-5	Closing Parenthesis)	Closing Bracket }
11-8-7	Circumflex ^	Delta Δ (Note 2)
0-8-2	Reverse Slant \	Record Stop (Stop)
0-8-4	Percent Sign %	Opening Parenthesis (
0-8-5	Underline _	Percent Sign %
0-8-6	Greater Than >	Reverse Slant \
0-8-7	Question Mark ?	Losenge (Note 3)

Note 1 Input: UNIVAC 706 Card Reader
Output: UNIVAC 758 Printer

Note 2 Prints Circumflex (^) or Up Arrow (↑) on interactive ASCII terminals. Prints a triangle (Delta) (Δ) on batch terminals.

Note 3 Prints Quotation Marks (") on interactive ASCII terminals. Prints a Losenge or Rectangular Box on batch terminals.

Hollerith hole patterns for Space (no punches), 10 decimal digits (one punch), 26 Latin capital letters (two punches), as well as hole patterns for Period, Comma, and Asterisk (three punches) are universally standard, in accordance with ANSI X3.26-1970. Hole patterns for Minus Sign, Slant, and Dollar Sign are nearly universal, in accordance with ANSI X3.26-1970.

1. Designation: FIPS PUB 2/ANSI X3.6-1965

2. Title: Perforated Tape Code for Information Interchange

3. Maintenance Authority: ANSI X3B2

4. Scope: Hardware Standard. This standard specifies the representation of the Federal Standard Code for Information Interchange (FIPS PUB 1, ASCII) on perforated tape used in Federal information processing systems, communication systems, and associated equipment.

5. Relationship to Other Standards: See FIPS PUB 12-2, page 18 for 11 other standards related to this one, under "Media, Perforated Tape." EIA RS-358 is a subset of this code for numerical machine control perforated tape.

6. Competitive Standards: None recognized as standards. All perforated tape codes in use prior to 1965 are competitive. EIA RS-244A is an early competitive standard for numerical machine control perforated tape. Teletypesetter code is still used in newspaper communications and in typesetting perforated tape.

7. Standardization Status: FIPS PUB 2 was approved on November 1, 1968. ANSI X3.6-1965 was approved in 1965. Neither has been updated. EIA RS-244A has been rescinded in favor or EIA RS-358.

8. Implementation Status: Widely used in perforated tape equipment, including input/output to minicomputers and in communications.

9. Known Manufacturing Uses: The subset in EIA RS-358 is widely used in numerical control perforated tapes for drafting and machine control.

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. John B. Booth, Teletype Corporation, (312) 982-3630.

11. Probable Sources of Information: DEC, Honeywell

12. Bibliography: FIPS PUB 2/ANSI X3.6-1965, Perforated Tape Code for Information Interchange; FIPS PUB 12-2, page 18.

13. Comments: IBM has always emphasized punched cards and magnetic tape in preference to perforated tape. DEC uses perforated tape and magnetic tape in preference to punched cards.

1. Designation: ANSI X8.1-1968/ISO 841-1974/EIA RS-267-A-1967/NAS 938-1962

2. Title: Axis and Motion Nomenclature for Numerically Controlled Machines

3. Maintenance Authority: EIA EI-31; ISO/TC97/SC8

4. Scope: Hardware Standard. This standard defines axis and motion nomenclature for numerically controlled machines.

5. Relationship to Other Standards: Definitions of terms (in the ANSI/EIA standard) are in accordance with EIA Automation Bulletin 3B.

6. Competitive Standards:

7. Standardization Status: This standard was approved as NAS 938 in June 1959 and revised by NAS in February 1963. EIA RS-267 was approved in July 1962 and was revised in June 1967 as RS-267-A which was approved as ANSI X8.1 in March 1968. ISO R841 was approved in 1968 and reissued as ISO 841 in July 1974.

8. Implementation Status: Widely used in numerical control equipment.

9. Known Manufacturing Uses: Widely used in numerical control applications to drafting machines, plotters, and machine tools.

10. Known Sources of Information: Mr. A. M. Wilson, EIA, (202) 659-2200

11. Probable Sources of Information: NMTBA, AIA

12. Bibliography: ANSI X8.1-1968/ISO 841/1974/EIA RS-267-A-1967/NAS 938-1962, Axis and Motion Nomenclature for Numerically Controlled Machines

13. Comments: This standard appears to have been developed first as National Aerospace Standard NAS 938-1959, and then modified into EIA, ANSI, and ISO standards.

1. Designation: Code Conversion

2. Title: Conversion of Codes, EBCDIC to/from ASCII

3. Maintenance Authority: None (IBM/NBS)

4. Scope: Definition of the one-to-one correspondence between the 256 character positions (bit patterns) of 8-bit EBCDIC as used in IBM 360/370 computers and (1) the 128 character positions on character sequences of 7-bit ASCII in ANSI Standard X3.4-1968 (FIPS PUB 1) or (2) the 256 character positions in 8-bit ASCII of ANSI Standard X3.41-1974 (FIPS PUB 35).

5. Relationship to Other Standards: The Hollerith Punched Card Code Standard, ANSI X3.26-1970 (FIPS PUB 14) provides 256 hole patterns mapped into 8-bit ASCII (in Table 1 of the Hollerith standard). These same 256 hole patterns are shown mapped into EBCDIC in Appendix B, which is not part of the Hollerith standard, but is included there for information. There is thus established a 1-to-1-to 1 correspondence between 256 card hole patterns, 256 ASCII bit patterns, and 256 EBCDIC bit patterns. However, IBM practice does not adhere fully to this correspondence, as noted below.

6. Competitive Standards: 8-BIT ASCII and EBCDIC can be considered to be competing standards, although EBCDIC has only the status of a corporate systems standard of IBM. Both define a different collating sequence which can serve as a default sequence for ordering files if no other file sequence is specified. Both ASCII and EBCDIC have "Interchange" in their names.

7. Standardization Status: None for the conversion, except as given by the Hollerith Punched Card Code Standard (FIPS PUB 14, ANSI X3.26-1970).

8. Implementation Status: Implemented in various ways in different computer types and systems, leading to confusion and complicating program and data transferability.

9. Known Manufacturing Uses: Used wherever IBM 360/370 computers operate with ASCII external devices or communications.

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. H. F. Ickes, IBM, (914) 463-9779.

11. Probable Sources of Information: Honeywell, UNIVAC

12. Bibliography: FIPS PUB 14/ANSI X3.26-1970 (Hollerith Punched Card Code). "Correspondences of 8-Bit and Hollerith Codes for Computer Environments -- A Tutorial," Comm. of the ACM, Vol. 11, No. 11, November 1968, pp. 783-789 with corrigenda in Vol. 12, No. 5, May 1969, p. 294. "IBM System/370 Principles of Operation," IBM GA 22-7000-4, File No. S/370-01 (Appendix H on EBCDIC Translate (TR) instruction, page 303, shows an elusive example of EBCDIC to ASCII conversion for 78 graphic characters common to ASCII, EBCDIC, and the Hollerith Punched Card Code). See also the bibliography in the separate writeup on "EBCDIC."

13. Comments: EBCDIC is defined by IBM Corporate Systems Standard 3-3220-002-CSS. The latest edition which IBM will make available to NBS is dated 70/11. Some EBCDIC control and graphic characters are not contained in ASCII. However, IBM chooses to map these, via the Hollerith Punched Card Code, into ASCII character positions, rather than into the 128 available non-ASCII character positions. EBCDIC equivalent characters to those displaced are mapped elsewhere, and the correspondence is thus spoiled. Selected examples are shown in the following chart:

CONTROL AND GRAPHIC CHARACTERS OF IBM EBCDIC WHICH MAP VIA HOLLERITH HOLE PATTERNS INTO 8-BIT ASCII IN POSSIBLY CONFUSING WAYS

IBM Name of Character	IBM EBCDIC Character	IBM EBCDIC Position Hex. Col. Row	Standard Hollerith Hole Pattern	Corresponding ASCII Position Column/Row	ASCII Symbol	ASCII Name
Interchange File Separator	IFS	1C	11-9-8-4	01/12	FS	File Separator
Field Separator	FS	22	0-9-2	08/2	None	None
Tape Mark	TM	13	11-9-3	01/3	DC3	Device Control 3
Cent Sign	¢	4A	12-8-2	05/11	[	Opening Bracket
Open Square Bracket	[	AD	11-0-8-5	13/5	None	None
Close Square Bracket	]	BD	12-11-0-8-5	14/5	None	None
Exclamation Point	!	5A	12-8-2	05/13	]	Closing Bracket
Logical Or	|	4F	12-8-7	02/1	!	Exclamation Point
Logical Not	¬	5F	11-8-7	05/14	^	Circumflex

There are four interchange separators in EBCDIC which correspond to the four information separators of ASCII. Only IFS is shown because of the possible confusion between EBCDIC FS (Field Separator) and ASCII FS (File Separator). The EBCDIC square brackets are not shown in the principal defining table (Table IV of the CSS) but are shown as publishing and printing graphic options (in Table VII).

There is no Cent Sign in ASCII. IBM has chosen to displace the ASCII Opening Bracket by the EBCDIC Cent Sign. However, in representing the ISO 7-Bit Code of ISO 646-1973, IBM drops the Cent Sign and uses the Hollerith hole pattern 12-8-2 to represent the ISO Left Square Bracket, as shown in Table X of the IBM CSS, as a displacement of a national use symbol.

The substitutions in ASCII of the symbols for Logical Or and Logical Not are permitted in the ASCII standard (FIPS 1/ANSI X3.4-1968). This was done by IBM in order to get all 60 of the PL/I language symbols into the 64-character subset of FIPS PUB 15. The ASCII Exclamation Point is displaced. The EBCDIC Exclamation Point then displaces an ASCII Bracket, and a ripple of confusion follows, as shown in the table above.

STANDARDS FOR COMPUTING SYSTEMS

PROGRAMMING LANGUAGE STANDARDS

INTRODUCTION

Programming languages serve the same purposes for computing as spoken languages do for human communications. They are the principal mechanisms by which ideas (algorithms), data, commands, response requirements, etc. are communicated from man to machine.

Like spoken languages, they have a tendency to diverge into "dialects," in which case users of different forms of the language find it difficult or impossible to continue communicating with each other. A cooperative standardization effort is frequently required in order to get the various dialects to converge acceptably, since the language compilers can not adapt to slight variations in use as can humans.

Programming language variations are inevitable and in many instances they are desirable, because through them better or entirely new forms of useful expression arise. The "better" forms are perhaps the more dangerous from a communication point of view because, if adopted, they must either supersede the older forms or introduce a redundancy into the language; in either case, considerable attention must be accorded these types of changes, as they constitute deviations from the approved language definition and threaten software portability.

New forms, or "unilateral extensions," are usually outside of the previously defined scope of the language and require some time to be defined, implemented, tested, understood by others and accepted into the language. As a result, they do not pose as immediate a threat to program portability as do the "better" forms. However, consideration must be given to the manner in which new forms are defined and employed in the building of application programs, so that users will be aware that the use of these new forms prevents them from creating portable code, at least until such time as the new form is accepted into the language definition.

STANDARDS ON EXISTING LANGUAGES

There are currently standards in existence or in the process of approval for four general purpose programming languages: FORTRAN, COBOL, PL/I, and BASIC. Of these languages, only PL/I is a "modern" language that has the potential for satisfying the requirements of the Air Force for a general purpose programming language for the CAM program. The choice of a general purpose programming language is not clear as will be shown. In fact, the language chosen by the Air Force for the ICAM program may not be any of these four discussed, although support for at least COBOL and FORTRAN is mandatory for the near future because of the body of existing programs in these languages.

Although ALGOL is mentioned several times in the following text, there is no formal standard and or standards committee for ALGOL and it is considered in terms of historical interest and the heritage it has brought to other languages. Current use of ALGOL is sufficiently limited that it is not considered comparable, even as a de facto standard, to the other languages discussed here.

Independent of which language standard is selected the Air Force must realize that the simple specification of a standard language in a procurement action will not be sufficient. Indeed an entire set of software development and documentation guidelines and validation and testing tools are mandatory to meet Air Force goals, as will be discussed below.

Only general purpose programming languages are addressed here. For specific

problem oriented needs, such as simulation or artificial intelligence, there are languages but no standards or defacto standards. It is believed that existing languages and compilers can be selected to satisfy ICAM project requirements when they are set.

FORTRAN

Originally designed in the early 1950's as a replacement for assembly code, FORTRAN is a simple higher level language that is easy to compile into machine code. However, the requirements of this efficiency have extracted a price which is paid for in annoying restrictions which crop up in use of the language. FORTRAN statements tend to reflect the hardware characteristics of the first machine to support FORTRAN. The memorable fact that every DO is always done at least once is an example. This is due to the fact that the original machine for the language has a test-and-jump instruction which worked by testing at the end of loops, rather that at their entry points.

FORTRAN lacks many features often expected of general purpose languages. Part of the omission is simply because the language is so old, about twentyfive years. Nonetheless, a user of ANS (or Standard) FORTRAN can not expect the following: good string handling; block structure; run-time allocation of space. FORTRAN's virtue is that it is simple and effective, and much preferable to assembly code; this point is important, because for many uses the competition is not other modern languages such as PL/I and PASCAL, but rather, machine code. FORTRAN in conjunction with an optimizing compiler can be very fast.

Elaborate libraries exist of FORTRAN engineering and scientific routines. In addition, techniques are available [Larmouth, 1976] which can stretch the design features a bit to circumvent the more annoying restrictions such as space allocation of arrays. Unlike BASIC, FORTRAN is sufficiently rich in coherent control structures that it can be sensibly used for large-scale implementations. The language is well suited to industrial applications which involve complex numerical calculations such as table interpolations, function integrations, or measurement smoothings and averaging. This is in contrast to COBOL which is rather inefficient and clumsy to use for scientific or engineering evaluations. (FORTRAN is not suited, on the other hand, for generating in a straightforward manner nicely formatted reports of a commercial nature.) FORTRAN input/output is both limited and slow. But as an interim scientific and engineering language for CAM, FORTRAN could serve nicely.

The new FORTRAN Standard, long in gestation, was released in draft form in early 1976. There was an avalanche of criticism--mostly that it should contain each critic's favorite structure--but it appears that debate will be cut off with the addition of IF-ELSE IF--END IF and perhaps STREAM input/output. Committee members hope to have solidified a new Standard by March 1977.

COBOL

COBOL was originally conceived as a business language for commercial data processing. It is an effective means for programming applications that are characterized by the requirement to manipulate characters, records, files and input/output (as contrasted with those concerned primarily with computational problem solving). Quoting Pratt, "COBOL is perhaps the most widely implemented of the languages...[See Pratt's book for the context of this.]... but few of its design concepts have had a significant influence on later languages, with the exception of PL/I. Both of these facts may be partially

attributed to its orientation toward business data processing, a major area of computer application, but one in which the problems are of a somewhat unique character: relatively simple algorithms coupled with high-volume input-output..."[Pratt, 1975,p. 359]

Like some other language of the same period, COBOL was developed and has been maintained by voluntary efforts of implementors and users. The COBOL standard, as is the case with any standard, does not in itself cure all problems associated with computer systems. As the language is used, its flaws and inadequacies become more apparent; action must be taken to correct, adjust and extend the standard definition.

There exists a rather elaborate mechanism dedicated to the continuing process of making COBOL evolve in response to user requirements. In addition, to enhance the viability of Standard COBOL as a tool, ancillary activities have been initiated to provide for testing of compilers for conformance to the standard, for interpretation of the language specification when questions of meaning arise and for development and establishment of policies relative to procurement and testing of COBOL compilers.

In a recent survey (NBSIR 76-1100), of the 132 Federal government computer installations responding to the survey question concerning usage of COBOL, 86.4% indicated that COBOL was available and 94.7% of those who had access to the language actually used COBOL to some extent. (Also see Phillippakis [1973].) A few examples of COBOL applications illustrate potential uses of COBOL within a CAM system. For example:

a. The National Weather Service, an agency of the Department of Commerce, has an operational on-line system providing weather forecast information. Approximately 30 terminals throughout the nation receive and send weather forecast information. Among the users are civilian and military agencies and radio and TV stations. The system is written in various languages; however, three to four dozen COBOL programs accomplish an important function in the system. These COBOL programs perform editing of input data for errors and formatting the data for its presentation over the network. COBOL was selected for use in implementing these programs because of its ability to handle editing and character manipulation.

b. The Defense Supply Agency (DSA), an agency of the Department of Defense, performs central supply service to all Defense agencies. It provides support materiel such as food, medical supplies, clothing and construction material. DSA has a very large logistics system called SAMMS (Standard Automated Materiel Management System) written in COBOL. SAMMS provides the following daily functions for DSA: distribution, requirements forecasting, financial management, procurement, and cataloging. This system is used in each of DSA's five major centers: Richmond, Virginia; Columbus, Ohio; Dayton, Ohio and two in Philadelphia, Pennsylvania. There are 400 to 500 individual reports produced by SAMMS. Examples of some of the reports are management reports, statistical reports, rejection reports, exception reports, and turn-around (time requirement) reports. The system requires about 1000 changes per year, mostly enchancements, because of changing requirements. The number of records in the system varies in each center from 800,000 to 1,500,000; approximately 12,000 records are updated per hour. With some 800 to 1000 COBOL programs, SAMMS is the largest logistics data system in the Federal government and is integral to DSA's daily operations. COBOL was chosen for implementing this system to enhance the portability of the programs and because of the attributes of COBOL for handling character data.

It is evident from the efforts pursuing development and standardization of COBOL and from the examples of how COBOL can be used effectively in its

typical application areas, that COBOL should be given serious consideration for use whenever a problem requires straightforward and few computations on numbers of limited range, along with a heavy volume of such numbers with special commercial formatting requirements (such as flush dollar signs).

BASIC

BASIC (Beginners All-Purpose Symbolic Instruction Code) is a computer programming language developed in the mid 1960's by Kemeny and Kurtz at Dartmouth College. The primary motivation behind the design of the language was the desire to educate large numbers of undergraduates in the use of a remote-console, time-shared computer to solve simple problems. Limitations on names to letter+numeral have been retained, even though the original purpose of this--to simplify symbol table maintenance--is no longer necessary. The language was designed to be learned and used easily, especially for simple problems. In this respect, the design criterion for BASIC differed from other languages such as FORTRAN (execution speed) or Algol 60 (expressing algorithms) which aimed at professional programmers.

Since its original design, more advanced features have been added to BASIC, both at Dartmouth and at other installations, so that BASIC now is often used as an alternative to FORTRAN or Algol 60. The divergence in the design of advanced features, in addition to divergence even in the features of original BASIC, has been a concern among suppliers and users of BASIC. In response to this concern, ANSI established an ad hoc committee "to investigate the computer programming languages generally known as BASIC, and determine the existence of a viable nucleus language suitable for standardization."

This committee recommended that ANSI create a technical committee charged with developing a standard for BASIC. This recommendation was approved by the committee at its January 17, 1973 meeting.

In addition to identifying and standardizing a nucleus for the BASIC language, the ANSI standards committee X3J2 is investigating advanced features in various implementations and is standardizing other modules as it sees fit. The standards committee felt it preferable to standardize more than just a BASIC nucleus, since the greatest divergence in BASIC implementations occurred in the treatment of features that would most likely not be in the nucleus.

The now proposed standard for MINIMAL BASIC reflects the original design of the language for use in a remote-console, time-shared system, though of course the standard does not preclude the use of BASIC in other modes of operation. In practical terms, the standard does assume that BASIC will be implemented in an environment which provides minor editing services and which emphasizes single-pass, fast compilation and execution rather than compilation of optimized code. Educational uses of programming languages require just such an emphasis.

The proposed American National Standard for Minimal BASIC was approved by X3J2 in January, 1976 and was forwarded to X3 for action. X3 has given the proposed standard the reference BSR X3.60 and has submitted the proposed Minimal Standard for public review. Comments were due by the end of September 1976. This nucleus standard contains those portions of the planned language not specifically contained in planned enhancement modules. Standards for enhancement modules concerning files, strings, matrices, subprograms and chaining, and formatted input/output are under development at present.

From the past experience of the development of the Minimal Standard the completion of the enhancements standardization may take on the order of 2 to 3 additional years. Therefore, a full BASIC standard should be available by 1980 provided the committee does not face any voting deadlocks. These estimates assume the regular committee meeting schedule of 4 meetings a year in the last week of each of the months of January, April, July, and October.

For any large scale effort in CAM systems being able to store, retrieve, and manipulate large sets of data is important. Minimal BASIC is not designed for this although future enhancements will allow file and formatted I/O capabilities. Large scale data handling can be accomplished in a more prominent language such as COBOL or PL/I, and with more probable efficiency.

There was an attempt to specify some minimal working precision for numerical constants and variables of at least 6 digits. However, no accuracy specification is imposed on arithmetic expressions or intrinsic functions. This limitation on precision makes BASIC unusable for some engineering calculations.

From the engineering design point of view many good algorithms for matrix manipulation, solving differential equations, etc. have already been coded and tested and installed through library packages such as IMSL (International Mathematical and Statistical Libraries, Inc.) or the Association for Computing's Collected Algorithms. FORTRAN and Algol 60 are the principal languages used for these existing collections.

The standard allows minimal string capability. No comparison except equal or not equal is allowed between strings. This is another disadvantage to the BASIC standard in its present form.

CAM should rely in the short run at least on languages and design support libraries that have the most wide spread use. Although BASIC has recently become popular the original intent of the language was for the learner to step on to another language; BASIC was not intended as a large scale production language. (The reader may want to reference Pratt, pp.475-476 for a lucid discussion of the demands of an interactive language, in his case APL, and possible detriments to doing large production programming.) Because of this fact, and because of the limitation in the BASIC standard, BASIC is not recommended for use in the ICAM program.

PL/I

PL/I is an extensive, general purpose language which was designed originally to enhance the IBM model 360 series of machines. Statements in the language are FORTRAN-like. In fact, the similarity is close enough that long time FORTRAN programmers are prone to lapse back into FORTRAN when writing some PL/I statements, such as the DO. (The meaning of a FORTRAN DO is not the same in PL/I!)

PL/I program structures are borrowed from ALGOL (e.g., BEGIN-END). Data types in the language show a clear COBOL influence. The language was meant to be a "universal" or omnibus for scientific, commercial and diversified general users. Whether it has met this intent is somewhat open to question. Use of PL/I has not grown nearly as fast as had FORTRAN or COBOL in an earlier period. Because of the very size of the language, the initial

compilers were difficult to write and slow in execution. This problem has been remedied somewhat with time.

The draft standard represents an attempt to simplify some features of an otherwise very large language. Because of the tentative nature of the draft, it is unlikely that any PL/I processor chosen today would conform to the letter of the new standard. Several PL/I dialects exist, including PL/C [Conway 1973], a student subset with fast compiling, and a systems support version PL/S. PL/I is not limited to IBM implementation even for system work. For example, 95% of Honeywell MULTICS is written in PL/I.

PL/I was accompanied soon after its introduction by a formidable formal model-- the Vienna Definition Language. This meta language, known also as VDL, has been retained in the draft of the standard. The modeling language is not very easy to read, and it remains to be seen whether use of it has removed the threat of ambiguities or omissions in the standard.

Besides the difficult VDL formalism, the PL/I standard has another drawback of not defining allowed subsets of the language. Implementation of the full capabilities of the language therefore requires a compiler that can only be run on large scale computers. Subsets of PL/I have been implemented for developing cross software for microcomputers (PL/M, PL/M68Ø0). More extensive standard subsets could be defined for minicomputers and medium scale computers.

If PL/I is used, the Air Force should specify standard subsets of PL/I for various applications within the context of the ICAM program.

Among languages mentioned in this report, PL/I is one that has potential in the long run as a good growth language for both systems and applications. The dialect PL/S[see below] is used by IBM on some of their systems work; student dialects have been mentioned above. Because it borrows from Algol for block structures, it is fairly easy to write "structured programs" in PL/I; in addition, the COBOL heritage provides a more definite input/output capability than that of, say FORTRAN, or (worst) Algol (where i/o is left undefined). Consideration of PL/I, along with other modern languages such as PASCAL and its extenions (e.g. EUCLID), should be made for longer-term planning in the CAM project. The ANS PL/I with specified subsets and with features of PL/S might be, for example, a good vehicle to write most of the CAM systems software.

SYSTEMS IMPLEMENTATION LANGUAGES

The development of large system software projects, e.g. operating systems, compilers, and data management systems, has been, and still is, hampered by the lack of adequate tools. The most important of these tools is a good high level systems implementation language (SIL). (The term systems programming language can be used interchangably.)

Despite the lack of a SIL that can be considered to be really good, the use of existing SILS is preferable to the implementation of system software in assembly language or macro-assembly language. If the resulting compiled code fails to meet execution time constraints, critical inner loops can be recoded in assembly language. If practical, they should not be placed in-line, but rather grouped together in a seperate module (or modules) and referenced

through procedure calls. This will isolate machine dependent code to enhance portability.

Of the existing SILS, there does not currently exist one that possesses a clear advantage over all others. Some notable attempts have been made in SIL design and implementation, but the resulting languages have nearly always been targeted to a single vendor's machine architecture or have not achieved widespead use. As mentioned above, a dialect of PL/I known as PL/S has been used internally by IBM to implement much of their system software. A dialect of Algol has been used by Burroughs in the same manner. Many other systems implementation languages have been developed but have not seen widespread use because of machine architecture dependencies. Several SILs have been designed specifically for microprocessors. There is obviously little incentive for a vendor to develop a systems implentation language that could be readily used to implement systems for another vendor's machines. Thus, if there is to be any movement to more machine independent systems implementation languages, that movement must come from without the mainframe vendors. The Air Force could provide that impetus.

It may be possible to avoid developing a special SIL. For example, 95% of Honeywell's MULTICS is reported to be written in PL/I, the rest in assembly language. The key features of a SIL are the ability to manipulate data at the physical level, rather than at the logical level, and to execute privaleged calls to the hardware. Implementation of a good data base management system and adequately standardized general purpose programming languages may be sufficient for Air Force needs in providing data manipulation and portable software.

The specification and initial design of a machine independent systems implementation language (DOD-1) is currently underway in the Department of Defense for use in system programming of weapons systems [Fisher, 1976]. Although its use is not mandated for general purpose, commercial computer systems, it may prove to be a good choice [DOD, 1976].

In summary, the Air Force must have a SIL. The choice is to pick one or develop one. There is no clear choice between the SILS that exist and have been implemented. With the possible exception of PL/S, a proprietary dialect of PL/I, the potential availability of PL/S should be investigated by the Air Force. Development of a adequate SIL may be the only choice; in this regard, the DOD-1 language effort should be carefully evaluated before an independent development effort is begun under the ICAM program.

FUTURE NEEDS IN PROGRAMMING LANGUAGES

General observations

C.A.R. Hoare[1973] has stated that a programming language should aid in program design, program documentation, and program debugging. He goes on to stress language simplicity, security, fast translation, efficient object code, and readability. (His paper also includes a very interesting annotated bibliography on some common languages, including FORTRAN, ALGOL 60, and COBOL.)

Documentation can be helped by syntactic forms in a programming language, or equally, hindered. Indeed, something as simple as a comment can be more (or less) useful in encouraging clear programs. Scowen and Wichmann[1974] review a number of comment conventions, including those in PL/I, ALGOL 60, FORTRAN, BASIC, and COBOL. They provide six design criteria for comments.

Program debugging occupies a sizeable portion of a programmmer's time and language features can be important. For example, data types in a language can help prevent improper transformations between disparate entities. However, a data-typing feature is defeated by an automatic , transparent type conversion (a la older PL/I), which may then require extremely tedious examinations of identifiers for improper type. Unchecked array bounds provide another very common source of error that can be difficult to catch without help from a compiler.

Although structured programming and related methods have met resistance in the programming community, the ideas are nonetheless attractive [Lucas, 1976; Yourdon, 1976-Chap.4]. Perhaps the situation would be different if programs were physical things which could be viewed for balance and workmanship [Cheatham, 1971]. Programmers may argue for complete latitude in connecting pieces of programs together; however imagine a carpenter who set wall studs sometimes at 16" apart, sometimes 14", and if his lumber was warped, at varying distances. He could argue that his buildings were no weaker than anyone else's, but the insulation workers would rate his handiwork less favorably, since standard batts are 16" wide.

The possibilities for connecting N points of a program are of order N*N. If nothing else the various structuring and programming refinement disciplines seek to introduce some constraint upon this potentially huge N*N. The most notorious restriction has been, of course, E. Dijkstra's condemnation of GOTO's [Dijkstra, 1968]. His point --quite valid--was that GOTOs represented a way of thinking about programming, that many GOTOs indicated shoddy program organization--a "Rube Goldberg" programmer in action. It was not enough that a program worked--so did most of Goldberg's bizarre inventions. The programming task should be thought through as one might organize an essay.

Yet even after the organization of a program has been expressed, it must be written in terms of some programming language. While an organization may reflect the virtues of modular pieces and good, tree-like dependencies among modules, it is equally clear that some languages will not allow one to ban GOTOs easily. COBOL and FORTRAN have control statements dependent upon GOTOs; for example, in COBOL the EXIT statement is, effectively, only an exit label at the end of the scope of a PERFORM; interior "exits" must GO TO this one valid point of egress. Any interior EXITS are treated as no-ops, and do not affect the PERFORM. And since FORTRAN has no compound statements, GOTOs are often introduced to produce the effect. More modern programming languages often include compound statements, conditionals, a DO or FOR statment, WHILEs, the CASE statement, and naturally, procedures including recursive ones.

A second place for a program to become unbuttoned is in its data; Hoare[1973] observes that untyped pointers allow as much arbitrary hazard in the data space of a program as GOTOs pose in the program (or control) space. A pointer can jump around, and if assigned an improper value, jump around into the wrong data locations. On an even simplier vein, it is possible to replace GOTOs by flags, only to find that the flags are so poorly designed that their meaning is dependent upon points of control in the program.

The moral is, if a programmer is messy, nothing will help.

Standards and Limitations

Any discussion on standardized languages and their status could be deceptive if unaccompanied by a caveat on the limitations of the language standards themselves, for in fact there are many system influences on language use, and in the wordings of the standards.

Larmouth[1976] provides details of many loose ends in FORTRAN. For example, local variables in a subroutine can become undefined (of indeterminate value) upon exit from the subroutine, even though most systems preserve local variables, treating them as Algol OWN values or PL/I STATIC. The reason for the Standard's hedging is that on stack-machines, such as Burroughs, the subroutine exit pops the storage stack. Local values are truly lost. On another plane, the recent problem with COMPUTE in COBOL was caused by a failure in the standard to define intermediate results for arithmetic expression evaluation. Some manufacturers used their machines' double precision floating point for the intermediate results, while others incorporated the various numbers of fixed point digits. It is impossible to state concisely all of the problems that one might encounter in a particular standard. The best advice would seem to be to refer the reader to the Larmouth article and indicate that the FORTRAN standard that generated all that discussion is about a tenth that of, say COBOL or PL/I Standards. Caveat emptor.

Files and the handling of system secondary storage exemplify the importance of uniform, simple conventions, especially among programs written in different languages. The dictum of "delayed binding", i.e. late fastening of attributes, implies that files should have no specific characteristics other than those absolutely necessary. This allows flexibility in rerouting inputs and outputs, typical requests for contemporary users. Usually there will be loadable files and text. Nothing else. Text, if sent to the printer process, generates--on paper-- a user's print file. It is not difficult to cite systems in which there are user card files, printer files, data files, and program source code files. For example, on the UNIVAC 1108 under EXEC II, it is quite easy to find that one has a COBOL preprocessor, written in COBOL to convert other COBOL decks, whose output is unreadable by the COBOL compiler! Gerhard Goos [1974] has remarked that:

"The most serious problem of today's system programming languages is the non-existence of a basic model for file-handling and I/O. All models either are developed with a certain operating system in mind and are difficult to adapt to other operating systems. Or they are too simple, allowing for sequential files only while random-devices are modeled by unstructured linear address spaces."

Much as one would like programs written in various languages to share files, one would also like to share library routines. K.W. Morton [1974] discusses the NAG library and practical limits in current operating systems; e.g. to serve both FORTRAN and Algol users some routines have to be coded twice. Hoare [1973] also reflects on the point briefly, and is not generally in favor of shared routines.

In any event, while specification of a standard in a language will improve compatibilities, such standards may require additional constraints to be really useful. This is especially true if distinct programming languages are to share the processing of file information on the system.

RECOMMENDATIONS

1. CAM systems and application software packages should be developed only with high level programming languages, except for the very few instances where acceptable performance can only be achieved by resorting to assembly language for coding of critical algorithms. These cases should be carefully controlled and documented.

2. The Air Force should encourage the use of standardized programming languages. NBS believes their effective use to be the key to software portability.

3. ICAM may not wish to prohibit the use of nonstandard programming languages where the reasons for their selection by a contractor are fully documented and supported. In those cases where the Air Force allows the use of a nonstandard language, it should at the same time initiate a standardization effort to formalize the product definition, through a consensus opinion of users and suppliers, so that compilers can be implemented on other computers to effect portability.

4. Because of the bulk of existing application programs are written in FORTRAN and COBOL, these two languages must be supported for the near term future in the Air Force ICAM program. Eventual conversion of existing programs to a modern language should be planned for under the ICAM program. At the present time FORTRAN and COBOL are the only two general purpose programming languages that are considered to be immediately useful to the Air Force.

5. The Air Force should support the establishment of a Federal FORTRAN standard based upon revision of the ANSI standard, now in progress. Should ANSI fail to approve a revised standard in 1977, the Air Force should support in writing the NBS goal of adopting the next ANSI committee proposal as a Federal standard.

6. Of all the general purpose programming languages submitted for standardization, PL/I is the only one that can be considered a "modern" language suited for Air Force ICAM applications. However, PL/I compilers can produce inefficient code and tend to require a large run-time support system. Furthermore, not all of the major computer manufacturers offer PL/I. Hence, it cannot yet be considered a "standard" language suitable for Air Force use. If it is desired to use PL/I, substantial effort in standardization will be required and particular attention should be given to the definition of subsets to run on smaller computers and to the development of extensions for systems work.

7. The Air Force CAM authorities should monitor the DOD-1 project because it appears to have the broad base of support that could produce a standardized language suitable for CAM needs in the 1980's. Among the candidates being considered in the DOD-1 effort that are particularly relevant to CAM projects are PASCAL and PL/1.

REFERENCES

Andreas S, Phillippakis. "Programming language usage."
ANSI BSR X3.53 Basis/1-12 1975(Feb.) Draft Proposed Standard Programming Language.

BSR X3.53 Chap 1 revised 1976(Feb).

BSR X3.53 Errata Sheet 1976(Jan.)

BSR X3.60

M. Beckmann, et. al. ADVANCED COURSE IN SOFTWARE ENGINEERING. Springer-Verlag (Berlin, 1974).

Bennet P. Leintz, "A Comparative Evaluation of Versions of BASIC," Comm. of the ACM, April 1976, Vol. 19, No. 4, pp. 175-188.

T.E. Cheatham. "The recent evolution of programming languages," Proceedings, IFIP Congress 71, Ljubljana, Yugoslavia, August 1971, pp I-118 -- I-134.

R. Conway and T . Wilcox. "Design and implementation of a diagnostic compiler for PL/I." Comm. ACM. 16, 3(March 1973), 169-179.

Donald R. Deutsch. Appraisal of Federal Government Cobol Standards and Software Management: Survey Results. NBSIR 76-1100, Final Report August 1976, U.S. Dept. of Commerce, National Bureau of Standards. DATAMATION, October 1973, 109-111.

E. Dijkstra. "GO TO statement considered harmful," Letter to editor of COMM. ACM 11, 3(March 1968), pp 147-148.

DoD Directive 5000.29, Management of Computer Resources in Major Defense Systems, 1976(April).

D.A. Fisher, A Common Programming Language for the Department of Defense -- Background and Technical Requirements, 1976(June).

M. Griffiths. "Relationship between definition of implementation of a language," Lecture Notes, op cit.

G. Goos. "Programming languages as tool in writing system software," in Beckmann, et.al, op.cit.

C.A.R. Hoare. "Hints on prgramming language design." Computer Science Department, Stanford University, Dec 1973, STAN-CS-73-403, 29 pp.

D.E. Knuth. "Structured programming with GO TO statments," COMPUTING SURVEYS 6, 4(December 1974), 263-301. (Special issue on programming.)

J. Larmouth. "Serious FORTRAN--The Rules of the Game." Chapter to appear as an NBS Tech Note for the NBS/NSF SOFTWARE ENGINEERING HANDBOOK, July 1976, 20pp. (An earlier version appeared in SOFTWARE--PRACTICE & EXPERIENCE)

Larmouth, J. "Serious FORTRAN," and "Serious FORTRAN-- Part Two." SOFTWARE: Practice & Experience 3, 2-3(1973), pp. 87-108, 197-225. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975.

H. Lucas Jr. and R.B. Kaplan. "Structured programming experiment," COMPUTER JOURNAL 19, 2(1976), pp.136-138.

K.W. Morton. "What the software engineer can do for the computer user," in Beckmann, et.al., op. cit.

Pratt, T.W. Programming Languages: Design and Implementation.

R.S. Scowen and B.A. Wichmann. "The definition of comments in programming languages," SOFTWARE--PRACTICE & Also see J.G.P. Barnes, op.cit.,pp.401-408.

N. Wirth. "On the composition of well-structured programs," COMPUTING SURVEYS 6, 4(December 1974),pp.247-262. (Special issue on programming.)

D.B. Wortman, et.al. "Six PL/I Compilers," SOFTWARE: PRACTICE & EXPERIENCE 6, (1976), pp. 411-422.

E. Yourdon. TECHNIQUES OF PROGRAM STRUCTURE AND DESIGN. Prentice-Hall, Inc. (Englewood Cliffs, N.J., 1976), 364pp.

STANDARDS DATA SHEETS

The following Data Sheets summarize those standards which apply to Computer Programming Languages.

1. Designation: ANSI X3J2/76-01

2. Title: Proposed American National Standard for Minimal BASIC, January, 1976.

3. Maintenance Authority: ANSI X3J2

4. Scope: BASIC (Beginners All-purpose Symbolic Instruction Code) was orginally developed at Dartmouth College for use by nonprogrammers. It was designed for interactive use in program construction and debugging. The range of usage of BASIC has grown beyond the scopes of the originally intended audience. Usage has expanded in universities as well as industrial organizations. BASIC is, in general, an easy-to-learn language and can be applied to nonnumerical as well as numerical problems.

5. Relationship to Other Standards: ANSI X3.4-1968 American National Standard Code for Information Interchange (base, 128-character set); ANSI X3.42-1975 The Representation of Numerical Values in Character Strings for Information Interchange (proposed Minimal BASIC accommodates forms stipulated in X3.42).

6. Competitive Standards: X3J3 dpANS FORTRAN[1]

7. Standardization Status: Approved by mail ballot of X3J2 and transmitted to X3 for action on 12/31/75. No official designation given as yet. The status of the standard at X3 is unclear at this time, as well as the plans for publication for comment.

8. Implementation Status: Honeywell 6635, 6080, 437; Hewlett-Packard 2000F, 3000; IBM 370/168, 145, 158; CDC 3300, 6500; PDP 1070; XDS 940; UNIVAC 1108.

9. Known Manufacturing Uses: Graphics, Interactive Language Requirements

10. Known Sources of Information: Dr. Thomas E. Kurtz, Director, Computation Center, Dartmouth College (Chairman X3J2); Mr. I. Trotter Hardy, NBS, (301) 921-3491, (NBS voting member on X3J2); Dr. David E. Gilsinn, NBS, (301) 921-3491.

11. Probable Sources of Information: IBM, Sperry Rand, Hewlett-Packard, General Electric, Dartmouth College, Digital Equipment Corporation, Control Data Corporation

12. Bibliography:

ANSI X3J2/76-01, Proposed American National Standard for Minimal Basic, Jan. 1976
BASIC/3000, HP
Real-Time BASIC, HP
IBM BASIC for the 370
BASIC (BNF) Burroughs
BASIC, CDC
BASIC, Multics
Xerox BASIC
JPL BASIC
GE Mark III BASIC
DEC, "BASIC-Plus Languages Manual"
Bennet P. Leintz, "A Comparative Evaluation of Versions of BASIC," Comm. of the ACM, April 1976, Vol. 19, No. 4, pp. 175-188.

13. Comments: The proposed standard specifies a minimum of 6 digits of numeric representation for precision. There is, however, a possible infinite loop case in the FOR-NEXT statement and there is a new language element OPTION BASE to specify array lower bounds that has not been implemented on any system.

---

[1]The FORTRAN standard was identified as a possible conflicting standard in the sense that BASIC is a FORTRAN-like language. However, BASIC is more interactively oriented and minimizes format considerations. Both can be used to solve similar problems.

1. Designation: FIPS PUB 21-1, ANSI X3.23-1974, ISO 1789

2. Title: American National Standard COBOL

3. Maintenance Authority: Commerce Department, National Bureau of Standards for FIPS PUB 21-1; ANSI X3J4 for ANSI X3.23-1974.

4. Scope: Programming language for use in computer applications that emphasize the manipulation of characters, records, files and input/output (as contrasted with those primarily concerned with computational problem solving).

5. Relationship to Other Standards: FIPS PUB 44 - Standard COBOL Coding Form

6. Competitive Standards: None

7. Standardization Status: The documents cited represent the current revisions of the COBOL standard.

8. Implementation Status: Wide range of general purpose computers.

9. Known Manufacturing Uses: No known use for applications directly contributing to the manufacturing process; however, as stated under "Scope" above, is appropriate for predominantly data manipulation applications, usually in support of business management functions.

10. Known Sources of Information: Ms. Mabel Vickers, NBS, COBOL Project Manager, (301) 921-3491; Jitze Couperus, Chairman, ANSI X3J4, (408) 734-7499.

11. Probable Sources of Information: William Rinehuls, USAF, DoD Standards Coordinator, (202) 695-6547

12. Bibliography:

FIPS PUB 21-1, COBOL, December 1, 1975
ANSI X3.23-1974, COBOL, May 10, 1974
CODASYL COBOL Journal of Development, 1976, (current developments of COBOL)
FIPS PUB 44, Standard COBOL Coding Form, September 1, 1976

13. Comments: The COBOL standard is supported by a mechanism for the continued development and standardization of the languge as dictated by user needs and state-of-the-art developments in language use and implementation. The Federal standard is supported by Federal Property Management Regulation 101-32.1305-1 which specifies the procurement and compiler testing policies applicable to Federal agencies.

1. Designation: ANSI X3.9-1966

2. Title: American National Standard FORTRAN

3. Maintenance Authority: ANSI X3J3

4. Scope: Programming language for scientific and engineering applications.

5. Relationship to Other Standards: ANSI X3.10-1966, Basic FORTRAN (subset); ANSI X3.42-1975, The Representation of Numeric Values in Character Strings for Information Interchange, (the FORTRAN standard accommodates forms stipulated in X3.42).

6. Competitive Standards: PL/I

7. Standardization Status: ANSI standardization completed March 1966; revision thereto is now out for public review, comment and X3 ballot; action date January 1977; new designation to be X3.9-1977.

8. Implementation Status: All general purpose computers; most manufacturers are now updating their compilers to meet the proposed revision.

9. Known Manufacturing Uses: Scientific applications, numerical control, preprocessors and postprocessors. Most scientific and engineering application programs are coded in FORTRAN.

10. Known Sources of Information: Mrs. Francis E. Holberton, NBS, (301) 921-3491; Mr. William F. Hanrahan, Secretary, ANSI X3, (202) 466-2288.

11. Probable Sources of Information:

12. Bibliography: ANSI X3.9-1966 (Current) ANS FORTRAN
ANSI X3 BSR 3.9, March 1976 (same as X3J3/76) draft proposed ANS FORTRAN

13. Comments:

1. Designation: MDC/28, MDC/33, and MDC/34

2. Title: MUMPS Language Standard

3. Maintenance Authority: MUMPS Development Committee

4. Scope: Programming language for interactive data handling.

5. Relationship to Other Standards: FIPS PUB 1/ANSI X3.4-1968, ASCII (base, 128-character set); FIPS PUB 15-1971 (base, 64-character graphic subset)

6. Competitive Standards: None

7. Standardization Status: ANSI letter ballot mailed May 24, 1976; action date November 24, 1976; designation to be ANSI X11.1

8. Implementation Status: Standard implementations: Artronix PC-16, Burroughs B-6700, DEC PDP-10, DEC PDP-11, IBM 360/370, Philips P856/857; being implemented on machines of six additional manufacturers.

9. Known Manufacturing Uses: String handling applications, such as in inventory control and parts cataloging.

10. Known Sources of Information: Mr. J. T. O'Neill, NBS, (301) 921-3485, Jack Bowie, Sc.D., Chairman, MUMPS Development Committee, (617) 726-3937.

11. Probable Sources of Information:

12. Bibliography:

NBS Handbook 118, issued January 1976, with errata dated March 9, 1976, MUMPS Language Standard[1]
MDC/29, 5/28/75, MUMPS Interpreter Validation Program User Guide
MDC/30, 6/25/75, MUMPS Translation Methodology
MDC/35, 10/14/75, MUMPS Documentation Manual
MDC 1/11, 6/13/75, MUMPS Primer
MDC 2/1, 5/15/75, MUMPS Globals and Their Implementation
MDC 2/2, 5/30/75, Design of a Multiprogramming System for the MUMPS Language
MDC 2/3, 6/15/75, Implementation of the MUMPS Language Standard
MDC 3/5, 8/31/76, MUMPS Programmers' Reference Manual

13. Comments:

---

[1] NBS Handbook 118 consists of three MUMPS Development Committee documents, namely, Part I, MDC/28, MUMPS Language Specification, dated March 12, 1975; Part II, MDC/33, MUMPS Transition Diagrams, dated September 17, 1975; and Part III, MDC/34, MUMPS Portability Requirements, dated September 17, 1975).

1. Designation: None

2. Title: PASCAL

3. Maintenance Authority: Prof. N. Wirth, Institut fur Informatik, Clausiusstrasse 55, CH-8006 Zurich.

4. Scope: General purpose programming language.

5. Relationship to Other Standards: Designed to replace ALGOL 60. A commercially available process control language is a superset of PASCAL.

6. Competitive Standards: ALGOL-W

7. Standardization Status: The original PASCAL is the product of Prof. Wirth, so the standardization is fairly clear.

8. Implementation Status: Available on: DEC PDP-10, PDP-11; CDC 6000; CII IRIS 80, CII 10070; IBM 360/370; Univac 1108; XDS Sigma 7.

9. Known Manufacturing Uses: Scientific and engineering programming as well as some systems implementation programming.

10. Known Sources of Information: George H. Richmond, Editor, PASCAL Newletter, University of Colorado Computer Center, 3645 Marine Street Bolder, Colorado 80302

11. Probable Sources of Information:

12. Bibliography:

A. N. Habermann, "Critical comments on the programming language PASCAL," NTIS # PD-224 777, Oct. 1973, 22 pp.
C.A.R. and N. Wirth, "An axiomatic definition of the programming language PASCAL," 30 pp.
Kathleen Jensen and Niklaus Wirth, PASCAL: User Manual and Report, Lecture Notes in Computer Science 18, Springer-Verlag (New York, 1974), 169 pp.
also Geo. H. Richmond, Ed., PASCAL Newsletter, from 1974 onward.

13. Comments: Primary community of users is found in academic institutions.

1. Designation: ANSI BSR X3.53 BASIS/1-12, Feb. 1975.

2. Title: PL/I

3. Maintenance Authority: ANSI

4. Scope: General purpose omnibus programming language.

5. Relationship to Other Standards: None, except that language is generally thought of as a replacement for both FORTRAN and COBOL, and probably ALGOL too.

6. Competitive Standards: dpANS FORTRAN, dpANS COBOL

7. Standardization Status: BASIS/1-12 + errata sent to ECMA General Assembly for vote in Jan. 1976. Also sent to ANSI X3 for general processing.

8. Implementation Status: All major IBM systems. Dialects on Honeywell, CDC, and university installations (PL/C, etc.), including Amdahl machines

9. Known Manufacturing Uses: Business, scientific, engineering.

10. Known Sources of Information: IBM Corporation; General Electric; Honeywell Multics documents, Cornell PL/C user guides, etc.

11. Probable Sources of Information:

12. Bibliography: ANSI BSR X3.53 BASIS/1-12, Feb. 1975
BSR X3.53 Errata sheets, Jan. 1976
BSR X3.53 CHAP. 1, revised Feb. 1976
Many textbooks, e.g., W.W. Peterson

13. Comments: Standard is pending approval. PL/I is a very large and very powerful programming language. It was designed somewhat hurriedly and the design is therefore not the most elegant.

1. Designation: None

2. Title: Composite summary sheet on Simulation Languages[1]

3. Maintenance Authority: Various developers

4. Scope: Simulation languages can be divided into three clases, namely, continuous discrete, and hybrid. Continuous languages are for implementing models of systems having continuous dynamic change (i.e., sets of differential equations). Discrete, languages are for models showing discrete change (i.e., queuing and resource allocation). Hybrid languages combine both features into one package.

   The process of simulation involves:

   a. developing a system model expressed in mathematical, logical, or graphical notation,

   b. implementing the model in a computer using simulation language notation,

   c. validating the model to insure an acceptable degree of accuracy, and

   d. running the model to accrue experimental data.

   Simulation languages are usually of three programmatic types:

   1. routines for simulation embedded in a general source language such as FORTRAN,

   2. an entire higher order source language, or

   3. a data base-driven set of object code.

   In certain cases, simulation languages are employed in real-time to accept inputs sensed from some controlled system or process.

5. Relationship to Other Standards: N/A

6. Competitive Standards: N/A

7. Standardization Status: None

8. Implementation Status:

9. Known Manufacturing Uses: Control of processes; design of processes and facilities; resource allocation and planning.

10. Known Sources of Information: Mr. Paul F. Roth, NBS, (301) 921-3545.

11. Probable Sources of Information: Various developers

12. Bibliography: G. Gordon, System Simulation, Prentice Hall, 1969.

13. Comments: No standard summary sheets are included here because there are no standards in this area. Simulation languages are, in most instances, developed and/or supported by computer mainframe vendors or by software houses specializing in simulation. Simulation languages encompass such a wide variety of forms and uses that early voluntary standardization is unlikely; however, some de facto conventions might well be adopted for the Air Force ICAM effort.

---

[1] Languages tentatively identified for detailed consideration are CSSL, CSMP, GASP IV, GPSS, and SIMSCRIPT.

1. Designation: None

2. Title: Basic Language for Implementation of System Software (BLISS)

3. Maintenance Authority: Digital Equipment Corporation (for official versions); Carnegie-Mellon University, Dept. of Computer Science (for unofficial versions).

4. Scope: De facto system implementation programming language standard for the Digital Equipment Corp. PDP-10 (BLISS 10) and PDP-11 (BLISS-11)

5. Relationship to Other Standards: It is assumed that the source is in ASCII.

6. Competitive Standards: BCPL (and variations, particularly C for the PDP-11); SAIL

7. Standardization Status: The de facto standard for the language BLISS is embodied in the implementations of BLISS-10 and BLISS-11 by Digital Equipment Corp. To date, there has been no effort to standardize this language.

8. Implementation Status: The language was designed and first implemented for the PDP-10 at Carnegie-Mellon University. It was later adopted by Digital Equipment Corp. and has become a supported language under their standard PDP-10 operating system. The PDP-11 version was likewise designed and first implemented at Carnegie-Mellon. It is now available from Digital Equipment Corp. via a cross compiler, i.e., it executes on a PDP-10, producing code for a PDP-11.

9. Known Manufacturing Uses: None

10. Known Sources of Information: Digital Equipment Corp., Maynard, MA; Carnegie-Mellon University, Dept. of Computer Science, Pittsburgh, PA.

11. Probable Sources of Information:

12. Bibliography:

BLISS-10 Programmer's Reference Manual (DEC-10-LBRMA-A-D), Digital Equipment Corp., Maynard, MA;

W.A. Wulf et al., BLISS Reference Manual: A Basic Language for Implementation of System Software for PDP-10, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

13. Comments: Digital has implemented sophisticated FORTRAN compilers using both BLISS-10 and BLISS-11 as implementation languages.

1. Designation: None

2. Title: PL/S

3. Maintenance Authority: IBM Corp.

4. Scope: System implementation language used by IBM for the 360/370 series

5. Relationship to Other Standards:

6. Competitive Standards:

7. Standardization Status: The specification of PL/S has not been released by IBM.

8. Implementation Status: By implication, it can be assumed that IBM has implemented a compiler for PL/S, since much of the system software for the 370 series is written in PL/S. However, none of the source code is distributed, since IBM refuses to distribute the compiler.

9. Known Manufacturing Uses: None

10. Known Sources of Information: IBM Corp., Data Processing Division, 1133 Westchester Ave., White Plains, NY 10604.

11. Probable Sources of Information:

12. Bibliography:

Guide to PL/S II (Form GC28-6794-0), IBM Corp., Data Processing Division, White Plains, NY;
Guide to PL/S-Generated Listing (Form GC28-6786-0), IBM Corp., Data Processing Division, White Plains, NY;
G. Wiederhold and J. Ehrman, Inferred Syntax and Semantics of PL/S, in Proceedings of a SIGPLAN Symposium on Languages for Systems Implementation (published as SIGPLAN Notices, Volume 6, Number 9, Oct. 1971).

13. Comments: IBM appears to heavily use PL/S for its own internal system implementations. Unless IBM or someone else releases a PL/S compiler for general use, this language is of no utility to anyone but IBM.

1. Designation: None

2. Title: BCPL and C

3. Maintenance Authority: BCPL: installation-dependent; C: Bell Telephone Laboratories

4. Scope: BCPL (Basic Combined Programming Language) is a system implementation language. C is also a system implementation language developed as a significantly enhanced dialect of BCPL.

5. Relationship to Other Standards:

6. Competitive Standards:

7. Standardization Status: There has been no formal effort to standardize these languages.

8. Implementation Status: BCPL has been implemented on a wide variety of machines. The most important implementation of C has been for the DEC PDP-11. Other implementations exist for the IBM 360/370 and Honeywell 6000 series.

9. Known Manufacturing Uses:

10. Known Sources of Information: C: Dennis Ritchie, Bell Telephone Laboratories, Murray Hill, NJ 07974

11. Probable Sources of Information:

12. Bibliography:

M. Richards, The BCPL Reference Manual (Project MAC Memo M-352-1), M.I.T., Cambridge, MA (1968)
M. Richards, BCPL: A Tool for Compiler Writing and System Programming, Proceedings, Spring Joint Computer Conference (1969);
D. Ritchie, C Reference Manual, Bell Telephone Laboratories, Murray Hill, NJ.

13. Comments: Outside of a few user communities, BCPL has not been used heavily. The primary user community for C is that of PDP-11 UNIX users. The UNIX operating system is almost completely written in C, and C is the best supported and most heavily used language available on UNIX.

1. Designation: None

2. Title: PL/M, PL/M6800, and MPL

3. Maintenance Authority: Intel Corp. (PL/M for the Intel 8008 and 8080); Intermetrics (PL/M6800 for the Motorola 6800); Motorola Corp. (MPL for the Motorola 6800)

4. Scope: All three of these languages are high level system implementation languages for 8-bit microprocessors.

5. Relationship to Other Standards:

6. Competitive Standards:

7. Standardization Status: To date, none of these languages has been the subject of standardization. However, there has been some effort to make PL/M and PL/M6800 compatible at the source code level.

8. Implementation Status: Cross compilers exist for all three of these languages and are available through nationwide timesharing services or as FORTRAN programs designed to run on a user's IBM 360/370 system.

9. Known Manufacturing Uses:

10. Known Sources of Information:

11. Probable Sources of Information:

12. Bibliography:

8008 and 8080 PL/M Programming Manual - Revision A (MCS-451-0275-10K), Intel Corp., Santa Clara, CA (1975);
D. Fylstra and R. Gardner, PL/M6800 Language Specification (Report No. IR-161), Intermetrics Inc., Cambridge, MA (1975).

13. Comments: PL/M and PL/M6800 offer an almost completely compatible language for programming Intel 8080 and Motorola 6800 microprocessors. MPL for the Motorola 6800 was not designed to be compatible. However, all three of these languages are subset dialects of PL/I and therefore will have a high degree of similarity.

1. Designation: None

2. Title: Composite summary sheet on Artificial Intelligence (AI) Languages[1]

3. Maintenance Authority: Various developers

4. Scope: Novel features (new data types and control structures, pattern matching, deductive mechanisms, etc.) embedded in programming languages for robotics, automatic programming, and the representation of knowledge (see bibliographic citation 12. a. below).

5. Relationship to Other Standards: N/A

6. Competitive Standards: N/A

7. Standardization Status: None

8. Implementation Status:

9. Known Manufacturing Uses: ". . . heuristic programming, algebraic manipulation, . . . pattern recognition, . . . information retrieval, numerical computation" (see bibliographic citation 12. b. below).

10. Known Sources of Information: See bibliography.

11. Probable Sources of Information:

12. Bibliography:

a. Bobrow, Daniel G. and Bertram Raphael, New Programming Languages for Artificial Intelligence, Computing Surveys, Vol. 6, No. 3, September 1974 (with 31 bibliographic entries).
b. Abrahams, Paul W. et. al., The LISP 2 Programming Language and System, Proc. FJCC, Vol. 29, (1966).
c. SAIL User Manual, Stanford Artificial Intelligence Laboratory, Memo AIM-204, Computer Science Department Report STAN-CS-73-373, July 1973.
d. Sammet, Jean E., Programming Languages: History and Fundamentals, 1969.

13. Comments: "For more than a decade, the list processing and symbol-manipulation languages -- such as COMIT, IPL, LISP, SLIP (Bobrow Raphael 1964) -- have been the media for almost all AI achievements. Although the effectiveness of research with these languages has improved dramatically due primarily to greatly expanded memory sizes and new interactive debugging facilities, the languages have remained remarkably stable. In recent years, however, new directions for emphasis in AI research -- such as studies of representation of knowledge, robotics, and automatic programming -- have led to a widely felt need for certain rather novel features to be embedded into programming languages; and some languages containing several of these features have recently been implemented" (see bibliography citation 12. a. above).

---

[1] Languages tentatively identified for detailed consideration are SAIL, PLANNER/CONNIVER, QLISP/INTERLISP, POPLER/POP-2, and LISP 2.

STANDARDS FOR COMPUTER SYSTEMS

OPERATING SYSTEMS

INTRODUCTION

Operating systems can be thought of as the system managers. In response to demands of a user's program, the operating system manages the allocation and use of the central processor unit, main and mass memories, and input and output resources.

The lack of standards and quality in existing operating systems is the major problem in transporting software from one computer installation to another, even with only a single make and model of computer.

Operating systems are at once the best and worst place to consider standardization. Ideally, if one had a standard operating system, then one could imagine true software portability, since all machines would appear identical. From a practical point of view, a standard operating system for a large computer is neither practical nor desirable.

Operating systems for large computers are huge collections of software programs intimately related to the particular hardware architecture for which they were designed. For this reason, those features that are common among large computers, and could be the basis for standardization, are generally a very small subset of the total features implemented in a modern operating system. This lowest common denominator approach would deny the user the best features of the large computers in use today. Further, the mainframe manufacturers have a market incentive to keep operating systems both unique and proprietary.

The second problem for the Air Force in considering operating systems is their size and complexity: the cost of developing a new operating system for a large machine would probably exceed the total resources of the ICAM program. Worse, advances by the industry in hardware and system designs would soon obsolete whatever system was developed.

Incompatible features of operating systems will undoubtedly cause the Air Force serious problems in creating complex integrated systems software that is sufficently independent of the host computer to be portable. However, overall operating system standardization does not seem to be a viable answer. There are several areas in which limited standards can and should be implemented for the ICAM program which will be discussed below.

The situation is somewhat different for mini and microcomputers. The 16 bit minicomputers are sufficiently similar in their hardware characteristics and system architectures that the idea of a standard operating system is feasible. For a distributed, integrated system based on 16 bit minicomputers, the development of a communications oriented standard operating system is probably within the resources of the ICAM program. The 32 bit machines which are byte oriented (in handling internal data communications) are generally extensions of comparable 16 bit machines and could also be considered in developing a standard operating system.

Microcomputers are too small to have much of an operating system. Simple terminal monitors or switch monitors are supplied on ROMS in microcomputer kits to allow the user to load programs, but that plus some simple debugging routines is the extent of the system software. There is an opportunity to facilitate the use of microprocessors in CAM systems through the development of a cross software system based on PL/M or some other subset of a high level language that would run on higher level computers. Such a system would be essentially independent of the rapid hardware innovations at the microprocessor level and could provide full system support capabilities.

OPERATING SYSTEM FUNCTIONS

Historically, operating systems first arose as a matter of convenience rather than necessity. In the early 1950's, each programmer actually operated the machine and debugged his program on-line, controlling card input formats and line printer formats with patch panels inserted in the periphals. Batch processing programs were developed in the late 50's to expedite this situation by automatically loading another program as one was completed.

Executive systems were developed in the early 1960's that provided users with common access to complex programs developed for handling input and output. At this time computers were basically constrained to a single user and each job was completed before the next one began.

Because input and output functions depend on external periphal devices generally much slower than the CPU, single user systems are very inefficient. For this reason, multiprogramming batch systems were developed that allowed more than one job to be executed at once.

The development of time sharing systems, on line file management, real time operating systems, and virtual storage and virtual machine concepts has led to the operating systems of the 1970's, in which multiple users can simultaneously have access to the resources of the computer. The operating system is required to schedule the computer resources while preventing unwanted interaction between unrelated processes and to enforce access restrictions to data.

The primary functions of modern operating systems can roughly be divided into 4 classes:

1. Job control
   - job scheduling
   - process scheduling
   - control of information flow
   - start/stop processes

2. Main Storage management
   - allocate memory (including partitioning and/or paging)
   - access control

3. Device management
   - schedule I/O devices
   - control data flow to I/O devices
   - monitor interrupts on I/O devices

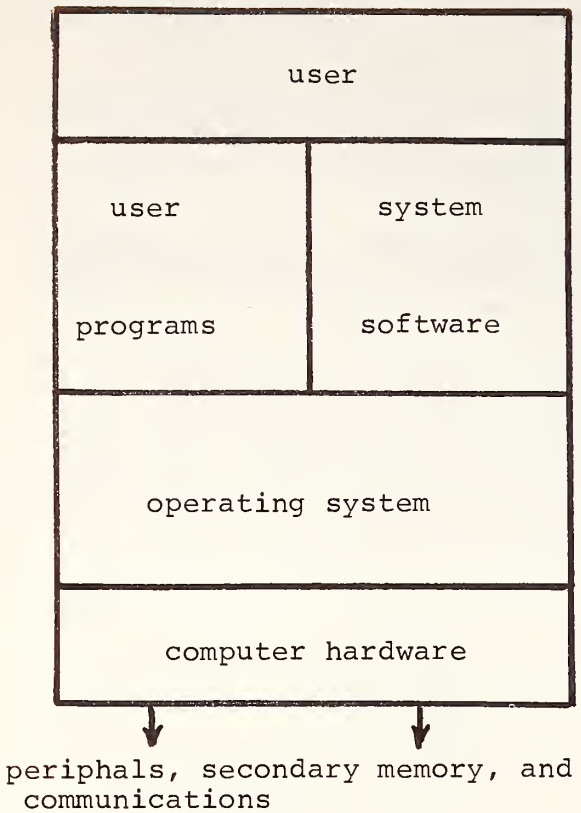
4. File system management
   - create/destroy file
   - open/close file
   - read/write file

It is in this last area of file management that many of the worst problems of software compatibility and portability arise, as we will discuss below.

COMMUNICATION WITH AN OPERATING SYSTEM

The user communicates with an operating system by two methods: system calls and an operating system command language (OSCL).

System calls can be thought of as procedure calls to special operating system procedures. They are used in programs to request services of the operating system. For example, READ and WRITE statements are supervisory



Functions

- specifies desired operation of computer
- user specified applications programs
- compilers, assemblers, loaders
- debugging, text editing
- libraries
- job scheduling and control
- storage management
- device management
- file system management
- main memory
- electronic data processing
- interaction of CPU with outside world

Figure 1

OPERATING SYSTEM FUNCTIONS

calls. The system calls represent the "primitive actions" that an operating system can perform for an executing process.

These primitives vary greatly between operating systems since they represent basic design decisions and implementation realizations. Standardization at the system call level is not practical nor advisable since it might stifle new innovation.

However, it is possible to present a more uniform view of the system call interface to a process by layering it with routines which map user intentions into system calls. This is, in fact, exactly what is done by the I/O runtime support routines for a programming language.

Figure 2 shows schematically how a user program interfaces to an operating system through a runtime support routine. These routines are necessary to translate the varying system calls in different languages to a form understood by the operating system. For example, OUTPUT FILEZ in BASIC and WRITE 600, FILEZ in FORTRAN may be translated into the same system call to initiate an I/O action.

It is at this level that the direct interaction takes place between a user program and an operating system for I/O.

An extension of this approach to other system feature calls may yield improved benefits and warrants investigation. However, any such approach is still limited by the basic primitives that the operating system designers implemented.

An operating system command language (such as JCL) is a self-contained but often rudimentary language for direct communication between a user and the operating system. The command language is used to schedule jobs, assign files, etc. and otherwise direct the execution of programs on the behalf of the user. The design of a command language is greatly influenced by the primary intended mode of operation of the operating system: batch or interactive. Unfortunately, there exist systems orginally designed for batch operation to which an interactive mode was later added. The resultant command languages are often ill-suited for interactive use.

Some attempts have been directed towards the development of a system-independent command lanugage. They have received very little, if any, vendor support and probably for that reason have had no success. However, on some of the more well-designed operating systems, the command language exists as a separable part of the system, and thus can be easily changed. If fact, some of these systems can support more than one command language.

Each vendor of operating systems has a unique appraoch to the implementation of the user-system interface from one generation to another. No operating system in widespread use can be said to possess sufficient redeeming qualities in its user-system interface that acceptance of it as even an ad hoc standard can be advocated.

VIRTUAL SYSTEMS

There are several concepts that can be considered under the general title of virtual systems. These include virtual memory, virtual devices, and virtual machines. All are intended to make a physical characteristic of the computer appear to be more than it actually is in order to help the user and improve the efficiency of utilization of the computer itself.

Virtual Memory: this is by now the well known concept of placing only parts (pages) of a users program or data files in the high speed main memory at any one time. By managing the partitioning of the main memory and by swapping appropriate pages to and from low speed, low cost,

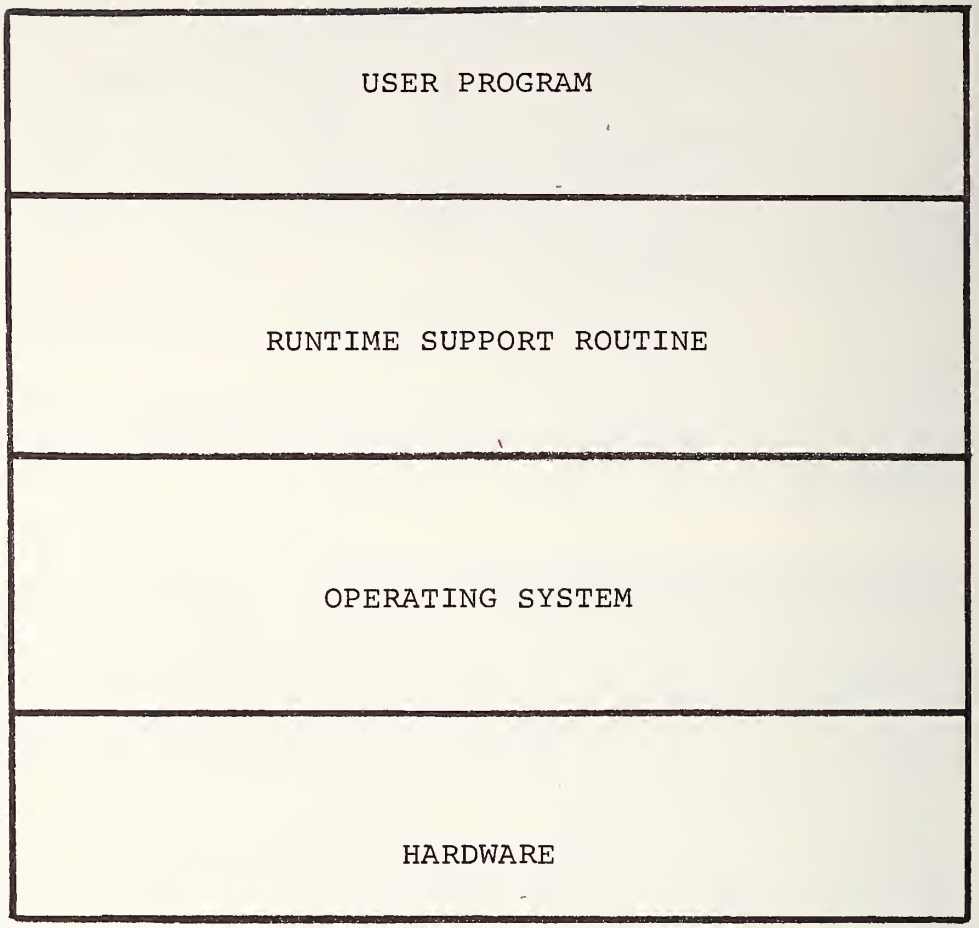


Figure 2

RUNTIME SUPPORT ROUTINE

high volume secondary disc storage, the user program sees a memory that appears to have the capacity of a disc with the speed of the main memory.

Virtual Devices: with multiprogramming systems, the limitations of communication to and from I/O devices can cause the system to bog down. This can be circumvented by creating duplicate, virtual devices. A program, then, will output to a virtual device. After a program is completed, the data file can be scheduled for output on the physical output device. Several different programs may be simultaneously performing I/O operations to the same (virtual) device.

Virtual Machines: the same essential problem exists with process management as with device management. By creating multiple, virtual versions of the operating system hardware interface, several operating systems can (seemingly) simultaneously execute privileged system calls at the hardware level. The virtual monitor, or hypervisor, is shown in Figure 3. The hypervisor operates on an interrupt basis in response to privileged instructions for the operating system. A file is set up of these instructions for execution when the hardware is actually available, and control is returned to each operating system in such a way that it thinks the instruction was executed. This can make one computer look like several computers with different operating systems.

The possibility of extending the virtual machine concept to gain hardware independence for Air Force software has been considered and discarded. The same arguments that were given at the first of this section still hold true:

1. The basic limits are the hardware features of the machine. Using only those features that are common to all large machines is inefficient and too severe a restriction.

2. Adding another layer of interpretation is inefficient.

3. The potential cost of operating systems development is huge and will be quickly rendered obsolete.

For these reasons, extensions of the virtual machine concept are not recommended.

FILE MANAGEMENT PROBLEMS

It is in this area that the Air Force can expect to encounter serious problems unless adequate care is taken in the early design stages. Different computers have different file management schemes which may cause problems in an integrated, distributed environment such as that envisionaged by the Air Force for ICAM.

File management can even be a problem in a single computer environment. For example, a file written by a FORTRAN program may be unreadable by a COBOL program because of the formatting and the addition of "invisible" bits such as file designations and check sums.

These problems can be solved by careful consideration and standardization of the file management system calls made by the runtime support routines for each of the languages to be allowed in the ICAM program. Changes can be made to these routines, if necessary, at low cost.

Standardization of file formats and naming conventions can and should be done for the ICAM program as special project standards. This will simplify the file compatibility problem and will help insure portability. This can be carried out in conjunction with development of the data base management system for the program.

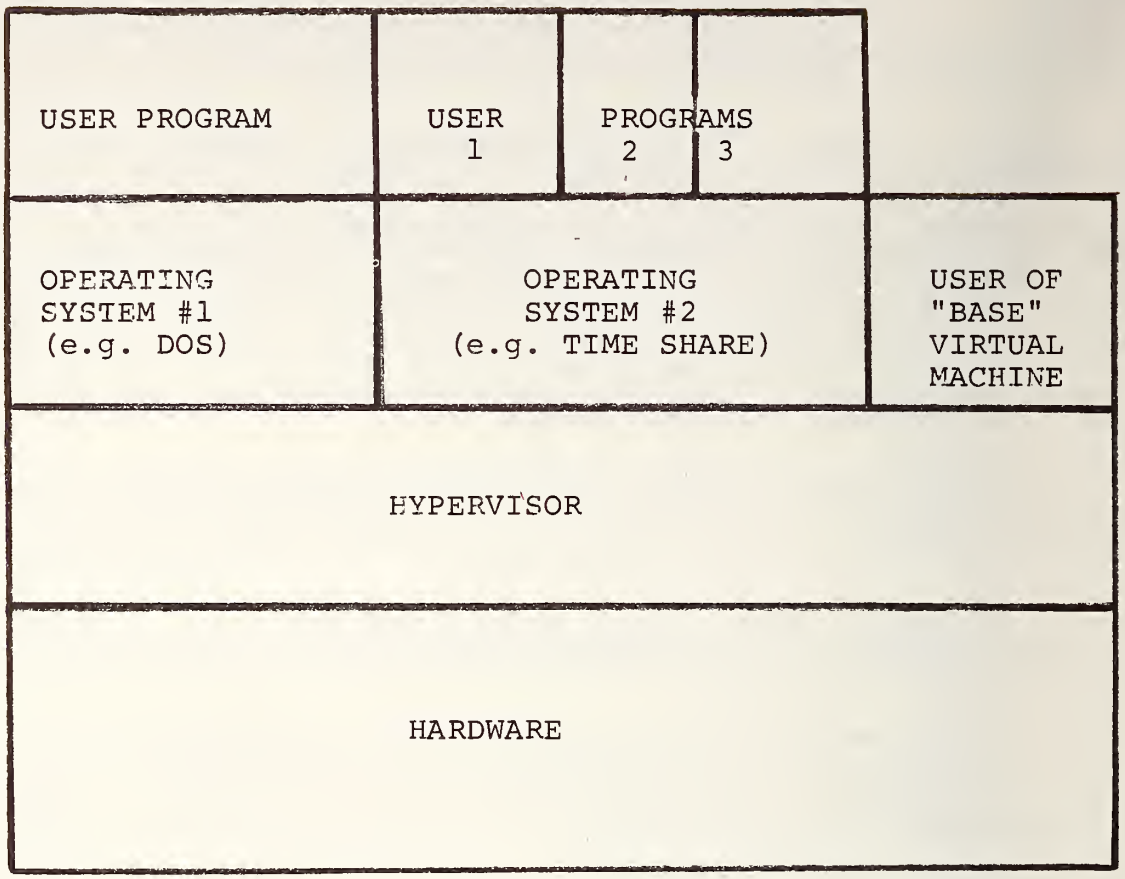


Figure 3

HYPERVISOR CONCEPT

Another problem is in the creation of files when reading from a magnetic tape. For example, the same problem of "invisible" bits mentioned above can occur here. As another example, if a 7 bit ASCII code is used on the tape, a 36 bit machine operating system may pack 5 1/7 characters into one word. This will produce an unreadable file.

Again, using an example from the field of automatic image pattern recognition, in loading digitized image data into a 60 bit word computer, the file management software may pack 7 1/2 8 bit bytes into each word. Since one 8 bit byte is a discrete information element (pixel), further processing of the data may be difficult.

System programmers are used to dealing with these problems. Still, several man-days may be spend in modifying software to read a tape into a computer. These problems can and should be avoided in the Air Force ICAM program through the use of proper specifications and standards for file management.

SUMMARY

In summary, the lack of standardization and quality in available operating system software is a major contributor to the difficulties and costs experienced in transporting program systems to different computer installations. The difficulties may be significant even when the computer hardware configuration is nearly identical between the source and the target installations. The costs due to operating system problems may now exceed the costs resulting from minor discrepancies in the programming language compilers involved. Thus some consideration of operating system standardization is essential to the future success of the Air Force CAM projects. It would not be feasible to seek industry or national standardization of this software in the near future; the extent of previous efforts to do so have never progressed past a study stage. It would not be economical either to consider developing a standard operating system or modifying an existing one for Air Force purposes.

However, several areas of interaction between user and the operating system have been identified in the discussion above where attention will be needed to maximize portability:

1. Runtime support routines between user program and operating system.

2. Operating system control language.

3. File management and data base management system interfaces.

4. Input/output software to read files to and from tapes or other media for transporting software and data.

RECOMMENDATIONS ON OPERATING SYSTEMS

1. The Air Force should not undertake to develop a new operating system or modify existing systems for large machines.

2. Standards on programming languages and data base management systems are the best approach to software portability and integratability. In other words, the operating system area should be avoided and system functions implemented using the general purpose programming languages, if at all possible.

3. Limitation of the number of operating systems for the ICAM system may ultimately be necessary. In any case, identification and isolation of all systems dependent code in ICAM software will expedite transitions to other computer systems.

4. No current operating system command language has such features as to recommend it over others. However, this is one area in which standardization is at least technically feasible and should be considered. Federal standardization is already underway in a limited way, addressing the user access protocol to computer networks and services. This effort in FIPS Task Group 20 could be expanded to consider the full range of command language functions. The Air Force should request the Associate Director for ADP Standards, NBS, to determine the feasibility of expanding the scope of work of TG 20 to address Air Force requirements for its CAM program.

5. A standard operating system could be developed for many of the 16 bit (and 32 bit) minicomputers in use today. For a distributed computer system based on 16 bit or 32 bit minicomputers, this approach is attractive and should be examined in detail.

6. File management standards, such as naming conventions for data files and library software, should be enforced for all ICAM development projects to maximize portability of CAM software products. Many potential problems in file management may be avoided through the use of an adequate data base management system (see next section).

REFERENCES

(1) Madnick, S.E. and Donovan, J.J., Operating Systems, New York, McGraw Hill, 1974.

(2) Organick, E.I., The Multics System: An Examination of Its Structure, Cambridge, MIT Press, 1972.

(3) Organick, E.I., Computer System Organization (The B5700/B6700 Series), New York, Academic Press, 1973.

SUMMARY DATA SHEET

The following page summarizes areas of standardization pertaining to Computer Operating Systems.

1. Designation: None

2. Title: Composite summary sheet on Operating Systems

3. Maintenance Authority: Various vendors

4. Scope: The functions of a modern operating system can be divided roughly into: 1) job control (job and process scheduling and control), 2) storage management (allocation of main and secondary memory resources), and 3) file system implementation.

   Communication with an operating system is across two interfaces: system calls and an operating system command language (OSCL). System calls can be thought of as procedure calls to special operating system procedures. They are used in programs to request services of the operating system. An operating system command language is a self-contained but often rudimentary language for direct communication between a user and the operating system. The command language is used to schedule jobs, assign files, etc., and to otherwise direct the execution of programs on behalf of the user. On some of the more well-designed operating systems, the command language exists as a separable part of the system, and thus can be easily changed. In fact, some of these systems can support more than one command language.

5. Relationship to Other Standards: Operating systems require a great deal of interaction with hardware interface standards, code standards and language standards.

6. Competitive Standards:

7. Standardization Status: There have been some attempts to develop a standard operating system command language. These attempts have not succeeded. There appears to be little vendor support for these efforts.

8. Implementation Status:

9. Known Manufacturing Uses:

10. Known Sources of Information: Various vendors.

11. Probable Sources of Information:

12. Bibliography:

Code, Inc., Standardized Job Control Language: Introduction to SJCL Concepts, 1971 October 22, (NTIS AD 742 542).

13. Comments: No standard summary sheets are included in this section because there are no standards in this area. Each vendor of operating systems has a unique approach to the implementation of the user-system interface. No operating system in widespread use can be said to possess sufficient redeeming qualities in its user-system interface that acceptance of it as even an ad hoc standard can be advocated. In fact, very few vendors use the same user-system interface from one generation to another.

STANDARDS FOR COMPUTER SYSTEMS

DATA BASE MANAGEMENT SYSTEMS

INTRODUCTION

TYPES OF DATA BASE MANAGEMENT SYSTEMS

- CODASYL
- Self Contained Packages
- Host Language Approach
- Relational Concept

CENTRALIZED VS DISTRIBUTED DATA BASES

- Areas of Consideration

DEVELOPMENT OF A DBMS VS A COMMERCIAL DBMS

ASSESSMENT OF SYSTEMS AND SOME POPULAR COMMERCIAL PACKAGES

RECOMMENDATIONS

REFERENCES

SUMMARY DATA SHEETS

- See Appendix E on Data Base Management File Structures

INTRODUCTION

A data base management system (DBMS) is a generalized tool for manipulating large data bases. It provides a flexible facility for accommodating different data files and operations while demanding less programming effort than use of conventional programming languages. DBMS possess the following general properties:

* Software which facilitates such operations as data definition, data storage, data maintenance, data retrieval, and output.

* Software which facilitates reference to data by name and not by physical location.

* Software which is general, rather than specific to a particular set of application programs or files.

Since the early 1950's, when generalized file handling routines were first developed, the technology of DBMS has matured considerably. Within the last ten years, a great number of DBMS packages have appeared on the market. No precise count of operational DBMS exists, but it is estimated that at least 200 are now available.

The use of DBMS to control large data bases and provide information to multiple users has already gained acceptance in the data processing world. A recent survey (1) of DBMS usage, just on IBM 360/370 computers in the United States, reported 3,900 DBMS installations as of 1976.

The off-the-shelf DBMS packages do not provide the same set of functions, and the implementation of functions differs widely in depth and strength of effectiveness (2). There are as yet no standards in the area of DBMS as a total package. Many groups are concerned about standardization and are actively working in this area. The CODASYL Data Base Task Group (DBTG) report (3) has been published by the Programming Language Committee of CODASYL as a part of the 1976 COBOL Journal of Development (4). This report represents a specification of a data base management system; future national and international standards will certainly be influenced by this report. The ANSI/X3/SPARC Data Base Study Group has been meeting since 1972; see Interim Report (5). Part of their charge is to develop a basis for DBMS standardization.

In planning the use of data base software for CAM, the Air Force should recognize the severe difficulties that stem from the lack of standard systems, the technical complexity of data base packages and the consequent problems of training and applications analysis, and the rather high costs in storage and processing time that may be unacceptable in some applications. Although available data base packages may be categorized by a similarity of concept, such as the CODASYL or network approach, none of the available packages are even close to being identical in their commands, language, and functions. No de facto standard data base systems exist or are likely to develop in the next three years. The transferability of data base packages between different computers, particularly between minicomputers and large machines, is very limited. Fundamental differences may be presented in the same package because of machine dependent factors, such as the available mass storage.

TYPES OF DATA BASE MANAGEMENT SYSTEMS

Although there are many DBMS packages in the market with different functions and strategies, for the purpose of this study, the total DBMS technology can be described as four broadly different approaches:

1. CODASYL Data Base Task Group Specification
2. Self-Contained Approach
3. Host Language Approach
4. Relational Approach

Inherent in this classification is the data organization which the data base management system supports. The three favored data model approaches are: network, hierarchical, and relational. (See Figures 1, 2 and 3). The CODASYL DBTG supports a network structure. Most of the self-contained type systems support a hierarchical structure. The non-CODASYL host languages are distinguished from the CODASYL host language types becuase of the two popular packages; IMS which is hierarchical, and TOTAL which supports networks. The relational approach models the relational data organization which has a tabular orientation. The characteristics of the four approaches are discussed below.

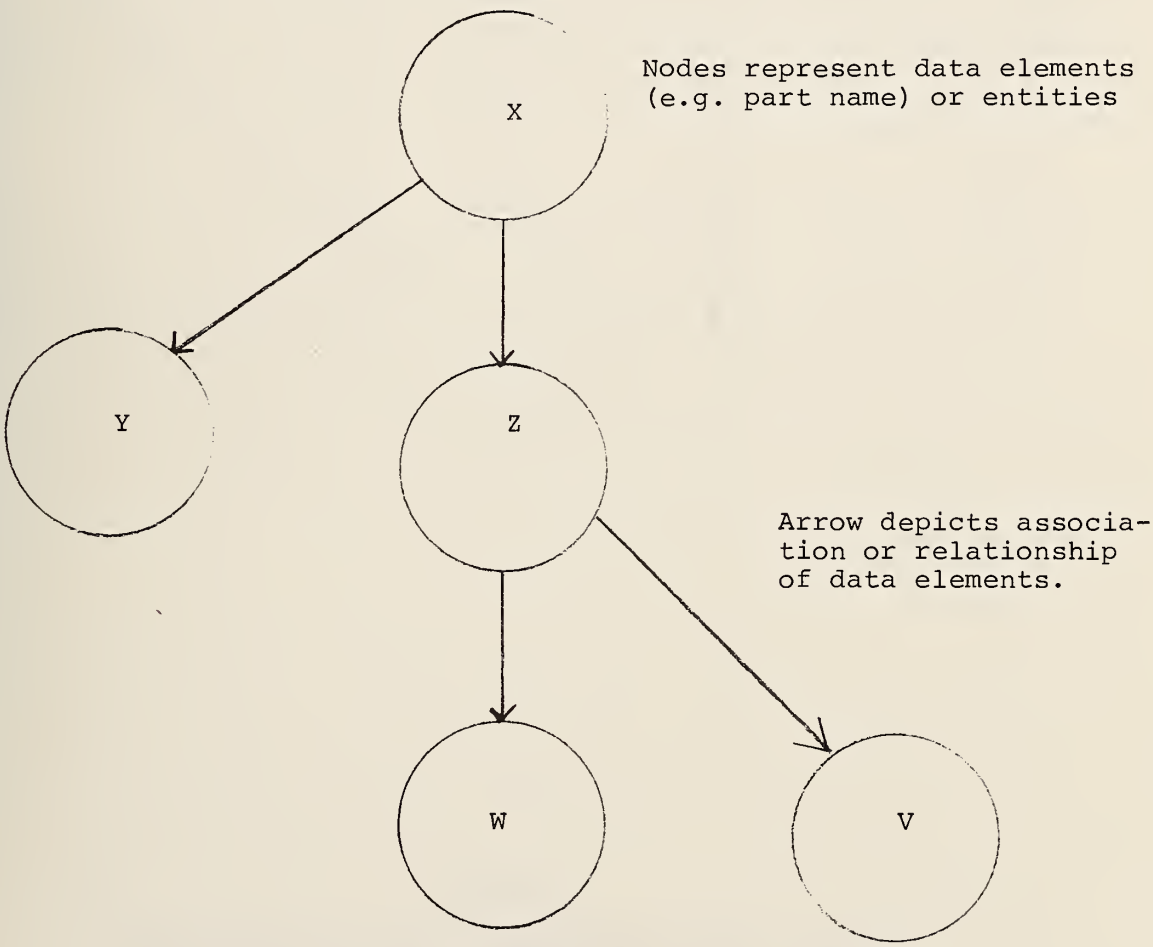


Figure 1

HIERARCHICAL DATA STRUCTURE ILLUSTRATION SHOWING SIMPLE SUPERIOR/SUBORDINATE ASSOCIATIONS

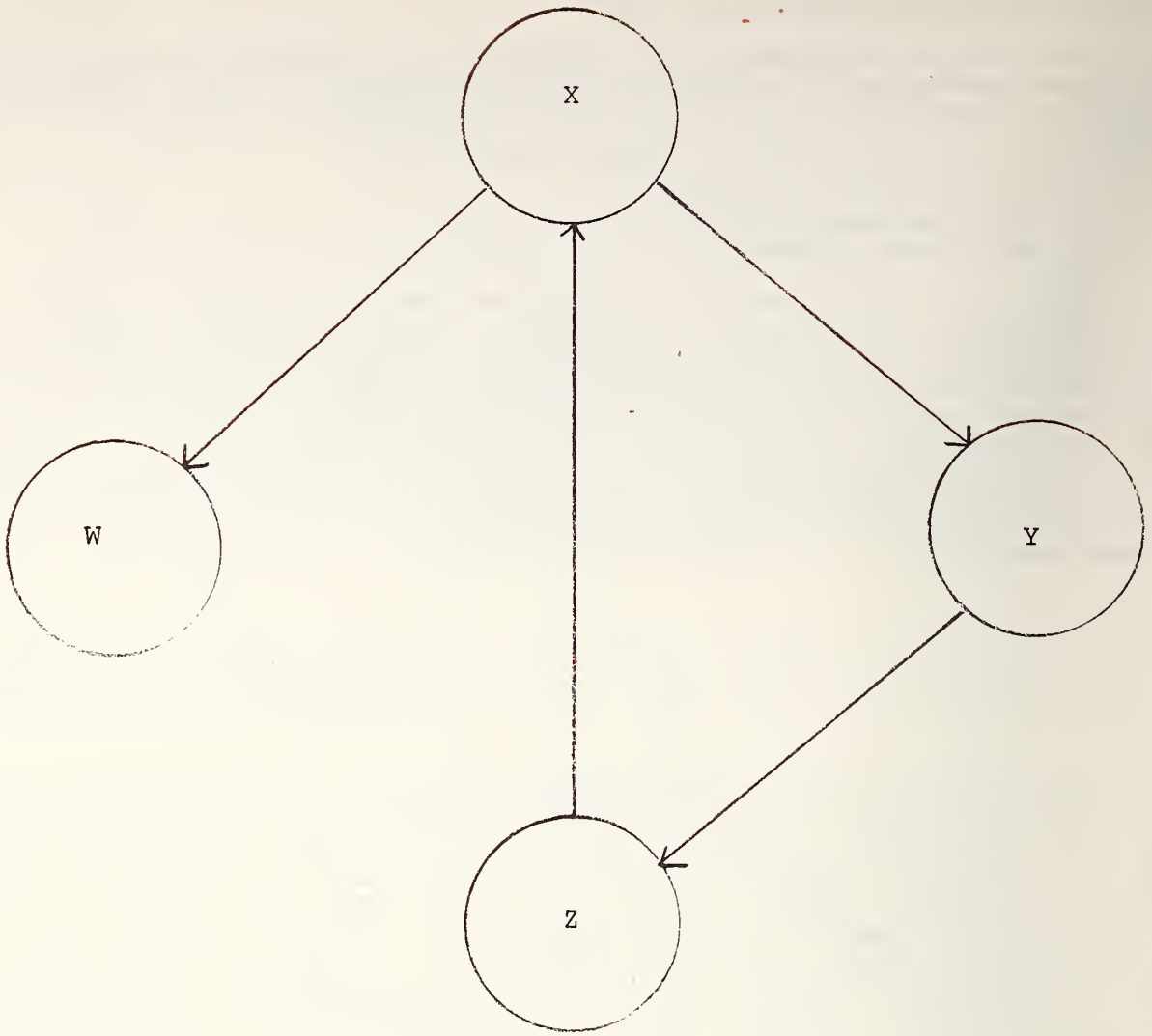


Figure 2

NETWORK DATA STRUCTURE ILLUSTRATION SHOWING
ARBITRARY ASSOCIATIONS OF DATA ELEMENTS

A Relation → A Particular Tabular Association

Date element X	Y	Z
values	associated values	associated values

Figure 3

RELATIONAL SYSTEMS REPRESENT COMPLEX DATA ASSOCIATIONS IN SIMPLE TABLES

CODASYL Data Base Task Group Specification

The CODASYL Data Base Task Group (DBTG) specification as published in 1971 consists of two parts: (1) syntax and semantics of a data description language (DDL) for describing the structured data base, (2) the definition of data manipulation language (DML) statements to augment COBOL (for retrieving and updating data in the data base).

Three important characteristics of the CODASYL DBTG specification (see Fig. 4) are as follows:

* The data relationships are explicitly defined in the data base. Records that are logically related are tied together by either pointers or by indexes. The relationships are defined when the data base (schema) is defined. The advantage of this architecture is that the relationships can be carefully worked out by the people who understand the data. The disadvantage is that it can be a nontrivial task to change the relationships.

* The Data Definition Language (DDL) is separated into two parts: (1) the Schema DDL is totally language-independent and used to describe the data relationships as mentioned above, and (2) the Sub-Schema DDL which is fashioned around the language of the user's program and restructures the data base for the particular requirements of the program. Thus, this separation permits multiple-language interface, data independence, a smaller view of the data to a program, and protection for the remainder of the data base not used in a given application.

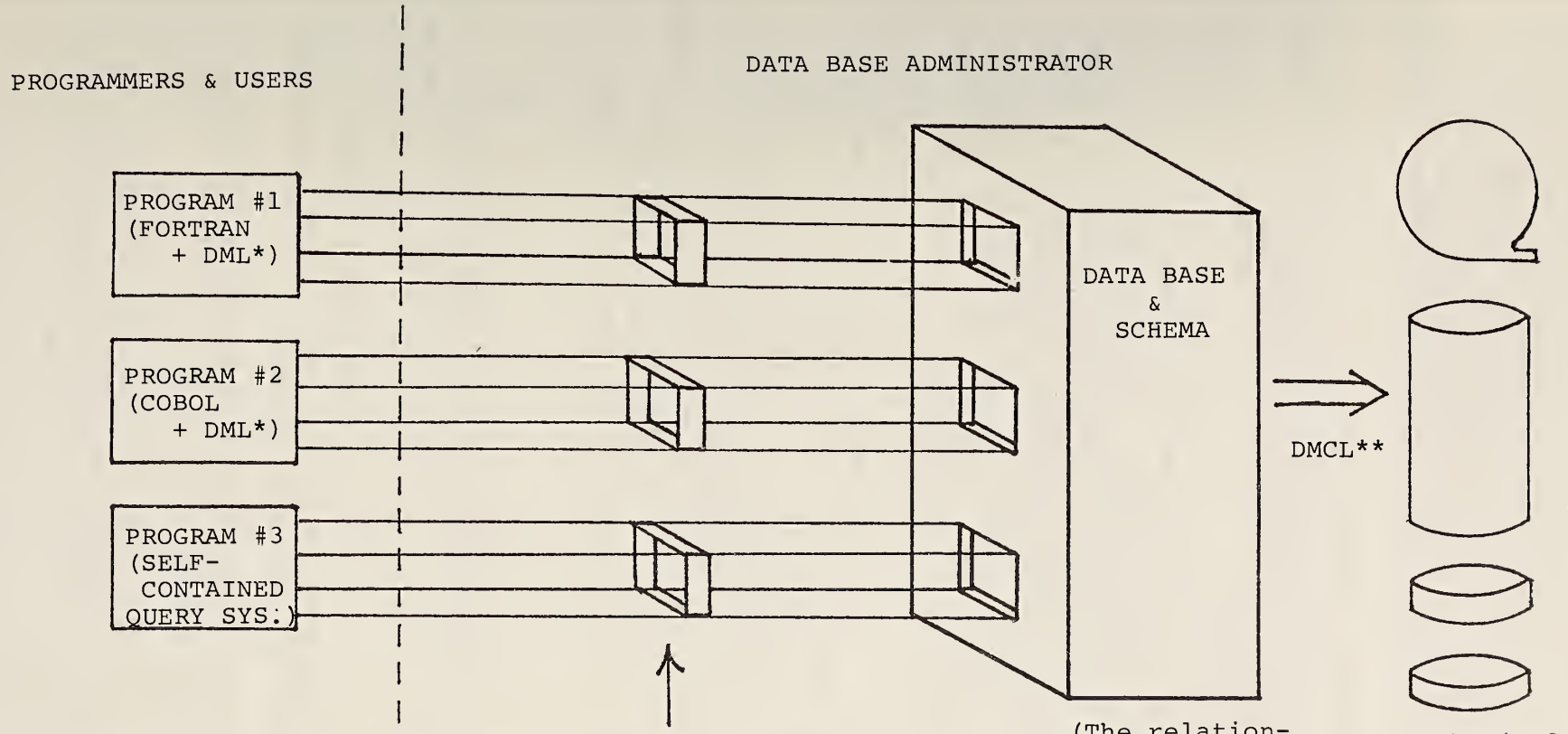
* The Data Manipulation Language (DML) has been designed to help the application programmer "navigate" within the data base. Any given record in the data base can be related to a number of other records, and it might be accessed by any of several paths. The application programmer must know where his program is operating, and how it should retrace its steps when a search proves unfruitful.

The CODASYL DBTG approach adopts the network data model. A network is a more general structure than a hierarchical structure because a given node may have any number of immediate superiors as well as any number of immediate subordinates. Therefore, this approach provides the most powerful means of handling complex data structures, but querying and reporting may prove to be a complex matter.

Self-Contained Packages

The majority of the commerically available data base software packages are of the self-contained type. Typically, these systems possess three major processing capabilities: data creation, data update, and data retrieval and report formatting. A self-contained user language is provided to accomplish all three processing tasks. These systems are aimed at handling a certain set of data base functions in such a way that conventional procedural programming is not required. The capability to specify in detail the search method and data retrieval the programmer wishes is replaced by preprogrammed or built in processing algorithms so that the amount of writing required by the user is minimized. The self-contained systems are optimized on their interrogation and update functions. As a result they represent the most advanced DBMS in the area of user language capabilities.

The very reason for the success of the self-contained DBMS, ie. their high level, non-procedural interrogation language, becomes a large disadvantage in those cases where the user wishes to exercise control over the sequence of detailed steps the system uses to process his requirements. Some systems also provide external programming interfaces where the user can enter his own



*CODASYL Specification
Schema DDL
Sub-Schema DDL
Data Manipulation Language (DML)

**Device Media Control Language not specified.

The Sub-Schemas (as defined by the Sub-Schema DDL*)

1) restructure the data base for the particular requirements of the program
2) prevent program access to unauthorized portions of the data base
3) allow the use of different procedural languages and self-contained languages

(The relationships between the data as defined by the independent Schema Data Definition Language (DDL*))

Figure 4

routines written either in FORTRAN, COBOL, PL/I or assembly language to perform processing not inherently supported by the system. However, this does not yield the same capabilities as a host-language DBMS with its appropriate data manipulation language (DML). The majority of the self-contained data base management packages model the hierarchical data structure with repeating groups. Typically, the system employs the inverted indexed technique to facilitate quick retrieval.

Characteristics of these systems are that they are:

* End-user oriented. The user-language is easy, natural and English-like. Very little application programming is necessary. However, the user is paying for an added layer of software with less efficiency and flexibility.

* Easy to install. After the data base has been created it is relatively easy to change the structure. The data base can be built an application at a time, without requiring that the whole data base be defined at the outset. These capabilities are largely a result of the implementation of an inverted or partially inverted file system for storing the data. However, the (partially) inverted file structure results in difficulty in handling of queries that specify records located in different branches and/or at different levels of a hierarchy, and in addition, results in considerable storage space required for the indicies.

* Easier to formulate unanticipated ad-hoc queries. Self-contained systems permit the user to ask the question directly, and he has no need to call on a programmer as an intermediary. For those applications that self-contained systems can handle, they offer considerably reduced set-up time and a vast reduction in the time required to prepare a new interrogation or update to the data base. However, the end-user must be aware of the data structure supported by the system; if the needed data elements are not keyed or inverted, the system either searches sequentially or refuses to respond. Another caution on a hierarchical tree structure, if the data elements requested in the queries are not of the same hierarchy, no "hits," or even erroneous ones, will be made.

Host Language Approach

Although the CODASYL DBTG specification is a host language type, we have treated it as a separate entity because of two distinctly different packages that are already widely used; IMS by IBM, and TOTAL by Cincom (See Table 1). The host language approach is characterized by the following features:

* The system is designed as a tool for the experienced programmer.

* System functions are invoked from within host programming languages (e.g., COBOL, FORTRAN, PL/I, assembly language).

* The supported data structures generally permit more user control, even down to the physical storage level, than those found in self-contained data management systems.

Host language DBMS generally lack high level language constructs for conditional data, updates and retrievals, as are found in the self-contained type. Typically, this is because the emphasis has been placed on defining logical relationships among records or group of records in large interrelated data bases, rather than on generalized functions. However, these systems do interface to separate Report Program Generators (RPG)/ Query packages (this provides some aspect of the interrogation capabilities

inherent in the self-contained systems) while providing the flexibility to the user of specifying the details of how his request is to be processed through the use of the procedural-DML.

Host language type systems can be thought of as extensions to the programming languages. The method chosen to interface the host language data management system with the programming language is usually through the facilities of the CALL statement in the programming language.

Host language type systems provide powerful data management functions for manipulating data, programmable through the flexibility of a programming language and considerable user control over the physical storage structure.

Relational Concept

With commercially available DBMS, the variety of data representation characteristics which can be changed without logically impairing some application programs is still quite limited. Some people feel that the present data base management systems require entirely too much knowledge on the part of the user on how the data base is structured and how the data should be accessed (for the case of host language systems) or are too limited by preprogrammed algorithms (for the use of self-contained systems). Instead, the user, be he an application programmer, manager, engineer, or other - should simply have to specify what data is desired, not how it is to be retrieved. The main problem with present systems is that the data relationships are structured, which favors some types of access at the expense of others, i.e. the application programs are not independent of the data base. (See Appendix for a discussion of some of the characteristics of various file structure techniques.)

Three of the principal kinds of data dependence are:

1) Ordering dependence - e.g. records of a file concerning parts might be sorted in ascending order by part serial number - these systems fail if this ordering is replaced by a different one (e.g. if a search is desired by part material - aluminum, brass, steel etc.) The same is true for a stored ordering implemented by means of pointers.

2) Indexing dependence - from an informational standpoint, indices are redundant components of the data representation, requiring large additional storage capacity from the data structure.

3) Access path dependence - many of the existing formatted data systems provide users with tree-structured files or slightly more general network models of the data. Application programs developed to work with these systems tend to be logically impaired if the trees or networks are changed in structure. Or, if a query is made for data in other than the structured form in the data base, then a time consuming, total and complete search of the data base may be required. In general, the user (or his program) is required to exploit a collection of user access paths to the data.

The relational data base is proposed as a possible solution to these problems. This concept is relatively new (Codd 1970) (6). The approach is based on the premise that users of data base management systems are becoming increasingly concerned with the information content of their data, as opposed to specific representation details. That is, there is a trend toward data base user interfaces that deal with information in application terms rather than with the bits, pointers, and lists that are used to represent information on computer mass-storage devices.

The relational approach to data base management can be characterized as follows:

* Simplicity of user interface. The relational user is presented with a single, consistent data structure and requests can be formulated strictly in terms of information content, without reference to most system-oriented complexities.

* Data independence. The user is relieved of concerns for knowing specific information storage and access strategy.

* Flexible response to ad-hoc queries. Since all information is represented by data values in relations, there is no preferred format for a question.

The most serious question regarding the relational approach is whether it can be implemented to form an efficient and operationally viable DBMS. Many prototype systems exist but no commerical systems exist that are truly relational. These systems are summarized in Table 1.

CENTRALIZED VS. DISTRIBUTED DATA BASES

There is no clear-cut best answer regarding which approach to use. Each approach has its advantages and shortcomings. But the answer seems to depend upon the particular needs and application environment. Data base management packages also need to be selected in the context of some architectural configuration. Two opposite data base architectures can be identified: the centralized data base approach and the distributed data base approach.

Centralized data base - A central data base is usually maintained using a large-scale third generation type mainframe. Data may be generated centrally or bulk entered from several remote data entry stations. The centralized approach allows centralized control of the data bases, which is necessary for efficient data administration. The data base management system for the central computer would need to have full facilities for storage and maintenance of data. In particular, some of the mandatory features should be powerful control functions for data validation, update control, centralized data dictionary capabilities to manage the centralized data base. Retrieval and output reports can be optionally weighed against very end-user oriented query language facility versus transaction invocation via predefined process written in a programming language such as COBOL. The data base management system for the centralized architectural approach can be all of the above four types: CODASYL DBTG-like, non-CODASYL self-contained, non-CODASYL host language, or the relational approach.

Distributed data base approach - The development of computer networks has led to the prospect of distributed data bases. Distributed data bases also include distributed processing which generally consists of remote stations distributed throughout remote locations. The remote stations evolved from intelligent terminals to, at present, minicomputers installed with their own secondary storage. Distributed data bases can have numerous configurations. One scenario might be identified as follows: The distributed information system might be a multi-level hierarchy of processors, generally matching, at each level, the organizational structure and complexity of the manufacturing system. The network could be composed of a number of mini-computers so that processing logic and storage (distributed data bases) would be placed at or near the points where transactions occur. A common design would be used for the numerous data bases and for the data base management systems, so that the total data base could be distributed throughout the system. However, due to the data base being stored under a common data base structure, using common data definitions, any portion of the total data base would be accessible from any node in the network.

This modular design allows modules to be added and others deleted to meet the needs of a particular situation. This would give the network

DATA BASE MANAGEMENT SYSTEM	SYSTEM 2000	ADABAS	IMS/VS	TOTAL	IDMS
TYPE OF SYSTEM	Self-contained	Self-contained	Host-language	Host-language	CODASYL
VENDOR	MRI Systems Austin, Texas	Software AG Reston, Virginia	IBM	CINCOM SYSTEMS Cincinnati, Ohio	CULLINANE CORP. Boston, MA
COMPUTERS	IBM - 360/370 UNIVAC - 1106/1108/1110 CDC 6000	IBM 360/370 UNIVAC Series 70	IBM 370	IBM 360/370, IBM 3 UNIVAC 70, 9000, CDC 6000 HON 200/2010, NCR H60 VARIAN, INTERDATA, DEC 11	IBM 360/370 UNIVAC 70, 90 PDP 11/45
MAIN MEMORY REQU.	144-210 K bytes	120 K bytes DML 150 K bytes DDL	450 K bytes	35 K bytes (not including buffers)	50-65 K bytes (includes 15 K buffers)
USER LANGUAGES	Own Language COBOL, FORTRAN, PL/1	Own Language COBOL, FORTRAN, PL/1	COBOL, PL/1	COBOL, FORTRAN, PL/1	COBOL, FORTRAN, PL/1
FILE STRUCTURE AND ACCESS METHOD	Hierarchical, Sequential, Inverted	Inverted Sequential Access Method (ISAM), Inverted Direct (IDAM), Inverted, Network	HDAM - hierarchical direct access method HSAM HIDAM HISAM	Full file inversions, Chains, Network, Random/Calculation	Chains, Networks, Rings Hierarchies Random/Calculation Full file inversions
TELE PROCESSING INTERFACES	Has own TP package	INTERCOMM, CICS, TASK/MASTER	CICS, DC, INTERCOMM, TASK/MASTER	CICS, ENVIRON/1, INTERCOMM TASK/MASTER	CICS, DC, INTERCOMM
REPORT GENERATOR QUERY FACILITIES	Has own RPG package and query	ADA WRITER, ADASCRIPT, EASYTRIEVE	CULPRIT, MARK IV, ASI-ST	SOCRATES, MARK IV, ASI-ST, CULPRIT, EXTRACTO, EASYTRIEVE	CULPRIT
NUMBER OF INSTALLATIONS	> 100	> 50	> 500	> 750	> 100

TABLE 1 - DATA BASE MANAGEMENT SYSTEMS

DATA BASE MANAGEMENT SYSTEM	INGRES	NOMAD	DMS 1100	IDS - II	EDMS
TYPE OF SYSTEM	Relational	Relational	CODASYL	CODASYL	CODASYL
VENDOR	University of California at Berkeley	National C.S.S. Inc.	UNIVAC Roseville, Minnesota	Honeywell Information Sys. Phoenix, Arizona	Xerox Information Systems Group El Segundo, CA
COMPUTERS	DEC PDP 11 with Unix	Time share service	UNIVAC 1106, 1108, 1110	H 9GS) 200, 400, 600 H-6000, Large Series 60's	XDS SIGMA 6, 7, 9, 560
MAIN MEMORY REQU.	----	----	16-65 K words (36 bit)	2 K words Monitor 10 K words/Partition	12-18 K words (32 bits)
USER LANGUAGES	QUEL	----	COBOL, FORTRAN	COBOL, FORTRAN	COBOL, FORTRAN
FILE STRUCTURE AND ACCESS METHOD	Relational	Relational and Hierarchical	ISAM, Direct Random/Calculation Inverted Tables, Pointer array	Random/direct, Hierarchy, Networks	Direct, Random/Calculation Indexed, Chains, Network
TELE PROCESSING INTERFACES	----	----	Transaction Interface Package	----	----
REPORT GENERATOR FACILITIES	----	----	----	MDQS Report Generator (on-line request only)	----
NUMBER OF INSTALLATIONS	> 6	> 5	> 40	> 170 (including IDS-I which is not CODASYL)	> 50

TABLE 1 - DATA BASE MANAGEMENT SYSTEMS

the ability to withstand severe damage to some of the processors (or some of the storage units) with the remainder being able to continue operation. (Due to the centralization of the data that has occurred as a result of the installation of DBMS some plants have experienced severe problems in the total shut down of the operations when something goes wrong with the system. The solutions organizations are arriving at turn out to be approximations to the network philosophy - some companies have logically and physically subdividided the data base to cause it to reside on different storage units - others have installed additional minicomputers to allow for continued data-taking if the main system goes down . The distributed communications in a network structure would provide at least two independent paths between any two modes, so as to provide automatic alternate routing for messages. Two kinds of data base management software can be considered in this scenerio:

1) There are data base management systems specifically built for the minicomputer. For example, Hewlett-Packard has developed a package called IMAGE, and Data General has a data base management system (INFOS) for its Eclipse series. Varian and Harris have signed contracts with Cincom to offer TOTAL. Cullinane offers IDMS which is precompiled on an IBM 360 and the object module run on DEC's PDP 11/45. Other prototype systems which are not yet commerically available are operational for DEC's PDP-11.

2) Another approach is relatively new. It is the concept of a Data Computer. It consists of hardware solely used for the accessing of data. A prototype is being built by Computer Corporation of America for the ARPA network. A similar concept is the "back-end" computer concept where the data base is maintained as the "back-end" computer, usually a minicomputer, and interfaced to a host computer, usually a large-scale third-generation type where user request language is translated.

Many advantages would accrue from a geographically dispersed approach to data base management, including:

* Better provisions for protection than with centralized systems.

* Flexibility and localized control of data processing activities.

* Data validation at local sites, resulting in cleaner data input to the central computer data base.

* Flexibility and potential of a distributed architecture.

The distributed data base concept is not without problems. For example, does the user have to know the data location, does the request language need to be different to access different distributed data bases, etc. Most importantly, there is no fully operational distributed data base system as yet.

Areas of Consideration

1) There are no standards in data base management as a total package. The CODASYL DBTG report lends itself to be a potential candidate for standardization, but many feel that the CODASYL DBTG approach is not universally accepted and should not be standardized. This is a result of the general feeling that data base management systems are very much an evolving technology where many developments are yet to come. It is felt that the standardization of the CODASYL data base task group (DBTG) report would provide sufficient inertia to the system so as to impede the development of new and perhaps better data base management systems. In addition, the

specifications of the DBTG report are felt to be too incomplete. The proposed data manipulation language (DML) is felt to be too procedurally oriented (therefore, not easily used by the non-programmer) and to have shortcomings with respect to data independence, data integrity, and compatibility (Guide-Share report). The CODASYL specifications make no provision for handling existing sequential and index sequential file structures. Nor do they define a device media control language (DMCL) which is the storage structure language used to describe the mapping of the data onto physical storage media. And while the specification of a totally language-independent data definition language (DDL) would theoretically allow access to the data base by either a host-language request or a self-contained-like query, only the host-language data manipulation language (DML) has been specified. CODASYL has not yet developed the specifications for query and reporting languages. As mentioned earlier, the application programmer must know how to "navigate" in a complex data base environment. Query and reporting languages will have to do such "navigating" automatically, and as a result could be quite complex in their development. However, this difficulty exists for any type of complex data base structure (i.e. a network or graph structure), and is not limited to CODASYL-like systems. Some report program generators (RPG)/ Query packages are available and interface to the commerical CODASYL systems. These are not as powerful, as yet, as the self-contained system query capabilities. In addition, CODASYL does not specify the recovery techniques to be used after a system goes down, nor the method to be used for restructuring and/or reorganizing the data base. All of these features are left up to individual vendor and/or user to supply.

2) The possibility exists that there will be standards in each of the different DBMS approaches. The rationale is that since there are different programming languages for different applications, e.g. COBOL, FORTRAN, PL/I, BASIC, it is conceivable that there may be different data base management systems under consideration for different applications.

3) The contemporary large-scale data base management systems are built with separable functional modules. For example, a data base management system may consist of a nucleus plus the following kind of functional modules:

   * the data definition language for specifying the logical structure,

   * the data dictionary/directories for ease of managing data description,

   * the teleprocessing message handling for on-line interactions,

   * the user language processor for user interface to manipulate the data,

   * the protocols for invoking procedures on the data base system,

   * the data access methods for physical storage accesses,

   * the report writer for formatting fancy reports.

   Each of these areas may potentially be considered for standardization. Already, the commercial world has been marketing data base management software in optionally upgradable and pluggable modules. Adjunct packages such as report writer, query languages, teleprocessing front-end, data dictionary and various utilities such as

bulk load, sort, etc. have started to appear in the marketplace. These adjunct packages usually operate in conjunction with a specific data base management system. Although the interfaces of these packages are not as yet flexible, standardizing the interfaces of a data base system leads to the concept of interchangeable parts.

DEVELOPMENT OF A DBMS VS A COMMERCIAL DBMS

The development of a data base management system is considered too involved, too expensive, and too risky to attempt. Years of problems of cost overruns, unattained goals, and expensive maintenance have plagued new development efforts. There are, at present, a number of commerical data base management systems available, that, while they fall short of meeting the requirements of an idealized DBMS, effectively provide for the recording, retrieving, and updating of large, complex stores of data. It is recommended that the Air Force choose one of these commercial systems with the expectation of updating or even converting to an entirely new system in several years as major advances in DBMS occur. To wait for these advances, or to attempt to develop new systems (which would entail a great expenditure of resources in a not-well-understood field to obtain questionable improvements) would cause a major setback to the overall project. Experience and a clearer understanding of the problem of what is really needed from a DBMS in an Integrated Computer Aided Manufacturing system can be better obtained from using an existing commercial DBMS in a working system rather than attempting to develop a DBMS for a non-working system.

At present, there are no working commerical packages of the relational DBMS type. This area is considered too experimental to be implemented in the Air Force project. Much more research and development is required to see if these relational systems can provide the theoretical advances they promise.

There are a number of commerical systems (both host-language and self-contained) available and the choice of a system should include such considerations as

flexibility - in terms of use on a number of different computers in a distributed system all addressing the distributed data base.

portability - in terms of being able to move a DBMS or a data base or portions of a data base from an existing hardware/software complex to another.

adaptability - in terms of the ability to change data definitions easily (i.e., to add, delete, lengthen, shorten, or change the relative location of fields within records, records within sets or files, or relationship indicators (pointers or indices)), to do all of this without having to make changes in application programs and without having to dump and reload the whole data base.

ASSESSMENT OF SYSTEMS AND SOME POPULAR COMMERCIAL PACKAGES

Self-contained systems - System 2000 marketed by the MRI systems corporation and ADABAS, distributed by Software AG, are among the most popular self-contained DBMS. As mentioned previously, these self-contained systems originated with their own internal language with no connection to any of the procedure-oriented languages (such as COBOL or FORTRAN). However, most self-contained systems now provide interfaces to allow use of COBOL, FORTRAN, and PL/1 in formulating data requests, but the use of these procedural languages does not result in the same capabilities or

efficiencies as obtained with host-language DML. Even though these self-contained systems are very good in query and reporting capabilities, they are not recommended for the Air Force project because, due to the limitations of a hierarchical file structure, System 2000 does not handle complex data structures (networks), ADABAS, however does have a network file structure that is like the CODASYL approach; the limited capabilities of the built-in processing algorithms which are only partially corrected for by providing interfaces to procedural language programs; and, their rather large size somewhat restricts their use in a distributed system.

Host-Language Systems - The host-language systems such as IBM's Information Management System (IMS) and CINCOM's TOTAL are embedded in a host language (COBOL, PL/1, or FORTRAN (TOTAL only)) and therefore are built upon the facilities of a procedural language.

IMS is a hierarchical based file structure which means network-type relationships are difficult to handle, but IMS does allow network-like structures. This probably results in a considerable overhead in additional pointers and indices and is probably partially responsible for IMS requiring the largest amount of main memory (450K bytes) of any of the DBMS. IMS does not have a FORTRAN host-language capability.

IBM will provide the hardware, operating system, data base management system, data communications package, and query and reporting facilities. Further, IBM makes frequent improvements to these, and gives them good support. IBM is not currently implementing the CODASYL DBTG specifications and has no plans to do so.

User comments on IMS include good recovery, flexibility in data orgnization and administration, versatile file structures, and that changes to data relationships can be achieved via rule redefinition without requiring major program modification or data reentry. However, IMS is also reported to be a very complex product requiring much application software support, and it has large core requirements. IMS is not recommended for this project because it is specific to IBM equipment and produces a sole source condition incompatible with the objectives of a portable system. In addition, the DBMS is so large that it does not fit in with the concept of a distributed system, which has been given as a potential Air Force objective.

TOTAL is the most successful data base management system in terms of number of installations (>750). It, like the CODASYL DBTG specifications, was derived from Integrated Data Store (IDS), the grandfather of the data base management systems. TOTAL does not, however, conform to the CODASYL DBTG specifications but conceivably could be converted (TOTAL is similar to the CODASYL specifications in the way data is structured and the way the data relationships are expressed).

TOTAL does allow file inversions, chains, and networks so that complex data structures can be easily represented and quickly retrieved. Users report that the system requires small amounts of core ($\sim$ 35K bytes) and is inexpensive and easy to install. It was developed with small users (DOS environment) in mind. But while it is simple to use, the system is somewhat awkward for large multi-file use since when one file is being accessed, all other files are locked out. Hence, simultaneous processing of several data files is impossible. Also the system's performance degrades with the addition of new variable data records over a period of time.

The major drawbacks seen with TOTAL at present are its inability to efficiently handle multi-file access. However, an interactive query package has recently been added, and future developments could easily make TOTAL a reasonable alternative to a CODASYL based system.

CODASYL Systems - The data base management systems built along the guidelines of the CODASYL specifications are considered to be the most promising, at present, for implementation by the Air Force. The CODASYL specifications represent the most comprehenisve effort to form a "common" (not standard) and machine independent approach in the development of a DBMS. No real standards are expected in this field for at least five to ten years due to the present lack of knowledge and understanding about how a data base should really be structured and accessed and what all the requirements are for the "best" data base management system.

The CODASYL specifications have gone the farthest in providing the basis for a common, modular architecture for DBMS. This approach of carefully partitioning the system to develop a modular architecture has two very important advantages:

1) by partitioning the system, interfaces can be carefully defined and eventually standardized,

2) a common architecture for data base management systems should facilitate the development of distributed systems with distributed data bases.

Data base management systems, in general, were originally designed either as a host-language system or a self-contained system according to expected applications and many that were developed were specially tailored for the unique applications of that individual company, that is, there are many unique DBMS solutions. Now, the trend of the successful system is to modularity: the self-contained systems have interfaces to procedural languages; the host-language systems have interfaces to report generation/query systems; both types of DBMS have provided interfaces to teleprocessing packages, and data dictionary/directories. Thus, all of the DBMS appear to be moving in the direction that the CODASYL specifications had originally outlined. The CODASYL specifications specifically and comprehensively attack this problem of modular partitioning and definition, rather than backing into it as the other commercial systems appear to be doing. Whereas all of the systems seem to be coming to the same end, CODASYL, alone has attempted to charter the path and define the architecture. Thus, the CODASYL sepcifications are most in line with the philosophy of correctly defining the logical modules and then standardizing on the interfaces connecting them. The CODASYL specifications provide the type of common architecture necessary for the distributed computer network. But although a number of CODASYL-type systems are available, they are by no means identical. The specifications themselves are in a state of change by the Data Description Language Committee. Additionally, it appears that TOTAL is widely implemented and is a reasonable alternative to the CODASYL approach. Hence, a competetive procurement should be used to select a single DBMS to suit ICAM requirements for the near future.

RECOMMENDATIONS

1. A common data base management system will be critical to the integration of ICAM software. In particular the DBMS provides the interface between all applications programs.

2. The Air Force should not attempt the development of any new general purpose data base management system due to the expenditure of resources required without any guarantee of success.

3. Functional specifications should be prepared for the competetive procurement of a commercially available data base software package to support all near-term ICAM projects. The specification should require the package to be available on all hardware systems that would be considered for CAM applications in the first few years of the program. Emphasis should be placed on obtaining modular architecture, well defined interfaces, portability of applications programs, integratability of ICAM modules, and future adaptability to a computer network system with distributed data bases. The evaluation for selection should include a benchmark demonstration of performance on a typical CAM application.

4. The Air Force should initiate participation in NBS FIPS Task Group 24, which has begun to consider government-wide needs for data base standards, and in ANSI efforts, such as the ANSI/X3/SPARC Study Group, that is identifying the need for ANSI standards.

5. The Air Force should monitor the continuing research and development work with relational data base management systems.

REFERENCES

(1) International Data Corp., "The Data Base Management Software Market on IBM 360/370 Systems," International Data Corp. #1685, 214 Third Avenue, Waltham, Mass., 02154, May 1976.

(2) Fong, E., Collica, J., and Marron, B. Six Data Base Management Systems: Feature Analysis and User Experience. NBS Technical Note 887, Nov., 1975.

(3) CODASYL Programming Language Committee, Data Base Task Group Report, Available from ACM, April 1971.

(4) CODASYL Programming Language Committee, COBOL Journal of Development, 1976.

(5) ANSI X3/SPARC/Study Group - Data Base Systems, "Interim Report" ACM/SIGMOD Newsletter: fdt. 7,2 (Dec. 1975).

(6) Codd, E. F. "A Relational Model of Data for Large Shared Data Banks", Comm. ACM 13,6 (June 1976) pp. 377-397.

Berg, J. L., ed. Data Base Directions, NBS Special Publication 451, Sept., 1976.

Sibley, E. H., The CODASYL Data Base Approach: A COBOL Example of Design of Use of a Personnel File. NBSIR 74-500, Feb., 1974.

(7) "Data Base Management System Requirements, Nov. 11, 1970," by the Joint Guide-Share Data Base Requirements Group. Order from Share Secretary, Suite 750, 25 Broadway, New York, NY 10004, price $1.50.

(8) "The Debate on Data Base Management" by Richard G. Canning. EDP Analyzer, March 1972, Vista California 92083.

(9) "The Current Status of Data Management" by Richard G. Canning. EDP Analyzer, February 1974, Vista California 92083.

(10) "Introduction to Feature Analysis of Generalized DBMS," Communications of the ACM, May 1971 p. 302-318.

(11) "Concepts of Data Base Management" Honeywell's manual for their IDS Technical Presentation.

(12) "Data Base Design" The Manual for AMR International, Inc.'s Course on data base design. Copyrighted 1973 by AMR International, Inc.

SUMMARY DATA SHEETS

The following data sheets summarize those standardization activities which apply to Data Base Management Systems.

1. Designation: None

2. Title: CODASYL Data Base Task Group (DBTG) Specification

3. Maintenance Authority: CODASYL Data Description Language Committee for the Data Definition Language (DDL) portion; CODASYL Programming Language Committee for the Data Manipulation Language (DML) portion.

4. Scope: Proposed standard for data base management systems. The DBTG specification includes the DDL and the DML.

5. Relationship to Other Standards: ANSI COBOL (base for DML specification).

6. Competitive Standards: ANSI X3/SPARC/DBMS Interim Report; Non-CODASYL Self-Contained approaches; Non-CODASYL Host Language approaches; Relational approaches.

7. Standardization Status: None

8. Implementation Status: There are commercial data base systems implemented, based on the CODASYL specification. While these systems may employ syntax that is slightly different from the CODASYL specification, they follow the same basic data model. Some of the commercially available systems which are generally deemed to be "CODASYL DBTG type" systems are:

   - DBMS-10 (Data Base Management System-10) developed by Rapidata, Inc.; as a Digital Equipment Corporation product runs on the DEC PDP-10.
   - IDMS (Integrated Data Management System) developed by Cullinane Corporation runs on IBM 360/370 and on UNIVAC 70.
   - DMS 1100 (Data Management System 1100) developed by Xerox runs on the Xerox SIGMA 6, 7, 9 and 560 machines.

9. Known Manufacturing Uses: Used primarily for business applications such as payroll and inventory control.

10. Known Sources of Information: Chairman, DDLC, CODASYL, Box 124, Monroeville, PA 15146

11. Probable Sources of Information: Cullinane, Rapidata, DEC, UNIVAC, NBS.

12. Bibliography:

1.a. CODASYL Programming Language Committee, Data Base Task Group Report. Available from ACM, April 1971.

1.b. CODASYL Data Description Language Committee, CODASYL Data Description Language, Journal of Development, U.S. Department of Commerce, NBS, NBS Handbook 113.

1.c. CODASYL Programming Language Committee, COBOL Journal of Development 1976, Chapter 12 - Data Manipulation Language, Section 4 - Subschema Specifications.

13. Comments: In May 1967, the Conference on Data Systems Languages (CODASYL) formed a group called the Data Base Task Group (DBTG). In October 1969, the DBTG produced a specification of a data base known as the DBTG Report (1.a.). The report detailed the semantics and syntax of a Data Description Language (DDL) and a Data Manipulation Language (DML). The DDL is a language for describing a data base. The DML is a language which is associated with a host language such as COBOL, FORTRAN, PL/I, etc., and which allows the manipulation of the data bases described by the DDL.

The Data Definition Language was developed with the intent that it would eventually become the basis for an industry standard and that many individual host languages could interface with implementations of it. A Data Description Language Committee was established. This Committee produced a CODASYL Data Description Language Journal of Development dated June 1973 (1.b.).

The data manipulation work was continued under the Programming Language Committee with the goal of developing subschemas and a DML specification. A CODASYL COBOL Journal of Development which specifies the Data Manipulation Language and subschema for COBOL was published in May 1975 (1.c.). Similar specifications for FORTRAN and perhaps for PL/I are still in the working stage.

1. Designation: None (composite summary sheet)

2. Title: Composite summary sheet on Self-Contained Data Management Approach

3. Maintenance Authority:

4. Scope: The majority of the commercially available data base management systems are of this type. They are widely used by government and industry as a nucleus in building special applications for customized data processing work.

5. Relationship to Other Standards: COBOL, FORTRAN, PL/I (for procedural language interface).

6. Competitive Standards: CODASYL DBTG Specification

7. Standardization Status: None

8. Implementation Status: There are many operational packages that are commercially available. These packages differ in the functions provided. Among the many, the following packages are believed to be representative of widely recognized and proven products available in the market today.

Package Name	Supplier	Computers	Initial Installation	No. of Users [1]
ADABAS	Software Ag.	IBM 360/370	March 1971 (in Germany) Marketed in U.S. since early 1972.	Over 150 as of May 1976
INQUIRE	Infodata Systems Inc.	IBM 360/370	1969	Approximately 70 as of May 1976
MODEL 204	Computer Corp. of America	IBM 360/370	1971	22 as of May 1976
System 2000	MRI Systems Inc.	IBM 360/370 UNIVAC 1100 Series CDC 6000 Series	July 1970	Over 100 as of May 1976

9. Known Manufacturing Uses: Current applications are: management information systems, inventory control, ecological data bases, personnel information systems, project control systems, pharmaceutical use history, health records, petrochemical data base, etc.

10. Known Sources of Information: Datapro Research Corporation, Delran, NJ

11. Probable Sources of Information: Specific Vendors of Systems: Software Ag.; Infodata Systems, Inc.; Computer Corp. of America; MRI Systems, Inc.

12. Bibliography:

2.a. CODASYL Systems Committee, "Feature Analysis of Generalized Data Base Management Systems," May 1971.
2.b. Datapro Research Corp, "A Buyer's Guide to Data Base Management Systems, "May 1976.
2.c. Koehr, G. J., et. al., "Data Management Systems Catalog." MITRE Corp. Report MTP-139, Jan. 1973.

13. Comments: "Self-contained" type of data base management systems is a classification distinguished by the CODASYL Systems Committee (2.a.). The key characteristic of the self-contained type is that the data definition, data retrieval and data input are all provided in such a way that conventional procedural programming is not required.

---

[1] Data gathered from bibliographic citation 2.b.

1. Designation: None (composite summary sheet)

2. Title: Composite summary sheet on Host Language Data Management Approach

3. Maintenance Authority:

4. Scope: The majority of host language type data base management systems are based on the CODASYL DBTG Specification. However, IMS (Information Management System), an IBM product, is of the non-CODASYL host language type. TOTAL, developed by Cincom Systems, came out in early 1968 before the CODASYL DBTG Specification; it too is a non-CODASYL host language type. The two packages alone account for approximately 50% of the current market.

5. Relationship to Other Standards: COBOL, FORTRAN, PL/I (base for host language)

6. Competitive Standards: CODASYL DBTG Specification

7. Standardization Status:

8. Implementation Status: IMS (Information Management System), developed and marketed by IBM, has gone through many evolutions and improvements. The earliest, IMS-I, operating on the IBM 360, appeared around 1969 and is based on another product developed jointly with North American Rockwell Company called DL/I (Data Language/I), which is a data description facility to describe and organize a hierarchically structured data base. IMS-I also provides an interface through which programmers can store data from the host language (COBOL). In 1971, IMS-VS was released to run on IBM 360/370s, under the VS (Virtual System) operating system. The IMS data base management package is the leading package among IBM 360/370 computer users. It is estimated that there were 1,000 installations at year-end 1975 (3.a.).

   TOTAL, developed by Cincom, is widely used and is second to IMS. TOTAL, in its initial release in 1968 was primarily a direct access data base management system. Facilities were soon added to process DBTG-like sets implemented with chain pointers. TOTAL is a host language system which can model the major data stuctures of the DBTG specification. It is considered non-CODASYL because Cincom claims TOTAL was implemented before CODASYL DBTG was published. TOTAL runs on the following machines: IBM 360/370, Honeywell 200/2000, UNIVAC Series 70 and 9400/9700, NCR Century Series, CDC 6000 Series, and IBM System/3 Model 10 and Model 15. TOTAL had 900 installations at year-end 1975 (3.a.).

9. Known Manufacturing Uses: Used in support of a large number of diverse applications, including applications in manufacturing, finance, and process control.

10. Known Sources of Information: 3.a. International Data Corp., "The Data Base Management Software Market on IBM 360/370 Systems," IDC # 1685, (May 1976) International Data Corp., 214 Third Ave., Waltham, MA 02154. For IBM information, see local representative. For information on TOTAL: Cincom Systems, Inc., 2300 Montana Ave., Cincinnati, OH 43211

11. Probable Sources of Information: Government users of IMS package are: Federal Reserve Board and Naval Materiel Command Support Activity. Users of TOTAL package are: Social Security Administration and Bureau of Labor Statistics.

12. Bibliography: None

13. Comments:

1. Designation: None

2. Title: Composite summary sheet on Relational Data Management Approach

3. Maintenance Authority:

4. Scope: It is still being researched within academia.

5. Relationship to Other Standards: None

6. Competitive Standards: CODASYL DBTG Specification

7. Standardization Status: None

8. Implementation Status: The concepts of n-ary relations as a tool for data base management systems dates from a 1970 paper by E.F. Cudd of IBM (4.a.). As yet, no large scale implementation exists. There were a number of early projects: MACAIMS developed at MIT and RDMS developed at General Motors. A large scale prototype data base management system, called System R, is presently under construction at IBM Research in San Jose. Another large scale attempt at constructing a relational prototype is the INGRES (Interactive Graphics and Retrieval System) of the University of California at Berkeley. INGRES is operational on a PDP-11/40 under the UNIX operating system.

   There are two commercially-developed data base software packages which claim to possess relational characteristics, but these are not considered true relational systems:

   - MAGNUM, developed by Tymshare, Inc., runs on IBM 360/370.
   - NOMAN, developed by National CSS, Inc., also runs on IBM 360/370.

9. Known Manufacturing Uses: There are no known manufacturing uses of the prototype university-based relational systems. Limited uses of the two commercial relational systems are: maintenance and inventory data, and invoice processing by a utility company.

10. Known Sources of Information: E.F. Codd, IBM Research Laboratory, San Jose, CA; For INGRES System contact: M. Stonebraker, University of California, Berkeley, CA; for MAGNUM System contact: Tymshare, 10340 Bubb Road, Cupertino, CA 95014; for NOMAN System contact: National CSS, Inc., 300 Westport Ave., Norwalk, CT 06851

11. Probable Sources of Information:

12. Bibliography:

4.a. Codd, E.F., "A Relational Model of Data For Large Shared Data Banks," Communications of the ACM 13.6 (June 1970), pp. 377-397.

13. Comments:

STANDARDS FOR COMPUTER SYSTEMS

SOFTWARE TESTING AND TOOLS

INTRODUCTION

The most thoroughly tested software pieces are usually the system components; compilers, editors, file management procedures, schedulers. This is hardly surprising considering that systems software is crucial to all aspects of a marketable system. Although this brief discussion will begin with an examination of typical systems testing procedures, there remain many aspects which arise mostly in applications. These topics are covered later in the discussion.

SYSTEM TESTING

Two aspects of system testing are easily visible even in the most cursory examination.

Performance measurement is fundamental to any operation involving expensive components such as cpu, discs and memory. The natural questions of What helps best? and Who pays for what? occur over and over. Each question involves some aspect of measurement (hence, testing) of a computer system. It is quite important that a system have a fine enough clock to allow meaningful measurements of user and system states. Without such a hardware clock a system is tuned only with difficulty, and (importantly) billing of services can become confused.

Language processor testing is another significant domain for system checkout and testing. Several checkout schemes have been mentioned earlier: for COBOL, FORTRAN, BASIC, and MUMPS. For languages such as these which are heavily used on their systems the investment in language test routines is quite justified, specifically since it also promotes program transferability among processors on distinct and different systems. Testing also assures conformance to an acceptable performance standard: that is, it shows a capability to handle required language features.

APPLICATIONS TESTING

There is a vast users' area over which the tag "testing" can be attached. For sake of convenience it is often the case that static (or textual) program features are treated as distinct from dynamic (or executable) behavior.

Static Testing

Static testing encompasses several labels. At this level of the lexicon, names must be accounted for, e.g., external system names should not conflict. A common problem along these lines occurs when one module in a system is rewritten and external storage maps are changed. The new maps may not agree with other module-maps unless some monitoring is made of storage definitions, and enforcements made to maintain consistency.

Syntactic testing is obvious, and every compiler does it with greater or lesser degrees of success. A number of points may be worth mentioning. First, the compilation facilities can serve as good enforcers of any system "standards" that are required for transportability or clarity. The compiler is an especially good and effective place for enforcement, in that failure to comply can imply failure to get any work done. Secondly, many compilers have extremely poor error message and diagnostic facilities. For some reason this is especially true for COBOL, and it seems to be more the fault of the compilers than the language. Some test programs have been written to test compiler diagnostics, but further work could be done on this aspect. The problem is easy to ignore but important to the everyday programmer. Thirdly, some languages such as early PL/1 have conventions, defaults design choices

which make compilation errors less apparent, and sometimes, almost invisible!

Semantic and functional testing are more wishes than current realizable technology. Questions arise on the meaning of primitive operations, machine dependencies (e.g., word size), and representations. Functional testing, or proof-of-correctness asks whether a program corresponds to its original specifications; this is an extremely difficult problem and little progress has been made of a practical nature.

Dynamic Testing

The first aspect of dynamic testing is one of correspondence. Has the correct problem been solved? Comparing actual runs against true answers may reveal the ultimate bug--having solved the wrong problem.

Performance measurement has been mentioned regarding the need for a good hardware clock for accounting and tuning. Similar requirements apply directly to applications programs. Three other points are worth mentioning:

(i) Program conversion and modularization--Help find "related" code to load together;

(ii) Learn variations in efficiency, isolate bottlenecks and non- critical parts;

(iii) Subsetting. Given cases of interest, chart "live" segments in a large program, thereby limiting the code to that of immediate interest.

The third area of dynamic testing could be called functional- empirical. One wants to thoroughly exercise a program [Huang, 1975]. In addition, the instrumented code can be tested against standard test cases to check that the latter are truly thorough. Weaknesses in test data are usually sins of omission, so parts of the program may not be used. This shows up immediately when program segment counters are zero.

Mathematical Software Testing

In CAM system utilization, any numerical errors arising during design or production management would be reflected in the finished product. It is important then that CAM engineers have confidence in the numerical performance of software as well as in the logical correctness of the programs. Although there are no standards for the numerical quality of mathematical software testing, there are testing practices that are somewhat de facto standards. We review some of these practices in this section.

Mathematical software according to Cody [1] denotes those computer programs that implement mathematical algorithms. Mathematical theorems are usually established about the theoretical nature of the algorithms and their convergence properties. In general, such results do not concern themselves with finite machine arithmetic. Very precise theoretical error bounds can usually be established for these theoretical function approximations [3].

However, when machine considerations such as word length, radix, floating or fixed point arithmetic are introduced, the theoretical algorithm must be restructured for a particular implementation in order not to lose the mathematical properties required to assure the theoretical error bounds. The restructuring of the theoretical algorithm for a particular implementation

might be referred to as the machine algorithm. By an implementation of a theoretical algorithm we mean the restructuring of the computation to make use of particular machine optimization of computations and the programming language used.

With respect to the testing of mathematical software there are two divisions. These are:

(1) Programming Languages Supporting Mathematical Functions
(2) Scientific/Engineering Support Mathematical Software

These divisions arise because, for language support mathematical software, i.e., mathematical function routines, there has emerged what appears to be a consensus approach, or de facto standard, for testing the mathematical function routines such as exponential, sine, cosine, etc. However, for general scientific software there are no general standards, but there are two approaches that might be referred to as test or benchmark problem sets and roundoff error analysis, (See Cody [1]). These latter approaches will not be considered here since we are concerned only with the mathematical function libraries that impinge on the language standards.

The simplest type of error testing is a direct comparison of computed function values against published tables. There are several difficulties with a naive application of this method. The first difficulty is the entry and storage of the large data set that would be needed in order to perform an exhaustive comparison. It would also require detailed checking of the input data to determine transcription error, and it would of course require editing of the data after entry. The next difficulty is the sparseness of the entries. Approximation procedures would have to be programmed to generate reference values to test the function subroutines at arguments between the table entries. The major difficulty is that these table-generated data points do not provide a sufficient sample of the behavior of the routine under test. Sample sizes of several thousand arguments have been used by some testers. Furthermore, table generated tests do not provide flexibility to the user.

Although the data table methodology is cumbersome and requires manual checking and preparation, the general idea is the same as the methodology used by the function testing community. The difference lies in the fact that the procedures for generating the comparison tables have been automated and allow a wider testing range and flexibility.

The most prominent scheme of accuracy checking is one that involves automatic tabular comparison where the standard table values are generated within the machine as needed. This usually requires the provision of a subroutine to compute standard values for a function to a precision greater than that of the routine under test. With such a routine it is possible to generate a table of comparison values automatically that can either be stored for future use or used immediately at the time of generation. This routine would generate high precision function values for specific arguments or for random arguments.

The emphasis in the mathematical testing community has been on the statistical sampling of the accuracy because of the objective ability to measure this. The approach has been widely used by a number of researchers (See Kuki [3], Cody [4], and Lozier [10] for examples.)

With regard to de facto testing methodologies, mathematical software divides itself into two classes. First the language support elementary function routines and second general scientific routines that are collected into libraries. There is a well- defined procedure for testing the language support function routines. However, there are a number of procedures that rely on performing arithmetics other than floating point that have been used to estimate numerical error. Since the process of developing scientific libraries, especially those that may be used to design critical parts, is a lengthy and expensive one, it is imperative to identify as soon as possible viable numerical software testing procedures and begin using them in evaluating user libraries.

SOFTWARE TOOLS

It is evident from prevailing experience and research that every software production project, regardless of complexity, must include a tool provisioning activity. The toolmaker faces several questions, to be answered in collaboration with his project manager: Is there a commonly accepted set of standardized tools applicable to every project?; What set of special tools can be identified for a project at the outset?; Are necessary tools already available as commercial packages with acceptable cost?; What are the economical approaches to creating special tools and modifying them as may be needed in the course of a project? Corresponding evidence shows there is inordinate difficulty in selecting tools tools from the marketplace shelf. Commercial items are available at reasonable cost, but there is essentially no standardization of tool capabilities. The number of suppliers and the diversity of packages confound the would-be buyer. But equally important, proprietary packages cannot be modified and tailored by the buyer since the source code is usually not delivered with purchase. Although a basic set of tools is identifiable for any project, it appears that that special modifications are warranted in many cases. Furthermore, a general expansion and integration of available tool functions would be well-advised to cope with the widely-recognized problem of software quality control. The following analysis tends to support a recommendation for standardization of basic tools at source code level, so that CAM software production can be conducted with a common set of tools amenable to user extension and specialization.

Types of Tools

The only standard tool for software production today is the high-level language compiler. This statement applies the traditional understanding that a standard is a formal specification produced by a recognized professional group for nearly universal application. Yet, national and international standardization of compilers has only addressed programming language definition, ignoring crucial capabilities such as the form and content of output listings, accuracy and scope of diagnostic messages, debugging features, and interactive and batch modes.

Even so, use and economics of tool design have lead to commonly discernible types. These tools cannot be called defacto standards, for they reflect only similar purpose and function, and not by any means a near equivalence of capability brought about by uniform commercial demand. The following common types have been determined from a survey of commercial packages. Omitted are compilers, assemblers, data base management systems, utility routines, application programs or libraries (e.g. mathematical routines). Also excluded are replacement packages for software normally offered by a hardware

vendor, such as operating systems and I/O access methods. Common tools are: Abort diagnoses--provide full or selective dumps; Breakpoint control--for interactive debugging; Cross reference generator; Data auditor/catalog--analyzes data relationships; Error analysis and recovery--intercept selected abnormal terminations; File or library manager--centralized retrieval and update; Flowchart generator; Program auditor--checks conformance of programs; Program execution monitor--see testing sections, above; Program formatter/documentor-- Rearranges and structures source text; Project manager--scheduling and production aid; Resource monitor--accounting information; Shorthand or macro expander--may also include decision table expansion; Source level translator--e.g. RPG to COBOL; Test data generator; Test simulator--simulates execution and flow, allow user decisions in testing; Text editor.

Minimum Essential Tools

Contemporary experience and practioners' concensus are sufficient to recommend some tools as essential for almost any software development project. Exceptions may arise if a computer has unusual architecture or limited capabilities (e.g. no mass storage). Minicomputer systems are generally included, particularly since the UNIX system [Ritchie] has demonstrated that a highly effective, interactive programming support system is practical on a low-cost minicomputer.

It is recommended that in general program development be done with support of an interactive computer system. Interactive support increases productivity throughout the changes, debugging, and testing that characterize most projects.

The primary tool is the compiler for the high-level programming language. Again, experience has amply proven enhancements of programming productivity using high-level languages. Only selected procedures critical to system performance need to be assembly-language coded for extreme execution speed. Other essential tools are recommended as a minimum complement for most projects:

Text editor - For entering, correcting, and modifying such texts as program specifications and design documentation. Requires a facility for online storage and recall of named text units for inspection, printing or editing.

Program editor - For entering, correcting, and modifying program texts. With free-form programming languages, one editor could serve both as text and program editor.

Program librarian - For storing all program texts, associated job control statements, common data definitions, and test data, and maintaining a chronological record of modifications between distinct versions. Includes appropriate access controls for members of the project group.

Debugger - For analyzing program behavior during execution on test data input, and deriving execution statistics and traces to help correlate program output with the results of individual high-level language statements.

Project manager - For recording chronologically the activity of the individual project members on defined program modules and deliverables of the project. Standard specifications of functions for each tool type appear feasible and desirable, and would assist those who undertake toolmaking without benefit of prior study and experience. Yet it is clear that

individual projects often may need to create special features that would not be available in standardized tools. Various project requirements or circumstances may dictate such specializations. For instance, large projects with many personnel especially would benefit from extensions to automatically enforce unique design standards and practices that are difficult to ensure through personal communications and code inspections.

Desirable specializations may range in difficulty from minor extensions of extant tools to new composite tools formed by integrating and refining several distinct packages. Both of these cases require the original tool's source code--ideally in a system-standard high level language--and thorough documentation of course. The latter case also requires that the building block tools be carefully designed, with flexible interfaces and modular design, permitting extensive modifications with relative ease.

RECOMMENDATIONS

It is appropriate therefore to recommend studies and development on CAM programming tools, with the following goals:

1. to make widely available a set of CAM building block tools, with standard designs and source code in CAM-system high level language;

2. to evaluate alternatives for interfaces and modular design that would support major modifications of tools without loss of efficiency and performance; and

3. to develop guidelines for rapid and reliable specialization of tools from available building blocks, based upon the characteristics of CAM projects most benefitting from special tools.

REFERENCES

(Computer Validation)

Federal Property Management Regulation 101-32.1305-a Validation of COBOL Compilers.

NBS Special Publication 399, Vols. 1-3, "NBS FORTRAN Test Programs."

MDC/29, MUMPS Validation Program User Guide.

(General)

NBS Technical Notes 874, "Software Testing For Network Services"; 849, "A FORTRAN Analyzer"; 800, "Computer Networking" (above some approaches to quality assurance).

K. V. Hanford, "Automatic Generation of Test Cases," IBM Systems Journal 9, 4(1970), pp. 242-257.

J. C. Huang, "An Approach to Program Testing," Computing Surveys 7, 3(September 1975), pp. 113-128.

(Mathematical Software Testing)

W. J. Cody, "The Evaluation of Mathematical Software," Program Test Methods, EWilliam C. Hetzel, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973), 121.

C. T. Fike, Computer Evaluation of Mathematical Functions, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1968).

H. Kuki, "Mathematical Function Subprograms for Basis System Libraries -- Objectives, Constraints and Trade-Off," Mathematical Software, Academic Press, NY, p187-199.

W. J. Cody, "Performance Testing of Function Subroutine," AFIPS ConProc., Vol. 34, 1969.

N. A. Clark, W. J. Cody, K. E. Hillstrom and E. A. Thieleker, "Performance statistics of the FORTRAN IV (H) Library for the IBM Systems/360," Report ANL7321, Argonne National Laboratories (1967).

C. L. Lawson, "Study of the Accuracy of the Double Precision Arithmetic Operations on the IBM 7094 Computer," JPL Tech. Memo #33-142, Jet Propulsion Laboratory, Pasadena, 1963.

D. W. Lozier, L. C. Maximon, W. L. Sadowski, "A Bit Comparison Program for Algorithm Testing," The Computer Journal, Vol6, N2, pp. 111-117.

A.C.R. Newbery, Anne P. Leigh, "Consistency Tests for Elementary Functions," AFIPS ConProc., Vol. 39, 1971.

W. J. Cody, "Software for the Elementary Functions," Mathematical Software, R. Rice Ed., Academic Press, NY, 1971.

D.W. Lozier, L. C. Maximon, and W. L. Sadowski, "Performance Testing of a FORTRAN Library of Mathematical Functions Routines-- A Case Study in the Application of Testing Techniques," Journ. of Res., NBS, B. Math. Sce., Vol. 77B, Nos. 3 & 4, July- December 1973.

(Software tools)

Brooks, Frederick P., Jr. The mythical man-month, Addison-Wesley Publishing Company, Reading, Mass., 1975, p.128.

Reifer, Donald J., "Automated aids for reliable software," Proceedings of the International Conference on Reliable Software, SIGPLAN Notices 10, 6(June 1975), pp. 131-140.

Ritchie, Dennis M. and Thompson, Ken. "The UNIX Time-sharing system," Comm. ACM 17, 7(July 1974), pp. 365-375.

Van Dam, Andries, and Rice, David E. "On-line text editing: a survey," Computing Surveys 3, 3(Sept. 1971), pp. 103-105.

Wichmann, B.A. "A syntax checker for ALGOL 60," NPL Report NAC 53, August 1974, Division of Numerical Analysis and Computing, National Physical Laboratory, Teddington, Middlesex, England.

SUMMARY DATA SHEETS

The following data sheets summarize the standardization activities which apply to Software Testing and Tools.

1. Designation: None

2. Title: Composite summary sheet on Computer Software (Testing and Validation)

3. Maintenance Authority: COBOL: (U.S. Navy); FORTRAN: (NBS); BASIC (NBS), MUMPS (MUMPS Development Committee)

4. Scope: Test for compiler compliance with 1968 COBOL and 1966 FORTRAN standards, and for ANSI proposed Minimal BASIC and MUMPS standards.

5. Relationship to Other Standards: Complement present or pending standards for COBOL, FORTRAN, BASIC and MUMPS.

6. Competitive Standards:

7. Standardization Status: COBOL 1968 and FORTRAN 1966 are available; Minimal BASIC and MUMPS are pending standards.

8. Implementation Status: Test data available from Federal Testing Service or NTIS. (BASIC) - NTIS & NBS

9. Known Manufacturing Uses: System procurement and quality control aids.

10. Known Sources of Information: Mrs. Frances E. Holberton (FORTRAN), Ms. Mabel V. Vickers (COBOL). Dr. David E. Gilsinn (BASIC), NBS, (301) 921-3491, Dr. Jack Bowie (MUMPS), Massachusetts General Hospital, (617) 726-3937.

11. Probable Sources of Information:

12. Bibliography: Federal Property Management Regulations 101-32.1305a, Validation of COBOL Compilers; NBS Special Publication 399, Vol. 1-3 "NBS FORTRAN Test Programs." NBS-IR (in draft) "Proposed NBS Minimal BASIC Test Programs." MDC/29, MUMPS Validation Program User Guide.

13. Comments: COBOL testing is used in acquisition of compilers for the Federal Govenment.

1. Designation: None

2. Title: Numerical Testing and Validation of Mathematical Software

3. Maintenance Authority: None, although some certification of algorithms and programs is given in the ACM algorithms collection and both IMSL (International Mathematical and Statistical Libraries, Inc.) and Argonne National Laboratory have produced and continue to test specialized mathematical libraries.

4. Scope: To define methods and guidelines to evaluate the numeric properties (such as accuracy) and ascertain this domain of mathematical software (that is, identify clearly the class of problems that a program solves). Another measure addressed is the speed of the programs. These qualities are referred to as the performance evaluation of mathematical software.

5. Relationship to Other Standards: Error analysis of mathematical software depends strongly on numeric representations. This subject is addressed from the point of view of character strings by ANSI X3.42-1975, American National Standard for the representation of numeric values in character strings for information interchange. Extrinsic mathematical functions routine capability is specified by the proposed ANS FORTRAN, BSR X3.9 and proposed ANS Minimal BASIC, BSR X3.60.

6. Competitive Standards: There are no standards for evaluations of mathematical software although there are a number of competitive approaches to evaluating the effect of error propagations in mathematical software (see comments below).

7. Standardization Status: None, although there is a recently organized IFIPS Working Group on Numerical Software (WG 2.5). The ACM SIGNUM, SIGMAP and SIGSAM are concerned with mathematical software.

8. Implementation Status: Ad hoc evaluation tools have been used by IMSL and Argonne. IBM has implemented in S/360 a quality mathematical function library developed at the University of Chicago. UNIVAC has also used a quality evaluation methodology to enhance the accuracy of its mathematics libraries.

9. Known Manufacturing Uses: To develop and evaluate the mathematical function software libraries peripheral to the programming languages FORTRAN, ALGOL, PL/I, BASIC, etc. IMSL uses evaluation techniques to generate quality mathematical software for engineering and scientific use.

10. Known Sources of Information: David E. Gilsinn, NBS, (301) 921-3491; Dan Lozier, NBS, (301) 921-2631.

11. Probable Sources of Information: IMSL, Argonne National Lab., Jet Propulsion Lab.

12. Bibliography: This bibliography is by no means exhaustive, but it covers a fair sampling of the literature dealing with tools and approaches to quality testing.

(1) W. J. Cody, "The Evaluation of Mathematical Software," Program Test Methods, Ed., William C. Hetzel, Prentice-Hall, Inc., Englewood Cliffs, NJ (1973), p. 121.

(2) C. T. Fike, Computer Evaluation of Mathematical Functions, Prentice-Hall, Inc., Englewood Cliffs, NJ, (1968).

(3) H. Kuki, "Mathematical Function Subprograms for Basic System Libraries -- Objectives, Constraints and Trade-Offs," Mathematical Software, Academic Press, NY, pp. 187-199.

(4) D. W. Lozier, L. C. Maximon, and W. L. Sadowski, "Performance Testing of a FORTRAN Library of Mathematical Functions Routines -- A Case Study in the Application of Testing Techniques," Journ. of Res., NBS, B. Math. Sci., Vol. 77B, Nos. 3 & 4, July - December 1973.

13. Comments: There are a number of proposed approaches to quality evaluation of mathematical software. There has been no concerted effort to determine which methods are appropriate to which programs. First, a straightforward approach to determining accuracy is to run a program on a standard set of problems and to compare the computed results against the known results. For certain problems this method can give meaningful and useful measures of the error generated within the routine. In particular, the accuracy of mathematical function routines is frequently determined by using the comparison approach. This method of analysis is sometimes called forward error analysis. Second, when a solution of a problem involves arrays of numbers, then another approach to quality evaluation is to show that the computed results are the exact solutions to a perturbation of the original problems and to measure that perturbation. This method is sometimes called backward error analysis. Third, the domain of the problem is evaluated. That is, the problems that a particular piece of mathematical software solves are determined. This method is related to method one, but requires a classification and codification of problem sets. Fourth, another approach to studying error propagations is statistical in the sense that the errors incurred for each operation are assumed to be random variables with some specified distribution. The error analysis can then proceed, using the tools of probability theory in order to determine the resulting error distributions, by computing the individual error random variables. Finally for some problems a-priori error bounds can be estimated by the use of inequalities and bounds for the accumulated error at each step of a program.

STANDARDS FOR COMPUTER SYSTEMS

DOCUMENTATION STANDARDS

FIPS PUB 38 DOCUMENTATION REQUIREMENTS

PROGRAM COMPLEXITY	Software Summary	Users Manual	Operations Manual	Program Maintenance Manual	Test Plan	Functional Requirements Document	System/Subsystem Specification	Test Analysis Report	Program Specification	Data Requirements Document	Data Base Specification
One shot-single use programs*	X										
Small limited purpose programs	X	X									
Multipurpose-general programs	X	X	X	X	X			**		***	***
Large scale-multiuser programs	X	X	X	X	X	X		**		***	***
Large scale systems Common data base Multi application programs	X	X	X	X	X	X	X	X		***	***
Totally integrated systems Multi disipline users Multi contractor development	X	X	X	X	X	X	X	X	X	X	X

* Additional documentation may be required by local policy.
** Test Analysis Report may be prepared informally.
*** May or may not be needed depending upon project.

Figure 1 Project Complexity Influences Documentation Requirements

INTRODUCTION

One of the recent developments in software has been the emphasis on control of the complete generation cycle, and an examination of the dependencies that should exist in this cycle. Documentation preparation should be treated as a continuing effort evolving from preliminary requirements-drafts, through change and reviews to the final documentation and continuation documentation of the delivered software products. Since clear and complete documentation is a keystone for portable and maintainable software modules, definitive guidelines for its preparation are of vital importance to the Air Force program. A documentation administrator should be assigned to work with the contracting officer to define and enforce requirements for clear documentation including source code of system components which should be Air Force property. The documentation administrator should have a clear idea of what is in the CAM system; therefore a model should be constantly kept of system components to catch any omissions. The model should be available to users for feedback on its adequacy and degree of coverage in current documentation.

The extent of documentation should depend on the size, complexity and value of the project. Special requirements are necessary for certain well-defined components; for example, interactive processors (editors, language translators, networking modules) should have available on-line "help" files to show how to use them. A user should be able to run these interactive components without shutting off his terminal, but rather, using it to advantage.

Documentation should spell out clearly the specific software compatibilities and incompatibilities; i.e. does compiler X read files of type Y. In addition, compatibilities should be spelled out as specific mandatory requirements in early stages of design documentation, and the design requirements written to preclude as many undesirable conflicts as possible. The extent, detail, and formality of software documentation must be included in all contractual arrangements for software procurement.

Automated aids for development and maintenance of ICAM documentation would be a great assistance to both contractors and the documentation administrator. The development of such aids should be considered as an early ICAM project.

SOFTWARE DOCUMENTATION GUIDELINES

In reviewing various guidelines for software documentation a growing tendency is noted toward the development of a full life cycle management system for the software creation process. NASA documentation standards for part of the Appollo project are a good example. Entitled "Procedures for Management Control of Software Development for Appollo" the guidelines address each functional step from requirements analysis to coding, testing and maintenance. Specifications are made for the documentation required for each functional step.

Considerable progress has been made in Federal standards for software documentation: FIPS PUB 38 is prehaps best suited for the development of large systems, providing as it does a checklist of items worthy of detailed attention in a project. The documentation categories of FIPS PUB 38 begin with functional requirements, pass through the natural stages of a project, and end with test plans and analyses. The standard recognizes that not all documentation categories are needed for every project. Rather as the size, complexity and visibility of a project increases so does its need for more extensive documentation. Figure 1 has been abstracted from FIPS PUB 38 to emphasize this concept.

CAM-I has established documentation standards to assure the availability of detailed information on the software products developed. The standard defines the structure, content and use of ten separate documents to be develope within each software project. The functional content of the CAM-I specified documents is quite similar to those in FIPS PUB 38 although the latter are more completely defined.

Three other differences are noted:

FIPS PUB 38 breaks out separate Test Plans and Test Analysis Reports rather than embedding the functions in other documents. These two are very important for validating the applications module and for assuring that the coding meets portability requirements.

CAM-I describes a Project Status Report necessary for good Air Force management control of the software development process. Also included is a Project Prospectus which ICAM may find quite useful as a brief descriptive outline of a module that can be sent to all prospective users or included in press releases, etc.

FIPS PUB 38 contains an in depth specification for defining a module's interdependence with various data bases. In the distributed processing, integrated systems environment envisioned for ICAM due attention must be given to these data requirements.

A number of miscellaneous recommendations exist for bits and pieces of program development. For example, there is FIPS PUB 24 on flowcharting. In addition, a large number of texts and articles exist. Yourdon, pages 23-24, provides some excellent common sense on documentation and maintenance of program modules, including advice on the use of variables in original codings.

The Department of Defense has issued in December 1972 a manual on Automated Data Systems Documentation Standards which has been implemented by all three services.

Smaller programs and projects in the CAM undertaking may find the work of the American Nuclear Society (244 East Orden Ave., Hinsdale, Illinois 60521) useful. Two documents of the Society are referenced at the end of this chapter.

RECOMMENDATIONS

1. The Air Force should extend FIPS 38 in order to have more detailed guidelines for computer program documentation and to software. The guidelines should use the framework of FIPS 38, and may incorporate useful sections of the CAM-I, NASA, and American Nuclear Society publications.

2. The Air Force should establish a documentation administrator to specify and maintain system and software documentation. Since there are many disparate CAM interests, a tight rein on documentation in the first development stages combined with industry participation (as practical) could promote a clarification of standard CAM system components and procedures.

3. Automated aids for production and maintenance of documentation should be considered as early program efforts.

REFERENCES

(1) Guidelines for Documentation of Computer Programs and Automated Data Systems. Federal Information Processing Standards Publications FIPS PUB 38, February 15, 1976. US Department of Commerce, National Bureau of Standards.

(2) Guidelines for the Documentation of Digital Computer Programs. Americal Nuclear Society, ANS-10.3. (Also ANSI N413).

(3) ANS Standard. Recommended Practices to Facilitate the Interchange of Digital Computer Programs. ANS STD 3-1971.

(4) Flowchart Symbols and Their Usage in Information Processing. FIPS PUB 24, June 30, 1973. (Same as ANSI X3.5-1970)

(5) Daniel D. McCracken. "How to write a readable FORTRAN program." DATAMATION (Oct. 1972), pp. 73-77.

(6) E. Yourdon. Techniques of Program Structure and Design, Prentice-Hall, Inc., (Englewood Cliffs, N.J., 1975).

(7) Department of Defense Automated Data Systems Documentation Standards Manual, 4120.17M, December 1972.

(8) CAM-I Standard for Computer Program Documentation, STD-73-SC-01, May 1973.

SUMMARY DATA SHEETS

The following data sheets summarize those standards which apply to Software Documentation.

1. Designation: FIPS PUB 38

2. Title: Guidelines for Documentation of Computer Programs and Automated Data Systems

3. Maintenance Authority: NBS (FIPS TG 14)

4. Scope: These software guidelines provide a basis for determining the content and extent of documentation for computer programs and automated data systems.

5. Relationship to Other Standards: FIPS PUB 30 (subset)

6. Competitive Standards: None

7. Standardization Status:

8. Implementation Status: Published on Feb. 15, 1976.

9. Known Manufacturing Uses: These documentation guidelines are applicable to all computer software development and use applications.

10. Known Sources of Information: James Gillespie, USN, FIPS TG 14 Chairman, (202) 695-0680; Thomas Kurihara, Department of Agriculture, FIPS TG 14 Vice-Chairman, (202) 447-6261; Bea Marron, NBS, FIPS TG 14 Executive Secretary, (301) 921-3491

11. Probable Sources of Information:

12. Bibliography:

FIPS PUB 38, February 1975, Guidelines for Documentation of Computer Programs and Automated Data Systems

Automated Data System Documentation Standards Manual, Department of Defense Manual 4120.17-M, December 1972

Computer Program Documentation Guideline, National Aeronautics and Space Administration, NHB-2411.1, July 1971.

13. Comments: Documentation of computer software provides information to support the effective management of ADP resources and to facilitate the interchange of information. It serves to:

- Provide managers with technical documents to review at the significant development milestones, to determine that requirements have been met and that resources should continue to be expended.
- Record technical information to allow coordination of later development and use/modification of the software.
- Facilitate understanding among managers, developers, programmers, operators, and users by providing information about maintenance, training, changes, and operation of the software.
- Inform potential users of the functions and capabilities of the software, so that they can determine whether it will serve their needs.

These guilelines were prepared to improve the quality and consistency of software documentation.

1. Designation: FIPS PUB 30, Standard Form 185

2. Title: Software Summary for Describing Computer Programs and Automated Data Systems

3. Maintenance Authority: NBS

4. Scope: This standard software summary form is used in documenting summaries or abstracts of programs and/or automated data systems that are developed or acquired by Federal departments and agencies.

5. Relationship to Other Standards: FIPS PUB 38 (superset)

6. Competitive Standards: None, but ANSI X3K7 was organized in October 1975 to develop a computer program abstract.

7. Standardization Status: Published June 30, 1974

8. Implementation Status: On Feb. 25, 1976, a Federal Property Management Regulation was announced which requires the use of this standard form for reporting "common use software" to a new Federal Software Exchange Center.

9. Known Manufacturing Uses: This documentation standard is applicable to all computer software development and use applications.

10. Known Sources of Information: James Gillespie, FIPS TG 14 Chairman; Thomas Kurihara, FIPS TG 14 Vice-Chariman; Bea Marron, FIPS TG 14 Executive Secretary.

11. Probable Sources of Information:

12. Bibliography: FIPS PUB 30, June 1974, Software Summary for Describing Computer Programs and Automated Data Systems

13. Comments: This standard, a one-page form with instructions on the back, is intended for succinctly describing computer programs and automated data systems for identification, reference, and dissemination purposes.

1. Designation: FIPS PUB 24

2. Title: Flowchart Symbols and their Usage in Information Processing

3. Maintenance Authority: NBS (ICST)

4. Scope: This publication establishes standard flowchart symbols and specifies their use in the preparation of flowcharts in documenting information processing systems.

5. Relationship to Other Standards: Same as ANSI X3.5 - 1970, American National Standard Flowchart Symbols and their Usage in Information Processing.

6. Competitive Standards: None

7. Standardization Status: Published June 30, 1973. The ANSI standard was approved Sept. 1, 1970 as a revision of USA Standard X3.5 - 1968.

8. Implementation Status: This standard applies to any Federal information processing operation where symbolic representation is desirable to document the sequence of operations and the flow of data and paperwork.

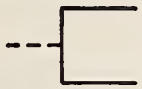
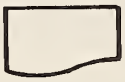
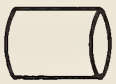
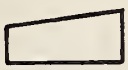
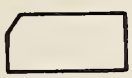
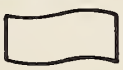
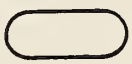
9. Known Manufacturing Uses: Applicable for all systems and software documentation.

10. Known Sources of Information: Mr. Harry S. White, Jr, NBS, (301) 921-3157.

11. Probable Sources of Information:

12. Bibliography: FIPS PUB 24, June 1973, Flowchart Symbols and their Usage in Information

13. Comments: A FIPS Jiffy Template (#673) of Flowchart Symbols, which conforms to FIPS PUB 24, is now available.

ANNOTATION  FOR THE ADDITION OF DESCRIPTIVE COMMENTS OR EXPLANATORY NOTES AS CLARIFICATION	AUXILIARY OPERATION  OFFLINE OPERATIONS PERFORMED ON EQUIPMENT NOT UNDER DIRECT CONTROL OF THE CENTRAL PROCESSOR.	CONNECTOR  A JUNCTION IN THE LINE OF FLOW	CORE  I/O FUNCTION IN WHICH THE MEDIUM IS MAGNETIC CORE (USE AUXILIARY OPERATION SYMBOL)
DECISION  POINTS IN A PROGRAM WHERE SEVERAL PATHS MAY BE POSSIBLE, BASED ON VARIABLE CONDITIONS	DISPLAY  I/O FUNCTION IN WHICH THE INFORMATION IS DISPLAYED FOR HUMAN USE AT TIME OF PROCESSING.	DOCUMENT  I/O FUNCTION IN WHICH THE MEDIUM IS A DOCUMENT	EXTRACT  REMOVAL OF ONE OR MORE SPECIFIC SETS OF ITEMS FROM A SINGLE SET OF ITEMS.
INPUT/OUTPUT  MAKING AVAILABLE INFORMATION FOR PROCESSING OR RECORDING PROCESSED INFORMATION	MAGNETIC DISK  I/O FUNCTION IN WHICH MEDIUM IS MAGNETIC DISK	MAGNETIC DRUM  I/O FUNCTION IN WHICH MEDIUM IS MAGNETIC DRUM	MAGNETIC TAPE  I/O FUNCTION IN WHICH THE MEDIUM IS MAGNETIC TAPE
MANUAL INPUT  I/O FUNCTION IN WHICH THE INFORMATION IS ENTERED MANUALLY AT THE TIME OF PROCESSING.	MANUAL OPERATION  ANY OFFLINE PROCESS GEARED TO THE SPEED OF A HUMAN BEING	MERGE  COMBINING TWO OR MORE SETS INTO ONE SET	OFFLINE STORAGE  REPRESENTS ANY OFFLINE STORAGE OF INFORMATION REGARDLESS OF THE MEDIUM
ONLINE STORAGE  REPRESENTS AN I/O FUNCTION UTILIZING MASS STORAGE THAT CAN BE ACCESSED ON LINE	PRE-DEFINED PROCESS  A NAMED PROCESS CONSISTING OF ONE OR MORE OPERATIONS OR PROGRAM STEPS, SPECIFIED ELSEWHERE. (SUBROUTINE)	PREPARATION  A GROUP OF INSTRUCTIONS WHICH MODIFY, UPDATE, CORRECT OR OTHERWISE CHANGE THE PROGRAM	PROCESS  REPRESENTS THE PROCESS OF EXECUTING A DEFINED OPERATION OR GROUP OF OPERATIONS
PUNCHED CARD  I/O FUNCTION IN WHICH THE MEDIUM IS PUNCHED CARDS INCLUDING MARK SENSE CARDS, STUB CARDS	PUNCHED TAPE  I/O FUNCTION IN WHICH THE MEDIUM IS PUNCHED TAPE	SORT  ARRANGING A SET INTO A PARTICULAR SEQUENCE (USE EXTRACT AND MERGE)	TERMINAL  A POINT AT WHICH INFORMATION CAN ENTER OR LEAVE

1. Designation: Data Element Standards (composite summary sheet)

   FIPS PUB 4 - Calendar Date
   FIPS PUB 5-1 - States and Outlying Areas of the United States
   FIPS PUB 62 - Counties and County Equivalents of the States of the United States
   FIPS PUB 8-4 - Standard Metropolitan Statistical Areas
   FIPS PUB 10-1 - Countries, Dependencies, and Areas of Special Sovereignty
   FIPS PUB 19 - Guidelines for Registering Data Codes

2. Title:

3. Maintenance Authority: NBS

4. Scope: Data Element Representations and Codes

5. Relationship to Other Standards: (See individual publications)

6. Competitive Standards:

7. Standardization Status: (See individual publications)

8. Implementation Status: (See individual publications)

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. Harry S. White, Jr., NBS, (301) 921-3157

11. Probable Sources of Information:

12. Bibliography: (See individual publications)

13. Comments: These data elements are not directly relevant to CAM; on the other hand, if any of these data elements are utilized in a CAM system, the standard formats should be followed to allow transferability. In addition, these standards offer models for standardizing the data elements that are directly relevant to CAM.

1. Designation: FIPS PUB 11/ANSI X3.12-1970

2. Title: Vocabulary for Information Processing

3. Maintenance Authority: NBS

4. Scope: This Vocabulary is a reference document for general use throughout the Federal Government to help promote a common understanding of information processing activities.

5. Relationship to Other Standards: Same as ANSI X3.12-1970

6. Competitive Standards: None known.

7. Standardization Status: Published Dec. 1, 1970. The ANSI Standard was approved Feb. 18, 1970 as a revision of U.S.A. Standard X3.12-1966.

8. Implementation Status:

9. Known Manufacturing Uses: Applicable to all information processing activities.

10. Known Sources of Information: Mr. Harry S. White, Jr., NBS, (301) 921-3157; Ms. Josephine Walkowicz, NBS, (301) 921-3485.

11. Probable Sources of Information:

12. Bibliography:

13. Comments: An "American National Dictionary of Information Processing" will be issued as an ANSI Technical Report on or about October 1, 1976.

STANDARDS FOR COMPUTER SYSTEMS

MEDIA STANDARDS

INTRODUCTION

Of basic importance to any computer system is the media on which computer readable information is prepared, stored and exchanged. Adherence to formal media standards is a simple economic principle. Consider a computer program of 20 thousand language statements punched onto a nonstandard card deck. A lengthy and costly keypunch task would await anyone wishing to use this program. Fortunately the industry has pretty well standardized the media in common use; punched cards, magnetic tape, punched paper tape, and disk packs.

PUNCHED CARDS

These are the familiar 3 1/4 x 7 3/8 inch heavy paper cards that are as common to a computer programmer as nails to a carpenter. ANSI Standard X3.11 describes the physical attributes and quality of these while ANSI Standard X3.21 defines the size and location of the rectangular holes. It should be remembered that for punched cards to be readily transportable it is necessary that a specification be made to the coding of characters on the card. See the Hollerith Punched Paper Card Code.

MAGNETIC TAPE

Specifications for 1/2 inch wide magnetic tape and reels are given in ANSI Standard X3.40 while format and recording data are detailed in ANSI X3.14 and X3.22. Together these standards enable mechanical, magnetic and recording format interchangeability of data among various systems and equipment utilizing the American National Standard Code for Information Exchange, X3.4. Magnetic tape written in this manner provides the best means of exchanging computer data. It is also a convenient method for use in archieval storage and distribution of ICAM developed software. A recent DATAMATION article details recommended procedures for maintaining good quality control over a magnetic tape based archieval record storage facility.

MAGNETIC DISK PACKS

ANSI Standard X3.46-1974 provides the general, magnetic and physical requirements for interchangeability of six-disk packs among various disk drives. However, ANSI leaves the formating and recording of data to the manufactures' discression. As a result absolute compatibility is not guaranteed. The six-disk pack is giving way to a twelve-disk pack for which an ANSI standard is yet unavailable.

PUNCHED PAPER TAPE

Two ANSI Standards exist for describing punched paper tape. This media is most extensively used for the numerical control of machine tools. However, some use is seen for data storage in minicomputer applications. ANSI X3.29 details the physical characteristics and acceptance test procedures for one inch wide and eleven-sixteenths inch wide unpunched, oiled paper tpae. ANSI X3.18 covers the physical dimensions of the tape as well as its perforations. Caution is advised that to insure portability of paper tapes one must specify the format and coding of the data as well as the physical characteristics.

When used in NC applications, punched paper tape has been justly described as the weakest link in the process. This is a result of the many maintenance problems that exist on paper tape punches and readers. It would be unfortunate if the Air Force perpetuated the use of paper tape in large scale CAM Systems. Direct wire link is today far more efficient, reliable, and versatile.

RECOMMENDATIONS

1. Magnetic tape should be used as the primary means of exchanging and storing computer readable information.

2. All magnetic tapes should conform to ANSI X3.40, X3.14 and X3.22 Standards.

3. The use of punched paper cards should be deemphasized as it is an inefficient media of information storage. However, where it is necessary to produce cards the ANSI X3.11 and ANSI X3.21 Standards should be specified.

4. The use of punched paper tape should be avoided for transmitting NC data. Direct wire link from computer to machine controller provides a higher quality system configuration.

REFERENCES

(1) Specification for General Purpose Paper Cards for Information Processing, ANSI X3.11, October 1969, American National Standards Institute, Inc., 1430 Broadway, New York City, New York 10018.

(2) Rectangular Holes in Twelve-Row Punched Cards, ANSI X3.21, October 1976, American National Standards Institute.

(3) Recorded Magnetic Tape for Information Interchange;
200 CPI, NRZI, ANSI X3.14, December 1972
800 CPI, NRZI, ANSI X3.22, December 1972
9 Track 200 CPI, NRZI, ANSI X3.40, February 1976
9 Track 800 CPI, NRZI, ANSI X3.40, February 1976
9 Track 1600 CPI, PE, ANSI X3.40, February 1976
American National Standards Institute.

(4) One Inch Perforated Paper Tape for Information Interchange, ANSI X3.18, March 1974, American National Standards Institute.

(5) Eleven-Sixteenths-Inch Perforated Paper Tape for Information Interchange, ANSI X3.19, March 1974, American National Standards Institute.

(6) Specification of Properties of Unpunched Oiled Paper Perforator Tape, ANSI X3.29, May 1971, American National Standards Institute.

(7) Unrecorded Magnetic Six-Disk Pack, ANSI X3.46-1974, May 1974, American National Standards Institute.

(8) Archival Data Storage, Sidney B. Geller, DATAMATION, October 1974, pp 72-80.

SUMMARY DATA SHEETS

The following Data Sheets summarize those standards which apply to Computer Media.

1. Designation: MEDIA

2. Title: Punched Cards (80-column "IBM" Type)

3. Maintenance Authority: NBS/ANSI/ISO/EIA

4. Scope: Card stock, card and hole dimensions, Hollerith coding

5. Relationship to Other Standards:

ISO Recommendation or Draft Recommendation	Related National Standard	Related Federal Standard (FIPS)
ISO 1679 Representation of ISO 7-Bit Coded Character Set in 12-Row Punched Cards	X3.26-1969 Hollerith Punched Card Code	FIPS PUB 14 Hollerith Punched Card Code
ISO 1681 Specifications for Unpunched Paper Cards	X3.11-1969 Specification for General Purpose Paper Cards for Information Processing	
ISO 1682 Demensions and Locations of Rectangular Punched Holes in 80-Column Punched Paper Cards	X3.21-1967 Rectangular Holes in 12-Row Punched Cards	FIPS PUB 13 Rectangular Holes in 12-Row Punched Cards

6. Competitive Standards:

7. Standardization Status: Summarized in 5 above.

8. Implementation Status: Widely implemented in IBM and other computer installations.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. H. F. Ickes, IBM, (914) 463-9779

11. Probable Sources of Information: Mr. Robert M. Brown, Vice-Chairman of ANSI X3, CBEMA, (202) 466-2288.

12. Bibliography: See 5 above.

13. Comments: Hollerith cards have 80 columns and 12 rows of rectangular holes. They are not compatible with the round hole 90-column cards formerly marketed by UNIVAC, nor with the 96-column cards introduced by IBM with the System 3.

1. Designation: MEDIA

2. Title: Magnetic Tape (1/2 inch, 9 track, Digital)

3. Maintenance Authority: NBS/ANSI/ ISO/ EIA

4. Scope: Unrecorded tape stock, recording formats, bit densities, coding, hubs, reels.

5. Relationship to Other Standards:

ISO Recommendation or Draft Recommendation		Related National Standard	Related Federal Standard (FIPS)
ISO R961	Implementation of the 6 and 7-Bit Coded Character Sets on 7-Track 12.7 mm (1/2 in) Magnetic Tape.		
ISO 962	Implementation of the 7-Bit Coded Character Set on 9-Track, 12.7 mm (1/2 in) Magnetic Tape.		
ISO R1858	General Purpose Hubs and Reels with 76 mm (3 in) Centrehole for Magnetic Tape Used in Interchange Information Applications	RS-346 Type A Hubs and Reels and Magnetic Tape	
ISO R1859	Un recorded Magnetic Tapes for Instrumentation Applications --General Dimensional Requirements		
ISO R1860	Precision Reels for Magnetic Tape Used in Interchange Instrumentation Applications.		
ISO R1861	7-Track 8 rpmm (200 rpi) Magnetic Tape for Information Interchange.		
ISO R1862	9-Track 8 rpmm (200 rpi) Magnetic Tape for Infortion Interchange.	X3.14-1973 Recorded Magnetic Tape for Information Interchange (200 CPI, NRZI)	
ISO RS1863	9-Track 32 rpmm (800 rpi) Magnetic Tape for Information Interchange.	X3.22-1973 Recorded Magnetic Tape for Information Interchange (200 CPI, NRZI)	FIPS PUB 3-1 Recorded Magnetic Tape for Information Interchange (800 CPI, NRZI)
ISO R1864	Unrecorded Magnetic Tape for Information Interchange, 8 and 32 rpmm (200 and 800 rpi), NRZI, and 63 rpmm (1600 rpi), Phase-Encoded.	X3.40-1973 Unrecorded Magnetic Tape for Information Interchange (9-Track 200 and 800 CPI, NRZI, and 1600 CPI, P.E.)	
ISO 2690	Unrecorded Magnetic Tape for Instrument Applications-- Physical Properties and Test Methods.		
ISO 3788	9-Track, 64 rpmm (1600 rpi) Magnetic Tape for Information Interchange.	X3.39-1973 Recorded Magnetic Tape for Information Interchange (1600 CPI, P.E.) X3.54-1976 Recorded Magnetic Tape for Information Interchange (6250 CPI, GCR)	FIPS PUB 25 Recorded Magnetic Tape for Information Interchange (1600 CPI, Phase Encoded)
SRM 3200	is used internationally.	SRM 3200 is used nationally.	Standard Reference Material 3200, Secondary Standard Magnetic Tape (Computer Amplitude Reference) (Sold by NBS).

6. Competitive Standards:

7. Standardization Status: Summarized in 5 above.

8. Implementation Status: Widely implemented in computers.

9. Known Manufacturing Uses:

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723; Mr. Sidney B. Geller, NBS, (301) 921-3723

11. Probable Sources of Information: Mr. Robert M. Brown, Vice-Chairman of ANSI X3, CBEMA, (202) 466-2288.

12. Bibliography: See 5 above.

13. Comments: There are no ANSI or Federal standards for the 7-track tapes shown in 5 above.

1. Designation: MEDIA

2. Title: Paper tape (one inch, 8-track)

3. Maintenance Authority: NBS/ ANSI/ ISO

4. Scope: Paper stock, tape and hole dimensions, coding, reels, rolls

5. Relationship to Other Standards:

ISO Recommendation or Draft Recommendation	Related National Standard	Related Federal Standard (FIPS)
ISO 1113 Representation of 6 and 7-Bit Coded Character Sets on Punched Tape	X3.6-1965 Perforated Tape Code for Information Interchange	FIPS PUB 2 Perforated Tape Code for Information Interchange
ISO 1154 Dimensions for Punched Paper Tape for Data Interchange	X3.18-1967 One-Inch Perforated Paper Tape for Information Interchange	FIPS PUB 26 One-Inch Perforated Paper Tape for Information Interchange
	X3.19-1967 Eleven-Sixteenths Inch Perforated Paper Tape for Information Interchange	
ISO 1729 Properties of Unpunched Paper Tape	X3.29-1971 Specifications for Properties of Unpunched Oiled Paper Perforator Tape	
	X3.20-1967 Take-Up Reels for One-Inch Perforated Tape for Information Interchange	FIPS PUB 27 Take-Up Reels for One-Inch Perforated Tape for Information Interchange
ISO 2195 Data Interchange on Rolled Up Punched Paper Tape-- General Requirements		

6. Competitive Standards:

7. Standardization Status: Summarized in 5 above.

8. Implementation Status: Widely implemented in minicomputers, teletype machines, certain other computer terminals.

9. Known Manufacturing Uses: Widely used to drive numerically controlled machine tools.

10. Known Sources of Information: Mr. John L. Little, NBS, (301) 921-3723

11. Probable Sources of Information: Teletype Corporation

12. Bibliography: See 5 above.

13. Comments: Some tapes contain Mylar plastic to resist tearing. Dimensions of such tapes are the same as for paper tapes. Dye is used in some tapes to improve optical reading of the holes. The 11/16 inch width of ANSI X3.19-1967 tape accommodates only 5 tracks for communications in "Baudot" 5-bit code.

1. Designation: FIPS PUB 25/ANSI X3.39-1973

2. Title: Recorded Magnetic Tape for Information Interchange (1600 CPI, Phase Encoded)

3. Maintenance Authority: ANSI X3B1

4. Scope: Hardware Standard. This standard specifies the recorded characteristics of 9-track, one-half inch wide magnetic computer tape, including the data format for implementing the Federal Standard Code for Information Interchange (FIPS 1) on magnetic tape media.

5. Relationship to Other Standards: See FIPS PUB 12-2, pages 17-18 under "Media, Magnetic Tape" for the relationship to 16 other standards dealing with magnetic tape. Also ANSI BSR X3.54 (6250 CPI).

6. Competitive Standards: All 7-track magnetic tape codes. All magnetic tape codes in use prior to 1967. EBCDIC is widely recorded on 9-track magnetic tapes, and such tapes are similar except for the coding.

7. Standardization Status: Magnetic tape standards were first approved in 1967 and have been augmented and updated ever since.

8. Implementation Status: With ASCII coding, as specified in the ANSI magnetic tape standards, these standard tapes are not nearly as widely used as similar tapes with EBCDIC coding, because of the prevalence of IBM System 360 and 370 machines using EBCDIC tapes.

9. Known Manufacturing Uses: Wherever manufacturing uses magnetic tapes.

10. Known Sources of Information: Mr. Michael D. Hogan and Mr. John L. Little, NBS, (301) 921-3723.

11. Probable Sources of Information: IBM, Honeywell, UNIVAC, Burroughs.

12. Bibliography: FIPS PUB 25/ANSI X3.39-1973; FIPS PUB 12-2, pages 17-18.

13. Comments: There have never been any ANSI standards for 7-track magnetic tape. The 9-track standards are identical to IBM 360/370 tapes except that the coding in the standards is specified as ASCII instead of EBCDIC. The 9-track tape was the first 8-bit environment in which 7-bit ASCII was embedded. The technique is to make the high order bit a "zero" bit when the other 7 bits are ASCII bits. Parity is always odd.

Relationships in 9-Track Magnetic Tape

Track No., ANSI X3.22-1967 (FIPS 3)	4	7	6	5	3	9	1	8	2
Environment (8 information bits)	P	E_8	E_7	E_6	E_5	E_4	E_3	E_2	E_1
ASCII (FIPS 1) Bits (high to low)	P	Z	b_7	b_6	b_5	b_4	b_3	b_2	b_1
IBM EBCDIC Bit Numbers (high to low)	P	0	1	2	3	4	5	6	7
Binary Weight (unpacked)	P	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Binary Weight (packed)	P	2^3	2^2	2^1	2^0	2^3	2^2	2^1	2^0
Packed Numeric Digit Order	P		High				Low		

Note that the ASCII low-order bit is b_1, and the EBCDIC low-order bit is bit 7. The packed numeric formats are not standardized. A more complex 10-character, 90-bit Group Encoding Scheme is employed on 6250 bpi magnetic tape.

1. Designation: ANSI BSR X3.48

2. Title: Magnetic Tape Cassette for Information Interchange (Co-Panar, 3.81 mm (0.105 in), 32 bpmm (800 bpi), PE)

3. Maintenance Authority: ANSI X3B5

4. Scope: Hardware Standard. This standard specifies the physical, magnetic, and recorded characteristics for a 3.81 millimeter magnetic tape cassette in order to provide for data interchange between information processing systems at a recording density of 32 bits per millimeter using phase encoding techniques.

5. Relationship to Other Standards: ISO DIS 3407 (technical deviations, probably compatible); ECMA-34, 1973 (technical deviations, probably compatible)

6. Competitive Standards:

7. Standardization Status: Final ANSI approval is pending and publication date is estimated to be August 1976. Designation will be ANSI X3.48-1976.

8. Implementation Status: The ANSI compatible cassette has been implemented widely in communication terminals, POS terminals, intelligent terminals, and minicomputers. It is used in General Electric Terminet 300; Hazeltine 2000, 3000, 5000; Interdata 74; Memorex 1280; Olivetti-P602; Sycor-340E, 310; TI-700 Series; and UNIVAC Uniscope equipments.

9. Known Manufacturing Uses: The ANSI cassette is used as a data storage device in data processing systems designed for scientific, business, and industrial applications.

10. Known Sources of Information: Mr. Raymond C. Smith, 3M Company, Chairman of ANSI X3B5, (612) 733-6297; Mr. Michael D. Hogan, NBS, Member of ANSI X3B5, (301) 921-3723; Mr. William F. Hanrahan, CBEMA, Secretary of ANSI X3, (202) 466-2288.

11. Probable Sources of Information: ANSI X3B5 Membership List (available from CBEMA)

12. Bibliography: ANSI BSR X3.48

13. Comments: It is anticipated that ANSI X3.48-1976 will be available by August 1976. A FIPS PUB adopting the requirements of ANSI X3.48-1976 has been drafted for Federal and public review.

1. Designation: ANSI BSR X3.56

2. Title: Magnetic Tape Cartridge for Information Interchange, 4 Track, 0.250 inch (6.30 mm), 1600 bpi (63 bpmm), Phase Encoded.

3. Maintenance Authority: ANSI X3B5

4. Scope: Hardware Standard. This standard specifies the recorded characteristics for 0.250 inch magnetic tape cartridge in order to provide for data interchange between information processing systems at a recording density of 1600 bits per inch using phase encoding techniques.

5. Relationship to Other Standards: ANSI BSR X3.55 (base); ISO DIS 4057 (technical deviations, probably compatible); ECMA-46, 1976 (technical deviations, probably compatible)

6. Competitive Standards:

7. Standardization Status: Final ANSI approval is pending and publication date is estimated to be late 1976. Designation will be ANSI X3.56-1976.

8. Implementation Status: The ANSI compatible cartridge has been chosen for use in many minicomputers and communication terminals. It is used in the Three Phoenix TCT-300, the Kennedy Co. 4344/45/46 digital cartridge recorders, and the Mohawk Data Sciences 2021/2022 cartridge tape drive, for example.

9. Known Manufacturing Uses: The ANSI compatible cartridge is used as a data storage device in data processing systems designed for scientific, business, and industrial applications.

10. Known Sources of Information: Raymond C. Smith, 3M Company, Chairman of ANSI X3B5 (612) 733-6297; Michael D. Hogan, NBS, Member of ANSI X3B5, (301) 921-3723; Mr. William F. Hanrahan, CBEMA, Secretary of ANSI X3, (202) 466-2288.

11. Probable Sources of Information: ANSI X3B5 Membership List (available from CBEMA)

12. Bibliography: ANSI BSR X3.56; ANSI BSR X3.55

13. Comments: ANSI BSR X3.55, Unrecorded Magnetic Tape Cartridge for Information Interchange, 0.250 inch (6.30 mm), 1600 bpi (63 bpmm), Phase Encoded, contains the mechanical and magnetic requirements for the 0.250 inch magnetic tape cartridge. It supports ANSI BSR X3.56 and will be published concurrently.

APPENDIX A - STATEMENT OF WORK

"National Bureau of Standards Support in the Determination and Evaluation of U.S. Industry Standards Applicable to the Development of a Computer Aided Manufacturing Architecture"

1.0 OBJECTIVE

The objective of this effort is to provide the Air Force with definition and analysis of existing and potential standards which are necessary for the optimum development and implementation of integrated computer aided manufacturing. The Air Force Computer Aided Manufacturing Program team will develop a close working relationship with the National Bureau of Standards staff. This relationship will serve as a basis for continued co-involvement in the standards area throughout the Computer Aided Manufacturing Program. Results from this effort will provide the Air Force with a sound basis upon which to structure the development of individual computer based subsystems so that these systems not only work independently, but are able to perform as an integrated computer aided manufacturing system.

2.0 SCOPE

Five tasks are outlined to fulfill the objectives of this program.

2.1 Identify and provide current standards applicable to computer aided manufacturing.

2.2 Analyze current standards.

2.3 Assess actual usage of standards throughout industry.

2.4 Hypothesize optimal standards for integrated computer aided manufacturing.

2.5 Assess the relative roles of existing standards organizations and the Air Force in organizing for the development of optimal standards for computer aided manufacturing.

The existing expertise and experience level of the NBS with respect to standards, standards usage, standards conflicts and standards organizations will be called upon to perform the program tasks. Especially in task 2.3, but in general for all tasks, additional information information should be gained from outside government and industry sources to the extent required to increase the confidence level in task results. Hypotheses about future standards needs should also deal mostly with NBS experience and should be tested by outside sources only to the degree required to insure confidence in results.

The results of each task will be a report containing findings, conclusions and recommendations as appropriate according to Section 5. In addition, monthly written reports will be provided to AFML/LT identifying both progress and problems. A close verbal working relationship is also expected between NBS and the Air Force Computer Aided Manufacturing team during program execution.

3.0 BACKGROUND

Developments in the use of the computer as an aid to manufacturing have proceeded in a modular but disjointed fashion. Hardware and software systems have been designed and developed to solve the particular problem of the day and have been by and large limited in scope in order to most expediently aid the performance of a particular manufacturing function. The events subsequent to the development of N/C machine tools and the APT language are examples of this approach. Integration of these systems has been attempted in some cases, but only as an afterthought. This situation has resulted in the proliferation of disjointed computer software and hardware that has in many ways tended to actually magnify problems in manufacturing.

The long term adverse effect of continuing development in this way has been recognized both in the United States and in many foreign countries. Information compiled by the Comptroller General of the United States and others suggests that foreign nations have not only identified this problem, but have developed national programs aimed directly at providing strong impetus to increased productivity through the application of integrated Computer Aided Manufacturing systems.

The evidence, both abroad and in this country, advises that the economic and sociological benefits to be gained from this integration far exceed those benefits that have been accepted as being directly attributable to individual development efforts. This is particularly true in discrete parts-batch manufacturing based-industries because of such factors as the dual requirement to maintain both a flexible fabrication base and a highly efficient, controlled operation. These companies comprise a high percentage of U. S. industry, but their individual outputs are relatively small. The prime aerospace companies and their vast network of subcontractors fall into this group.

The Air Force recognized these facts and in 1973 produced a conceptual master plan (AFML-TR-74-104) which attempted to identify and group the major functions of manufacturing so that an organized approach at integration could evolve. The results of this contractual effort were briefed to American industry in June of 1974. At that time there appeared to be a general opinion in industry that an important new data base in support of integrated Computer Aided Manufacturing had been created, but there was little agreement in either the public or the private sector as to a subsequent course of action. Dialog in this vein continued between industry and DoD for the remainder of 1974. Subsequently, further study of Computer Aided Manufacturing was undertaken in 1975 by the Air Force in response to a memorandum by Deputy Secretary of Defense, W. P. Clements. This study focused on the state of the production art in aerospace and related industries. Its primary objective was identification of cost saving opportunities in the production of defense materiel through the application of computers and elements of computer technology. Among the conclusions was that subsystem integration provides the key to ultimate benefit realization in this area.

NASA, through the IPAD Program, is attempting to accomplish the same objective in the design area, but through the use of a single, dedicated hardware/software computer system. Other organizations such as the Aerospace Industry Association, Computer Aided Manufacturing-International Incorporated, the Society of Manufacturing Engineers, the National Science Foundation (RANN Program) and possibly others have also attempted to evolve programs which consider not only advances in individual areas of manufacturing, but also the relationship of some of these areas.

All of these programs recognize the need for an organized plan for integration of subsystems in order to insure such factors as portability of software and adequacy of communications. However, to date, although it is clear that a key to affordable integration is through the use of various types of standards, no effort has been made to specifically identify and characterize actual requirements which would enable integration of Computer Aided Manufacturing subsystems.

In addition, while good work continues to be accomplished in areas related to Computer Aided Manufacturing by various standard groups, such as ANSI, ISO, EIA, NCS, SME, IEEE, CAM-I and some computer system manufacturers, no work has been done to identify potential conflicts or to establish a master plan for standards development.

The Air Force has proposed a major new initiative in the development of Computer Aided Manufacturing. This is a long term program which includes development of individual subsystems within the general areas displayed in Section 5, Attachment 2. The long term goal of this program is totally integratable Computer Aided Manufacturing.

In its ultimate, this would allow manufacturing activities to be performed in a manner which today is only barely within the ability to comprehend -- both managerially and technically. For example, two illustrative, conceptual goals could be:

(1) The ability for a part designer to not only optimally design a part, but at the same time to subject this part to a performance evaluation and to plan for the most economical fabrication of the part within the constraints of schedule and availability of raw materials. Further, it could be envisioned that the fabrication test may be performed immediately and the part production may be automatically introduced into the overall manufacturing plan.

(2) A manufacturing capability where all information is available in standard data formats "on time" via computer display and where the chief executive's staff could be able to perform "what if" simulation ranging from global risk analysis to plant layout.

The first five years of this program have been outlined in some detail. Included are projects that are both quite specific within the functional areas of manufacturing and projects solely designed to effect the interface of code within subsystems and communications between subsystems. Some of these projects will advance the state of the art in discrete areas such as sheet metal part fabrication and assembly. These projects are required both for the long-range goal and in order to demonstrate short-term payoff. But, even in short-term projects, the overriding goal is integration. This can only be accomplished through the use of interface standards, acceptable validation procedures and techniques and other such concepts.

It is definitely not the intent of the Air Force to legislate in these areas, neither does it seem feasible that all standards related problems can be solved with today's technology. This is also a most dynamic environment and the probability and possibility of change must be allowed in order to accommodate unforeseen technical advances and to not stiffle individual initiatives. Nevertheless, it is believed that both problems and requirements must be at least recognized in the early stages of the Computer Aided Manufacturing Program. Where existing standards will aid integration, they should be utilized. Where standards do not exist, they should be developed by the appropriate agency and then adopted within the Air Force Program. Where conflicts arise, they should be identified and a plan for their resolution outlined.

4.0 TASKS/TECHNICAL REQUIREMENTS

4.1 The first task involves the accumulation and grouping of current standards which may be relevant to the use of computers in all aspects of manufacturing.

4.1.1 For the first report NBS shall obtain and provide the Air Force with copies of all current standards which may be relevant to the use of computers in all aspects of manufacturing. As a minimum, such areas as CAD/CAM interfaces, hardware interfaces, software interfaces and procedures, communications codes and protocols, test validation concepts, security issues, Federal Information Processing Standards (FIPS), et al, which apply to manufacturing shall be addressed.

4.1.2 NBS shall organize these published standards into logical groupings for easy reference by the reader. A graphical matrix display of these groupings shall be prepared as part of the Task Report.

4.1.3 NBS shall develop a bibliography of all standards obtained for Task 4.1.1 and include this in the first report as identified in Section 5, Attachment 1.

4.2 NBS shall analyze standards obtained in Task 4.1.1 in order to determine the merit of various standards or groups of standards for use in integrated Computer Aided Manufacturing.

4.2.1 NBS shall analyze each standard or groups of standards (as appropriate) obtained in Task 4.1.1 for the standards merit in terms of relevance for use in integrated Computer Aided Manufacturing. NBS shall identify existing and potential conflicts between standards or groups of standards considering such factors as: fitness for use in particular application approaches and stability in light of advancing manufacturing and computer technology.

4.2.2 NBS shall modify the matrix of Task 4.1.2 in order to clearly display the results of Task 4.2.1 in graphical form.

4.2.3 Task 2.0 report of NBS shall annotate the bibliography obtained in Task 4.1.3 and shall include this bibliography and the matrix of Task 4.2.2 as part of the Task Report identified in Section 5, Attachment 1.

4.3 Standards actually in use today within manufacturing industries should be identified. This shall include both those standards displayed in Task 4.1 and any additional private standards which may be of significance to the CAM Program. Included shall be both computer based standards now in use as well as those standards likely to be affected or which must be considered in the application of computers to aid a particular manufacturing function.

4.3.1 Standards actually in use should be identified including communications codes, protocols and line disciplines. Of particular interest are defacto communications and security standards which may be evolving as a result of advanced network research and recent announcements by IBM, Digital Equipment Corporation, and Control Data Corporation.

4.3.2 NBS shall develop a report at the completion of this task summarizing the actual usage of standards as identified in Section 5, Attachment 1.

4.4 Upon completion of the analysis of current standards in Task 4.2 and the assessment of the usage of standards in Task 4.3, hypotheses from NBS about the most appropriate ("best") standards for specific applications are required.

4.4.1 NBS shall integrate their experience and expertise with their findings from Task 4.2 and Task 4.3 and for each standard or set of standards identified in Task 4.1.2, NBS shall recommend the "best" to be used in integrated Computer Aided Manufacturing.

4.4.2 If new standards are suggested, status and timing for development of these new standards should be outlined and suggested mechanisms for development of these needed new standards explained.

4.4.3 If existing standards require modification, then status and timing for changes shall be outlined and mechanisms explained as per Task 4.4.2.

4.4.4 When existing standards meet the "best" requirements, these should be noted.

4.4.5 NBS shall clearly display the results of Task 4.4 through modification of the matrix of Task 4.2.

4.4.6 All requirements of Task 4.4 shall be included in the fourth Task Report as identified in Section 5, Attachment 1.

4.5 A review of existing standards orgdnizations shall be performed in order to identify issues such as potential conflicts, duplication of effort and procedural approaches which should be addressed in conducting the integrated Computer Aided Manufacturing Program.

4.5.1 The present and planned activity of existing standards organizations such as ANSI, ISO, IEC, EIA, NCS, SME, CAM-I shall be assessed to identify potential conflicts and duplication of effort which should be addressed by the Air Force in resolving issues or filling in holes displayed in the matrix of 4.4.5.

4.5.2 These organizations shall be reviewed to identify the most effective individual structures and procedural approaches which are utilized.

4.5.3 An approach including funding, time phasing and required working relationships shall be outlined to result in the most effective standards and their related practices and procedures for integrated CAM.

4.5.4 All requirements of Task 4.5 shall be included in the fifth task report identified in Section 5 of Attachment 1.

5.0 DELIVERABLE REPORTS

5.1 This Statement of Work contains five (5) tasks to be performed by NBS. Each task has a report as its end product. These reports are deliverable items due thirty (30) days after completion of each task.

5.2 The task schedules, program reviews and report delivery dates are identified by the contract Milestone Chart (Attachment No. 1 of this Section).

6.0 SPECIAL CONSIDERATIONS

6.1 All AFML funded travel by NBS personnel necessary for this program shall be subject to AFML Project Manager approval.

APPENDIX B - CAM SYSTEMS ARCHITECTURE

To identify where standards are needed in a large system, and particularly to identify where the major system interfaces are located, one must have a concept of the overall system structure or architecture.

Since the Air Force will develop the detailed ICAM architecture after this study is complete, existing system concepts and architectures will be examined to identify the common elements to guide the further presentation and discussion of relevant standards.

In discussing the architecture of CAM it is soon apparent that there is no widely accepted definition or overall concept. CAM can be defined as the application of computers in the manufacturing process. This definition is very global and does not clearly define the boundaries of CAM and does not identify concepts which are not CAM.

Several examples are useful. The early use of computers in manufacturing industry fell into three main categories: business applications such as payroll and accounting programs, scientific and engineering support programs, and APT. In fact, it has been estimated that fully 30 percent of the use of IBM 709 series machines in the aerospace industry in the first half of the 1960's was committed to APT runs. Subsequent uses included inventory control, customer order servicing, scheduling and control, and computer aided design. It should be noted that, with the exception of interactive CAD systems, most CAM programs have been batch type programs, even when available on time sharing systems to smaller companies.

Two typical examples of CAM systems are the Rock Island Arsenal Pilot Automated Shop Loading and Control System (PASLACS) specifications (Figure 1), which cover several commonly available systems and the Norwegian AUTOKON 71 programs for ship building shown in Figure 2. Even if an online data base is kept of all files, these types of systems are basically a set of programs that run in batch mode.

The more recent development of large Data Base Management Systems, Management Information Systems, communication systems, and networks, including many computers and real time operating systems, has led to concepts of CAM systems in which there is a real time interaction with the system. IBM's COPICS, Figure 3, is an excellent example, as is the CAM system in use at McDonnell Douglas, Figure 4. The Caterpillar system, Figure 5, shows the use of such concepts outside of the aerospace industry.

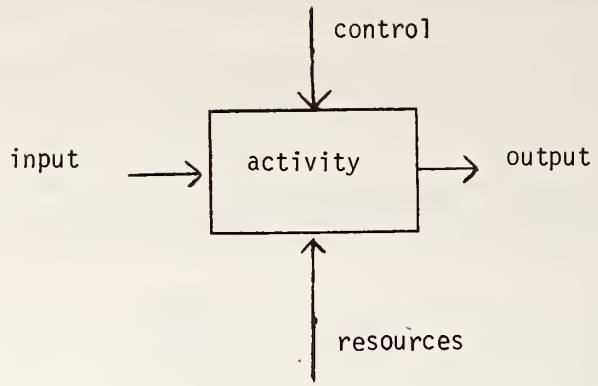
Future systems concepts emphasize distributed processing and data structures and "smart" data base concepts. A "smart" data base is one which can answer questions about information that is implicitly, as well as explicity, in the data through the use of modelling or simulation programs. The ICAM schematic, Figure 6, illustrates this idea.

The development of concepts of integrated manufacturing systems, in which applications programs are all interfaced to a central operating system, data base management system and management information system structure which can operate in a real time multi-user manner, possibly with several computers coupled in a hierarchy or other network, has led to a proliferation of concepts. There are significant commonalities that are often obscured by different formats of presentation and by the use of conventions that mix physical activities with information processing activities.

In fact, there are three different architectures that are simultaneoulsy present in a CAM system and which must all be considered together. These are:

1. THE ARCHITECTURE OF THE MANUFACTURING SYSTEM

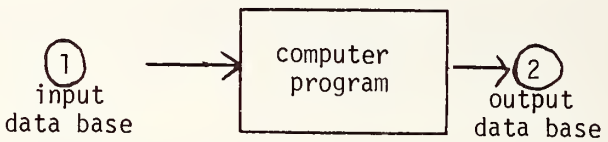
   This is the structure of the operation of the manufacturing process itself, including both physical activities and the management of those activities. Figure 7 shows the CAM-I Advanced Technical Planning Committee concept of a manufacturing system, using the cell model convention:



Note that in a cell model the input and output may be either physical material or information, depending on the activity.

2. THE ARCHITECTURE OF THE CAM SYSTEM

This is a set of computer programs that process information. The input and output are always data which, of course, may have a physical analog in material or operations in the manufacturing process. Figure 8 shows an architecture of a CAM system developed by the CAM-I Standards Committee, working from the CAM-I Long Range Plan, and using a modified "nodes and paths" convention:



This convention was chosen to highlight the data bases which are the interfaces in a CAM system.

3. THE ARCHITECTURE OF THE COMPUTER SYSTEM WHICH RUNS THE CAM PROGRAMS

Figure 9 shows a schematic of part of a computer system with at least two computers networked together. This schematic was developed to show the main elements of a computer system that must be considered in evaluating standards relevant to CAM.

Note that if only one computer is involved and the Input/Output channels are dropped, the parts of the schematic could be "wrapped around" the data base to obtain the form of Figure 6. Figure 9 is thus a valid representation of the ICAM concept, with the communications subsystems explicitly identified.

It is this architecture of a computer system, Figure 9, that will receive the greatest attention in this study, since it is here that systems standards must be set to assure software transportability and computability.

Figure 1

U.S. Army Rock Island Arsenal Pilot Automated
Shop Loading & Control System (PASLACS)

The Rock Island Arsenal PASLACS specifications show the function of early CAM systems for scheduling and control, many of which are still in active use.

Each of the functions shown is typically a batch program with data input prepared for each specific run.

This schematic is particularly useful in showing the feedback required in a scheduling and control system for batch manufacturing.

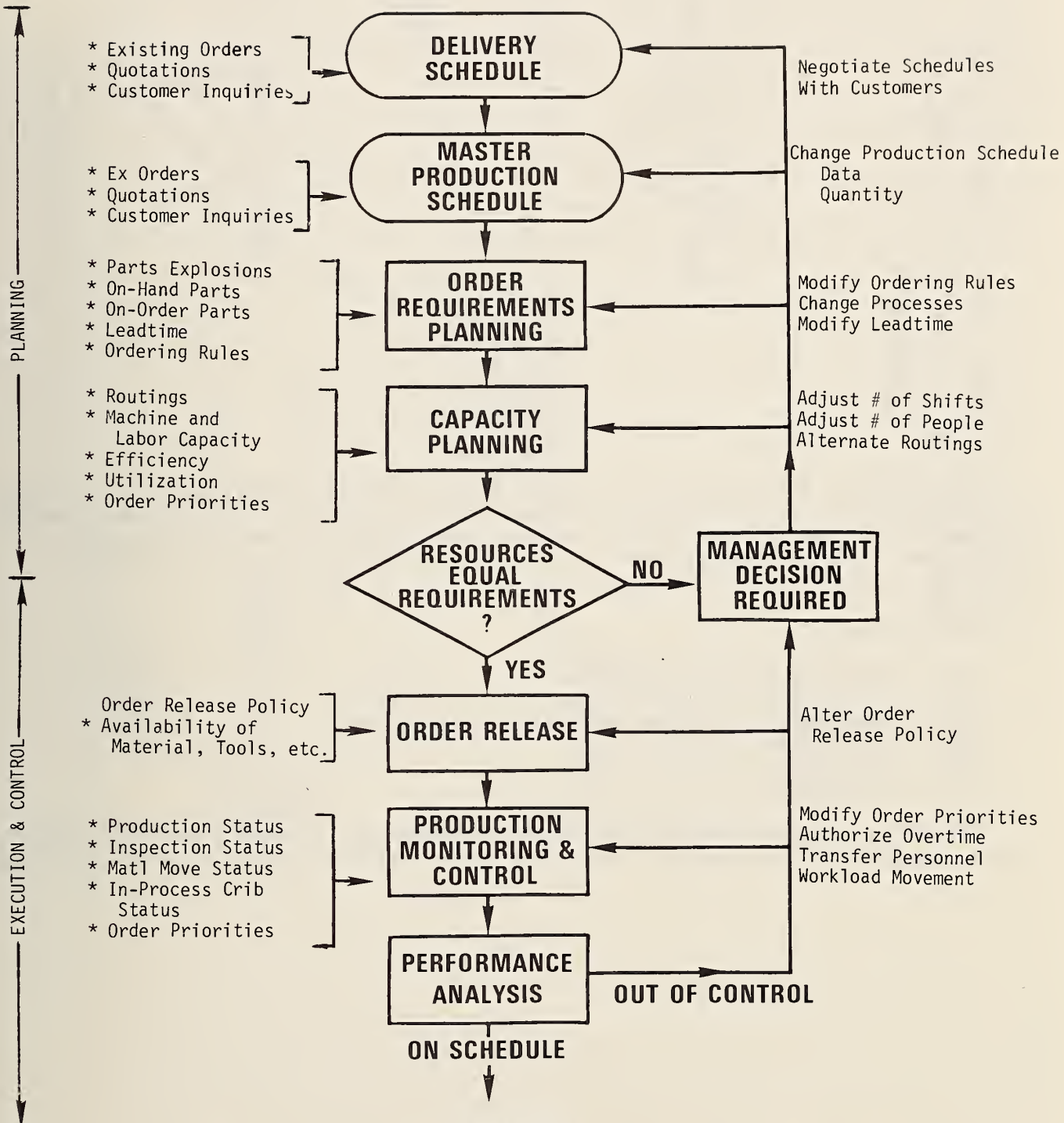


FIGURE 1
U.S. ARMY ROCK ISLAND ARSENAL PILOT AUTOMATED SHOP LOADING & CONTROL SYSTEM (PASLACS)

Figure 2

AUTOKON 71

The AUTOKON 71 system is a set of batch computer programs for ship design and fabrication linked directly or indirectly to a central data base manager. The system was developed in Norway and purchased by the Maritime Administration (MARAD) of the U.S. Department of Commerce for use by U.S. shipbuilders. The Illinois Institute of Technology Research Institute maintains this software under contract to MARAD.

The programs are:

FAIR, DRAW, TRABO: Fairing programs
ALKON: NC flame cutter part programming
NEST, PRODA: Assist in developing flame cutting programs
LANSKI: Longitudinal curves
SHELL, TEMPLATE: Hull plate programs
DUP: File management utility.

AUTOKON is a computer aided design and NC part programming system. PASLACS, Figure 1, covers only scheduling and control. Both concepts are part of computer aided manufacturing.

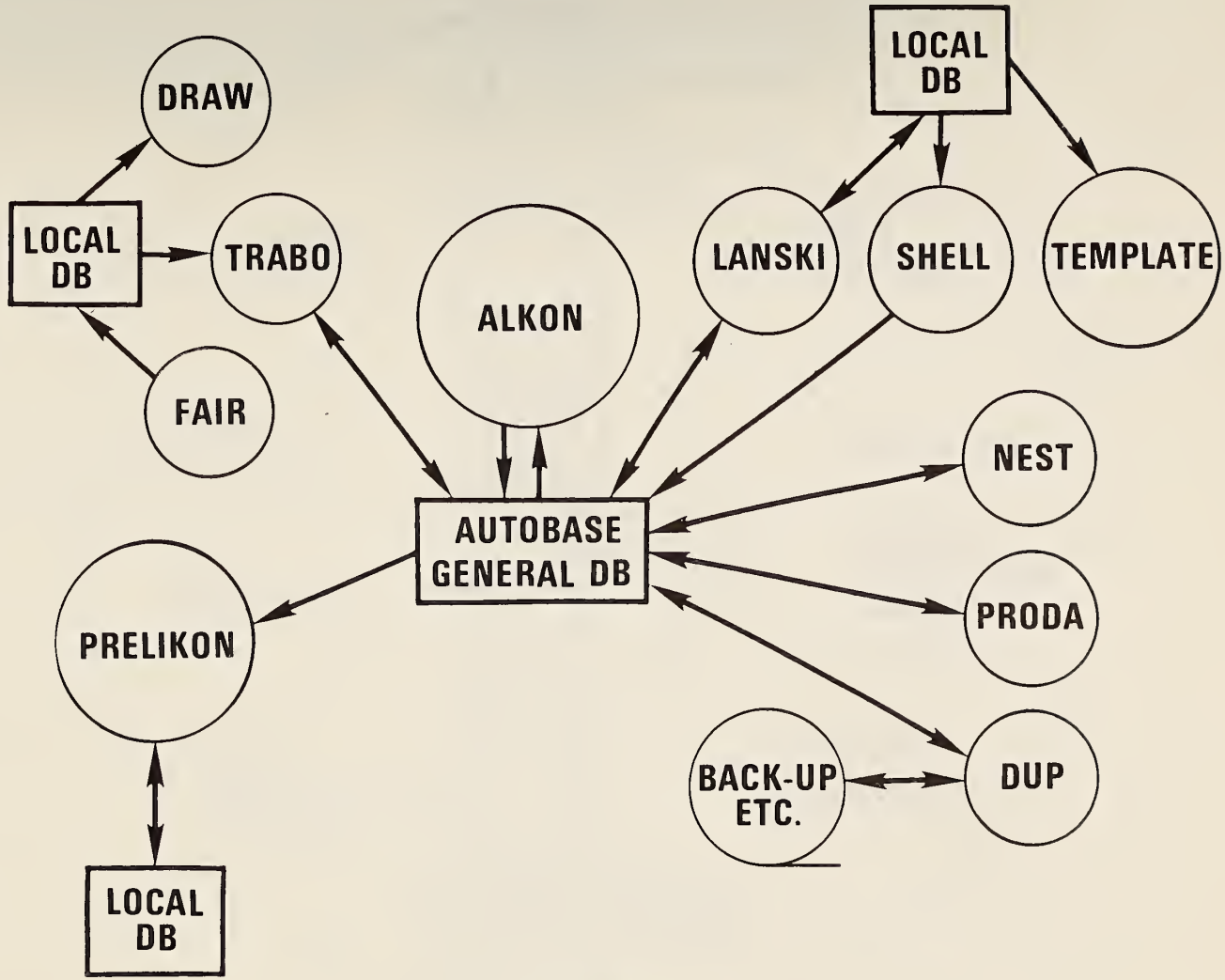


FIGURE 2
NORWEGIAN AUTOKON 71 SYSTEM
FOR SHIP DESIGN AND FABRICATION

Figure 3

COPICS Concept

The IBM Communications Oriented Production Information & Control System (COPICS) concept emphasizes the idea of CAM systems created around a central data base with a data base management system (DBMS). COPICS is a conceptual design study not a specific product. The system is conceived of being implemented on one computer or several linked computers, with possibly hundreds of terminals accessing the system throughout a company on a real time interactive basis. The system thus depends on modern multi-user, real time operating systems with DBMS capability.

FIGURE 3
IBM COPICS (COMMUNICATIONS ORIENTED PRODUCTION INFORMATION & CONTROL SYSTEM) CONCEPT

Figure 4

McDonnell Douglas CAM Concept

All of the production and service departments shown on the opposite page can (or will in the future) access the data and programs of the MCAIR CAM system which runs on the computers of McDonnell Douglas Automation Company. This system is essentially an implementation of the COPICS concept of Figure 3.

The diagram shows the extent of applications coverage of a major state of the art CAM system.

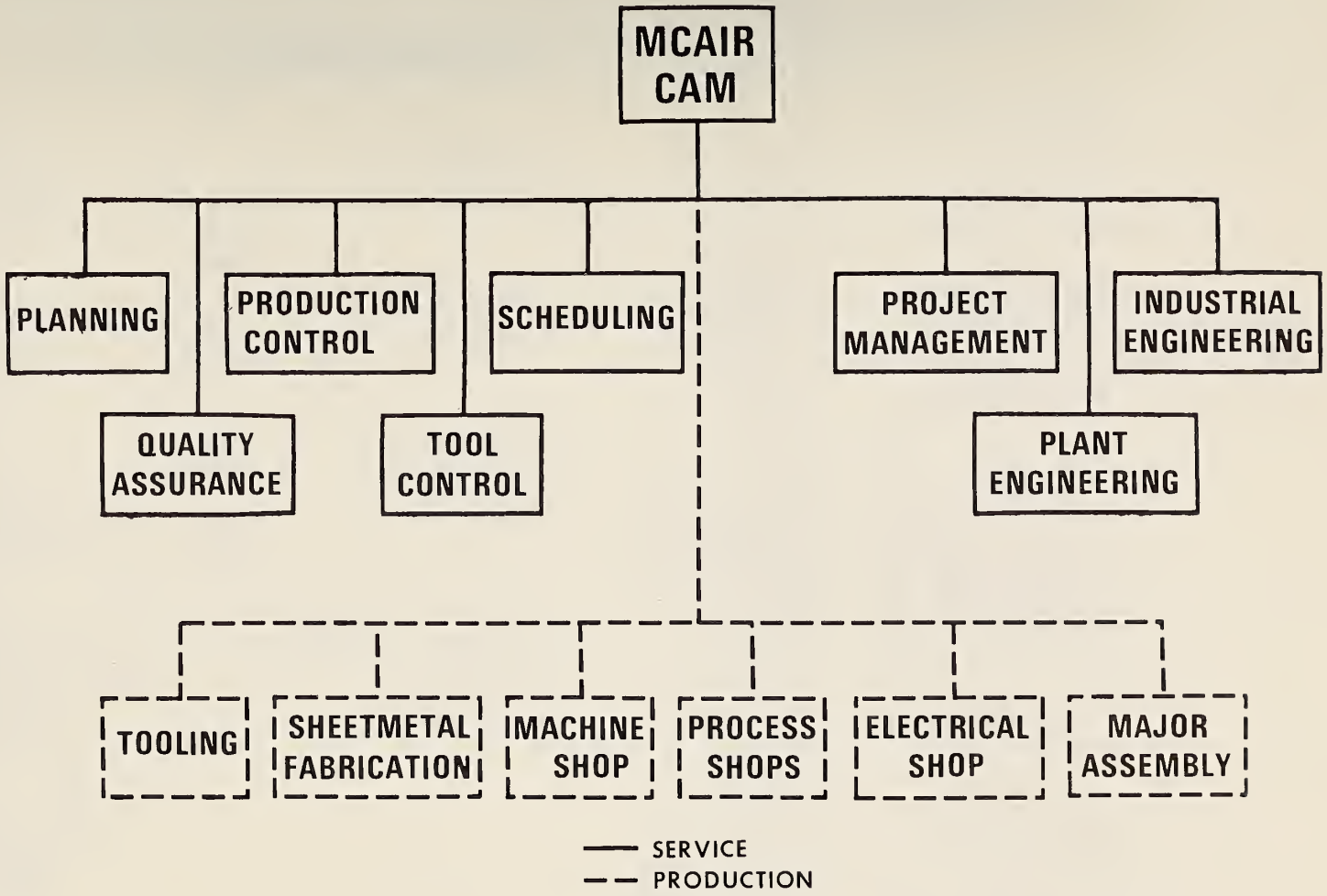


FIGURE 4
McDONNELL AIRCRAFT CAM DEPARTMENT

Figure 5

Caterpillar CAM System

This diagram shows the use of integrated CAM concepts outside the aerospace industry. The Caterpillar system, which could be applied to any large batch manufacturing operation, shows the integration of design, process planning, and manufacturing operations in a single system with a central data base structure.

The basic information flow in this integrated system, from left to right with feedback loops, is clearly shown.

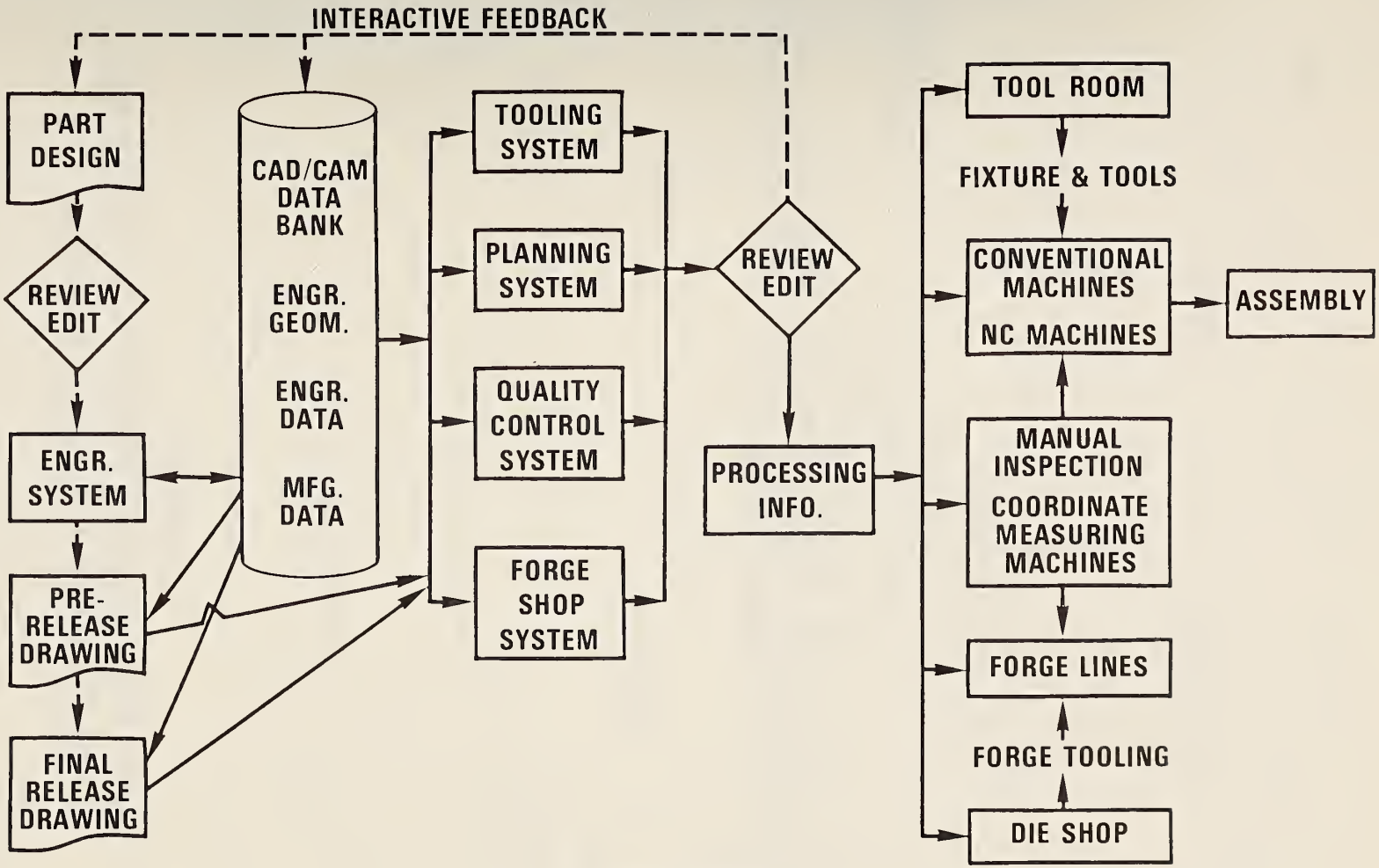


FIGURE 5
CATERPILLAR TRACTOR COMPANY CAD/CAM CONCEPT DIAGRAM

Figure 6

Air Force ICAM Architecture

The Air Force ICAM (Integrated Computer Aided Manufacturing) concept is similar to the COPICS concept, Figure 3, but adds a third layer of software: general purpose utility programs, including simulation capability to create a "smart" data base system.

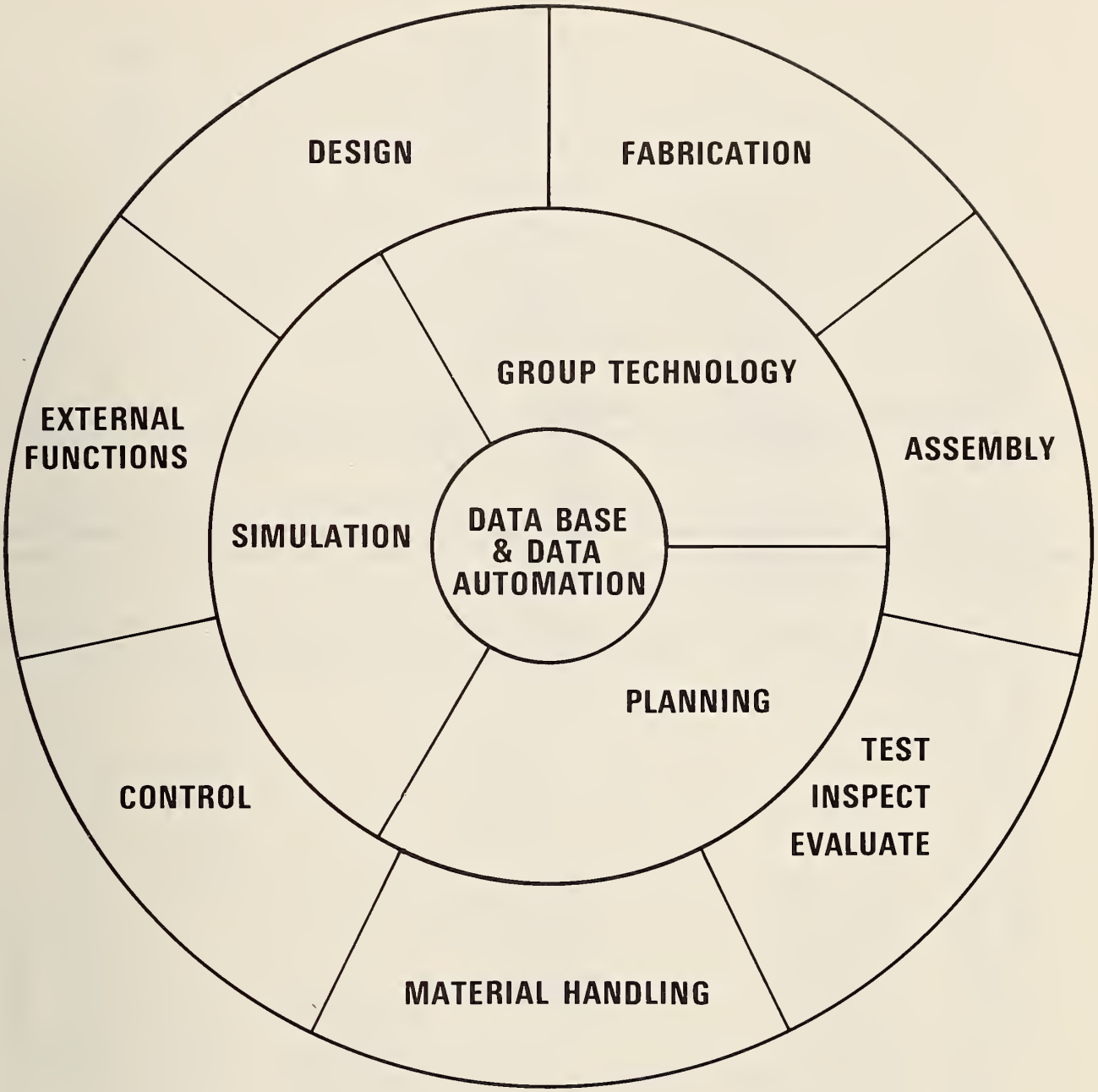


FIGURE 6
AIR FORCE ICAM (INTEGRATED COMPUTER AIDED MANUFACTURING) ARCHITECTURE

Figure 7

CAM-I Cell Model

Computer Aided Manufacturing-International, Inc. (CAM-I) is a not-for-profit organization of industry, Government, and universities dedicated to advancing the use of computers in manufacturing.

The Advanced Technical Planning Committee of CAM-I has created a cell model diagram of manufacturing. This diagram shows 6 basic functions of manufacturing. The nomenclature used in earlier figures would be: design, planning and scheduling, process planning, inventory control, manufacturing control, and shipping.

This diagram is one structuring of the functions needed in an integrated CAM system and should be compared with Figure 5, which has a comparable but different partitioning.

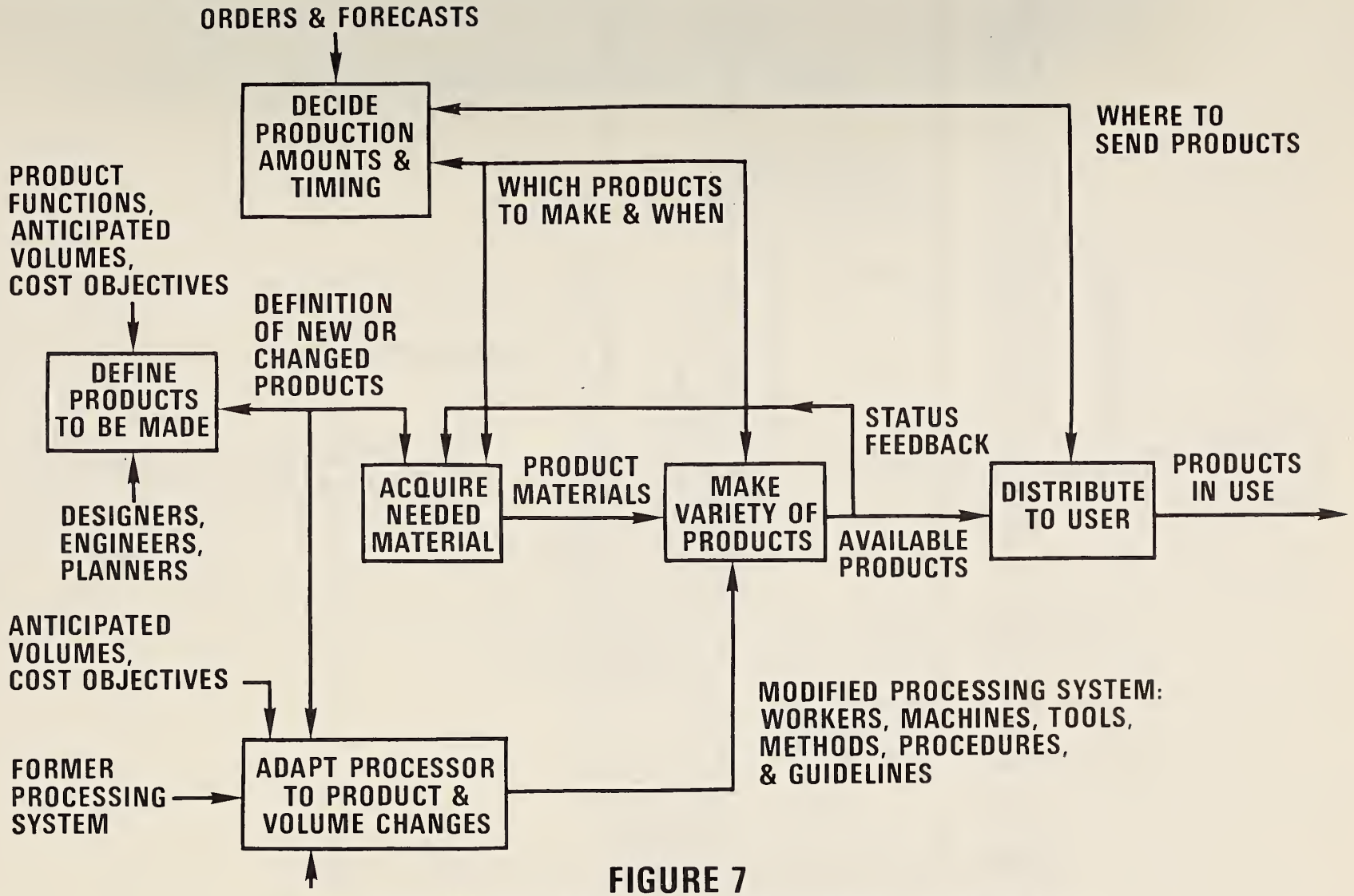


FIGURE 7

CAM-I ADVANCED TECHNICAL PLANNING COMMITTEE CELL MODEL DIAGRAM OF MANUFACTURING

Figure 8

CAM-I Standards Committee CAM Architecture

A CAM system may be strictly defined as a set of computer programs which process data. Working from the cell model diagram in Figure 7, the CAM-I Standards Committee created this diagram of a CAM system.

The functions and data flows of this diagram should be compared with Figures 1 and 5.

The data bases, shown as circles in the diagram, are:

1. Production Schedule
2. Product Design
3. Raw Material Inventory
4. Work-In-Process, Finished Goods Inventory, Machine Tool Utilization
5. Shipments
6. Overall Manufacturing Plan (Routing Sheets, Tooling, Part Programs, QC Plans)
7. Production Order Release, Production Plans and Schedule
8. Schedule, Process or Product Revision Requirements
9. Group Technology Data Base (Parts Data, Standard Plans)
10. Management Data: Output Node
11. External Information: Input Node (Marketing Information, Customer Orders, Product Functions, Cost Objectives, Anticipated Production Volumes)

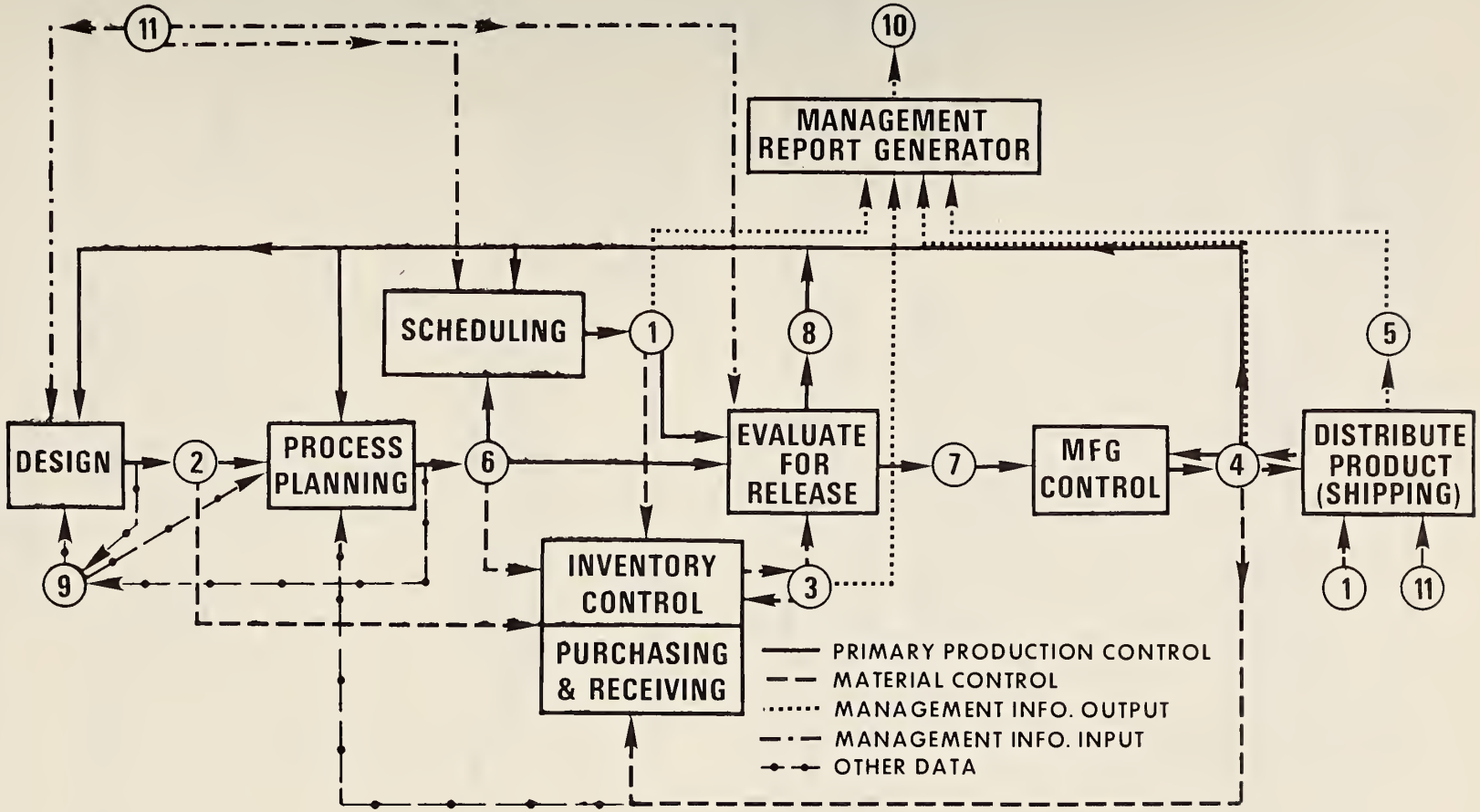


FIGURE 8
CAM-I STANDARDS COMMITTEE CAM ARCHITECTURE

Figure 9

Architecture of a Computer System

Two computers out of a distributed network are shown in this figure. This diagram shows the interfaces between the application programs and the host system and between various computers and provides a visual framework for discussing standards important to software integration and portability in a distributed system.

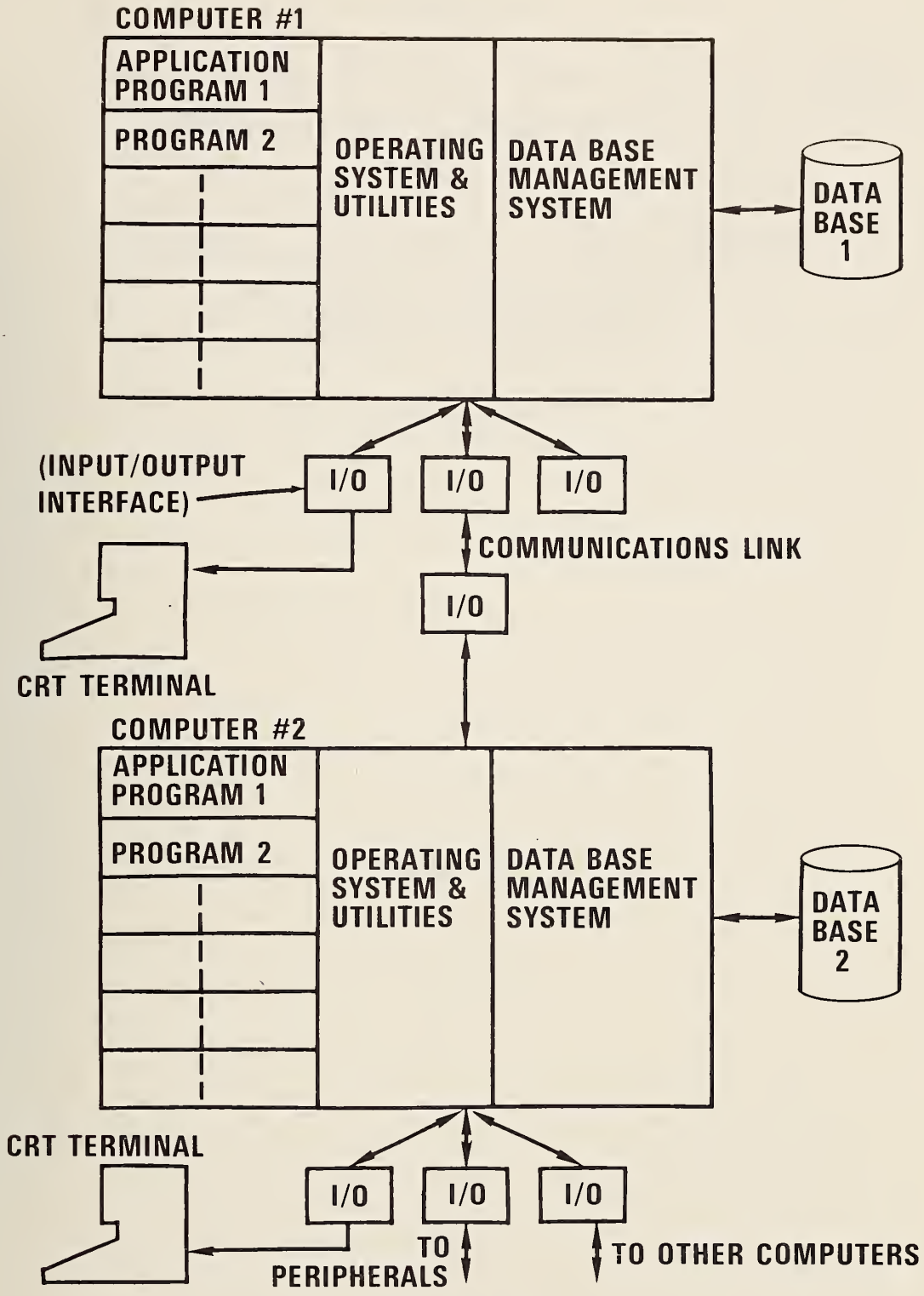


FIGURE 9
ARCHITECTURE OF A COMPUTER SYSTEM USED IN A CAM SYSTEM

APPENDIX C

A SURVEY OF ARTIFICIAL INTELLIGENCE PROGRAMMING LANGUAGES

Prepared for National Bureau of Standards

by

Chuck Rieger Hanan Samet Jonathan Rosenberg

Department of Computer Science
University of Maryland
College Park, Maryland 20742

November 1976

CONTENTS

1. Introduction

This paper describes some of the recently developed Artificial Intelligence programming languages: SAIL, LISP, MICROPLANNER, CONNIVER, MLISP, POP-2 and AL. These programming languages are distinct from languages previously used in computer-aided manufacturing environments [Leslie72] in that they provide capabilities for the development of high-level symbolic planning and supervisory control in addition to the simple numerical control of machine tools. The paper includes (1) surveys and comparisons of the distinctive features of these languages as they might be used in a computer-automated manufacturing environment, (2) a sample automated manufacturing task, and how it might be expressed as a program in each language, (3) discussions of the standardization status of each language, and (4) conclusions and recommendations to NBS, with emphasis on the types of features which are most desirable and applicable to the automated-shop environment.

2. SAIL

2.1. Introduction

SAIL has its origins in a merger of LEAP [Feldman69], an associative language, and a version of ALGOL 60 [Naur60]. Therefore, unlike most of the other artificial intelligence languages, it is not LISP-based. It is a general purpose compiled language with an extensive run-time library of functions. As befits its ALGOL nature, SAIL has block structure and explicitly typed, statically scoped variables. The data types available include INTEGER, REAL, STRINGs of arbitrary length, structure, pointer, LIST, SET, and aggregates of the previous (i.e. ARRAYs).

Some of the more important features of SAIL are discussed separately below. These include the associative data base facility, the capability for usage of SAIL as a host language in a CODASYL [CODASYL71] data base management system, the control structures, and the system building facilities. Finally, a summary is presented of current standardization efforts.

2.2. Associative Data Base

SAIL contains an associative data base facility known as LEAP which is used for symbolic computations. This enables the storage and retrieval of information based on the partial specification of the data. Associative data is stored in the form of associations which are ordered three-tuples of ITEMs. Associations are often known as TRIPLES. For example:

```
FASTEN XOR NAIL EQV HAMMER
FASTEN XOR SCREW EQV SCREWDRIVER
FASTEN XOR BOLT EQV PLIER
```

Associations may be conceptualized as representing a relation of the form

```
        Attribute XOR Object EQV Value
or      Attribute (Object) = Value
```

Most programming languages (e.g. LISP) provide the following associative-like mechanism:

```
Given:  Attribute,Object
Find:   Value
```

However, SAIL enables the programmer to specify any of the components of the association and then have the LEAP interpreter search the associative store for all triples which have the same items in the specified positions. For example, the following may be used to retrieve all items that can fasten a nail:

```
FASTEN XOR NAIL
```

An ITEM is a constant and is similar to a LISP atom. Items have names and may also be typed so that data can be associated with them. An item may be declared, or created during execution from a storage pool of items by use of the function NEW. For example:

```
REAL ITEM VISE;
```

declares VISE to be an item having data of type real associated with it. VISE

could be used in an association of the form:

```
TOP XOR WORKBENCH EQV VISE
```

The data associated with an item is obtained by use of the construct DATUM. Thus in the above example, DATUM(VISE) denotes the capacity of the vise.

In order to deal with items, the user has the capability of storing them in variables (ITEMVARs), SETs, LISTs, and associations. The distinction between SETs and LISTs is that the latter have an explicit order while the former do not. In addition, an item may have an arbitrary number of occurrences in a LIST.

Associations are ordered three-tuples of items and may themselves be considered as items and therefore participate in other associations. Triples are added to the associative store by use of a MAKE statement and erased from the associative store by use of an ERASE statement. For example:

```
ERASE ATTACHED XOR ASSEMBLY1 EQV ASSEMBLY2
MAKE  ATTACHED XOR ASSEMBLY1 EQV ASSEMBLY3
```

The motivation for using an associative store is a flexible search and retrieval mechanism. Binding Booleans and Foreach statements are two methods of accomplishing these goals.

The Binding Boolean expression searches the associative store for a specified triple and returns TRUE if the triple is found and FALSE otherwise. The aim of the search is to find an association which meets the constraints imposed by the specified triple. If some of the components of the triple are unknown (such components are preceded by the special item BIND), then a successful search will result in the binding of the designated component. For example:

```
IF FASTEN XOR BIND OBJECT EQV PLIER THEN PUT OBJECT IN PLIER!SET;
```

In this case the store is searched for an object that can be fastened by a PLIER and if such an object is found, then it is placed in the set PLIER!SET. Note the use of the item variable OBJECT in the association. A successful search will result in this variable being bound and the corresponding item being placed in the set.

The FOREACH statement is the heart of LEAP. It is similar to the FOR statement of ALGOL in that the body of the statement is executed once for each binding of the variable. For example:

```
FOREACH X | PART XOR B747 EQV X AND DATUM(X) < 3
    DO PUT X IN B747!ORDER!SET;
```

In this case, assuming that the data associated with each part denotes quantity at hand, the associative store is searched for all parts of a B747 of which there are less than three pieces. These parts are placed in the set B747!ORDER!SET.

2.3. Data Management Facility

Unlike other artificial intelligence languages, SAIL has a capability of being used with an existing data base management system (DBMS-10 [DEC]) to handle large data bases stored on external storage. An interface exists [Samet76] which allows SAIL to be used as the data manipulation language in a CODASYL based data base management system. SAIL is relatively unique in this respect in that COBOL [COBOL74] has almost been exclusively used as the data manipulation language (DML) of such systems. This situation is not surprising since examination of the data description facility of the CODASYL report

reveals a very strong similarity to the data division of COBOL. Nevertheless, there have been some attempts to use FORTRAN ([Stacey74], [RAPIDATA]).

Ideally, a data manipulation language should include the following features. First, a full procedure capability which allows parameter passing, dynamic storage allocation, and recursion. Second, processing of Boolean requests should not be difficult. In a COBOL-based system this task is rather cumbersome as pointed out by [Parsons74]. In order to avoid currency problems raised by partial satisfaction of Boolean requests (the backtracking problem [Taylor76]), the user must build collections of pointers to related records. Third, there should be a capability for building an in-core data base so that operations such as set UNION and set INTERSECTION can be performed without the overhead of accessing extended storage more than once for any record.

SAIL has a mechanism, LEAP, for building associative data bases. Currently, this only works for internal memory due to implementation decisions. SAIL also has a record structure capability which enables the user to build an in-core data base. In a COBOL-based data base management system, whenever the user obtains an instance of a record type from the data base (i.e. he locates it via a FIND and fetches it via a GET), he has no convenient way of keeping it in temporary memory while obtaining another instance of this record type. Of course, he can allocate temporary storage for the various fields; however, this becomes rather unwieldy especially when he wishes to keep track of more than two instances of a record type. Alternatively, instances of certain record types can be refetched from the data base. In fact, this is the strategy that is generally followed. However, the cost is prohibitive.

Briefly, the SAIL interface provides a SAIL record structure declaration for each record type that has been defined in the data base management system. Primitives exist for the creation and modification of such records. The dynamic storage allocation capability of SAIL enables the creation of several instances of each record type each of which is identified by an entity known as a record pointer.

As an example of the use of SAIL as a host language in a data base management system, consider the following program fragment. The task is to traverse a set named SUPPLIER owned by a WAREHOUSE record and extract an integer data item known as PARTNUM from each PART record which is a member of the set. The exact instance of the set occurrence is identified by the owner record, WAREHOUSE, having the value ELECTRICAL for the data item INDUSTRY. Since SAIL has a data structuring facility (known as a RECORD!CLASS and similar to a PL/1 [Beech70] structure) we define a data structure known as LISTX and a function to add items to the front of a list. The data structure LISTX has two fields - ELEMENT which is of type INTEGER and NEXT which is of type RECORD!POINTER (and points to another instance of the LISTX data structure). The function ADDTOLIST has two arguments - a pointer to the head of an instance of LISTX and the integer to be added to this instance.

```
RECORD!CLASS LISTX(INTEGER ELEMENT;
                   RECORD!POINTER (LISTX) NEXT);

PROCEDURE ADDTOLIST(REFERENCE RECORD!POINTER(LISTX) HEAD;
                    INTEGER VAL);
BEGIN
    RECORD!POINTER (LISTX) TEMP;
    TEMP_NEW!ELEMENT(LISTX);
    LISTX:ELEMENT[TEMP]_VAL;
    LISTX:NEXT[TEMP]_HEAD;
    HEAD_TEMP;
    END;
```

The COBOL/DML and SAIL encodings are given below. The critical difference is the step "Add PARTNUM in PART to result list." It is not immediately obvious how the concept of a list would be implemented in COBOL.

```
COBOL Program:
            MOVE 'ELECTRICAL' TO INDUSTRY IN WAREHOUSE.
            FIND WAREHOUSE RECORD.
            IF SUPPLIER SET EMPTY GO TO NONE!SUPPLIED.
NEXT:       FIND NEXT PART RECORD OF SUPPLIER SET.
            IF ERROR-STATUS = 0307 GO TO ALL!FOUND.
            GET PART.
            Add PARTNUM in PART to result list.
            GO TO NEXT.
ALL!FOUND:


SAIL Program:
            INDUSTRY←"ELECTRICAL";
            FIND!CALC(WAREHOUSE);
            IF EMPTY!SET(SUPPLIER) GO TO NONE!SUPPLIED;
            WHILE TRUE DO BEGIN
                FIND!NEXT(PART,SUPPLIER);
                IF ERROR!STATUS = 0307 THEN DONE;
                GET(PART);
                ADDTOLIST(HEAD,PARTNUM);
                END;
```

2.4. Control Structures

In addition to the ususal control structures associated with ALGOL-like languages (e.g. for loops, while loops, case statements, recursive procedures, etc.), SAIL has capabilities to enable parallel processing, backtracking, and coroutines. In SAIL, a process is a procedure that may be run independently of the main procedure. Thus several processes may be run concurrently. Note that the main procedure is also a process.

A process is created with a SPROUT statement as follows:

```
SPROUT(<item>,<procedure call>,<options>)
```

where <item> names the process for future reference, <procedure call> indicates what the process is to do, and <options> is used to specify attributes of the SPROUTed and current process. Unless otherwise stipulated (in <options>), a SPROUTed process begins to run as soon as it is SPROUTed and in parallel with the SPROUTing process.

Similarly, there exist primitives which result in the suspension of a process, the resumption of a process, and in the blocking of a process until a number of other processes have terminated. These tasks are accomplished by the SUSPEND, RESUME, and JOIN primitives respectively.

SUSPEND and RESUME have as their arguments single items while JOIN has a set of items as its argument. These items are the names that have been set up for the process by an appropriate SPROUT command.

For example, a procedure to tighten a bolt may be defined as follows:

```
ITEM P1,P2;
   .
   .
   .
SPROUT(P1,GRASP(HAND1,SCREWDRIVER));
SPROUT(P2,GRASP(HAND2,BOLT));
   .
   .
   .
JOIN({P1,P2});
TURN(HAND1,CLOCKWISE);
   .
```

Since SAIL runs on a single processor computer system, true multiprocessing is not possible. Instead, the SAIL runtime system contains a scheduler which decides which process is to run and for how long. The programmer makes use of the <options> field of the SPROUT statement to specify information which the scheduler uses to determine the next process to be run. Such information includes time quantum sizes, priority, whether or not to immediately run the SPROUTed process, etc.

A process may result in the binding of ITEMVARs by use of a MATCHING PROCEDURE which is basically a Boolean procedure. When one of the parameters is an unbound FOREACH itemvar, then upon success the parameter will be bound . The matching procedure is actually SPROUTed as a coroutine process and SUCCEED and FAIL are variants of RESUME which return values of TRUE or FALSE respectively. In addition, FAIL causes the process to terminate whereas when the matching procedure is called by the surrounding FOREACH via backup, then the procedure is resumed where it left off on the last SUCCEED.

For example, consider a box containing a number of different fasteners (nails, regular screws, bolts, nuts, tacks, etc.). The goal is to obtain Phillips screws. This can be achieved by the following MATCHING PROCEDURE which returns a different Phillips screw each time it is invoked.

```
MATCHING PROCEDURE GET!FASTENER (?ITEMVAR FASTENER,F!TYPE);
BEGIN
    FOREACH FASTENER,F!TYPE | FASTENER IN BOX AND
                              TYPE XOR FASTENER EQV F!TYPE
        DO SUCCEED;
    FAIL;
    END;
```

Note that FASTENER is a FOREACH ITEMVAR which upon success will be bound.

Backtracking is supported by variables of type CONTEXT. However, the programmer must specify the points to which backup is to occur (for example, recall SUCCEED). State saving and restoring is achieved by use of CONTEXT variables which act as pointers to storage locations of undefined capacity in which are stored the entities to be saved and restored. Actual state saving and restoring is accomplished by use of the primitives REMEMBER and RESTORE.

Processes may communicate with each other by use of the SAIL event mechanism. This is a message processing system which enables the programmer to classify the messages and to wait for certain events to occur. Events occur via the CAUSE construct which has as its arguments the event type, the actual notice, and instructions with respect to the disposition of the event. Similarly, there is a construct known as INTERROGATE which specifies a set of event types and instructions with respect to the disposition of the event notice associated with the designated event types. A variant of this facility has been used extensively in the implementation of the Stanford Hand Eye Project [Feldman71].

2.5. System Building Capabilities

SAIL includes many features which are designed to aid in system building. Assembly language statements may be interspersed with regular SAIL statements by use of the START!CODE and QUICK!CODE constructs. A number of different files which are to be used with the program can be specified via use of REQUIRE statements.

The statements:

```
REQUIRE "TOOLS" LOAD!MODULE;
REQUIRE "CAMLIB[1,3]" LIBRARY;
```

will cause SAIL to inform the loader that the file TOOLS.REL must be loaded. In addition, the file CAMLIB on disk area [1,3] serves as a library and is searched for needed routines.

The statement:

```
REQUIRE "HEADER.SAI" SOURCE!FILE;
```

will cause the compiler to save the state of the current input file, and scan HEADER.SAI for program text. When HEADER.SAI is exhausted, scanning of the original file resumes directly following the REQUIRE statement. This feature is particularly useful when dealing with libraries since in this case the REQUIREd file can contain EXTERNAL declarations thereby freeing the application programmer from such work and possible errors.

A rather extensive conditional compilation capability is associated with SAIL. This enables the development of large programs which can be parameterized to suit a particular application without compiling unnecessary code and thereby wasting memory for program segments which are never used. This capability is used to enahance a macro facility to include compile-time type determination; for loops, while statements, and case statements at compile-time; generation of unique symbols, and recursive macros. For example:

```
DEFINE GRASP(SIZE) = [IFCR SIZE > 1 THENC VISE
                      ELSEC PLIERS
                      ENDC];
```

results in the definition of a macro named GRASP having one formal parameter, SIZE. The result is the name of a tool that is appropriate for the size of the item that is to be grasped - i.e. a vise in case size is greater than 1 (assuming size is measured in centimeters, etc.) and pliers otherwise. For example:

```
TOOL1←GRASP(10.0);
TOOL2←GRASP(0.5);
```

will result in the following statements:

```
TOOL1←VISE;
TOOL2←PLIERS;
```

Note that the choice is made at compile-time and thus the programmer need not be concerned with the available grasping mechanisms Thus the program compilation step can be used to aid in the writing of the program. The example illustrates the importance of such a feature when certain tasks can be achieved by similar, yet not identical, means.

SAIL also provides an excellent interface with the operating system. This enables its use for real-time applications such as control of external devices. Interrupts can be handled and the user has at his disposal all of the I/O capabilities that an assembly language programmer has. This enables the development of programs ranging from scanners to mechanical arm controllers. In addition to compatibility with assembly language debuggers, SAIL has a high-level breakpoint package known as BAIL [Reiser75].

2.6. Standardization

Currently, SAIL has only been implemented on the PDP-10. It runs under both the TENEX [BBNEXEC] and TOPS-10 [TOPS10] operating systems. There is an effort underway at SUMEX to develop a language similar to SAIL known as MAINSAIL [Wilcox76]. The goal of that project is to capture the features that make SAIL an attractive language (in particular the ease of interaction with the operating system) and to develop a language that is capable of being run

on a large number of machines. The orientation of the project is towards mini-computers. The language is considerably different than SAIL and existing SAIL programs will have to be modified in order to be capable of compiling. An extensive run time library is being provided as is a record structuring facility. It is still uncertain whether the associative data base capability of SAIL (i.e. LEAP) will be incorporated in MAINSAIL.

3. The LISP Family of Languages

3.1. LISP

LISP ([McCarthy60], [Levin65], [Weissman67], [Siklossy76]), which stands for List Processing Language, was developed by John McCarthy at MIT in the late 50's, having been inspired by Alonzo Church's work [Church41] in lambda calculus. McCarthy's intention was to recast this elegant recursive function theory as a theory of computation. Thus, the first implementations of LISP relied exclusively upon recursion as the computational paradigm (i.e. no iteration), which, although elegant, resulted in a first version of LISP which was not competitive with FORTRAN as a practical programming tool. Since then, LISP's character has changed considerably, so that today LISP is an extremely powerful and general programming language which retains its original elegance.

The most interesting features of LISP are:

(1) The language is practically devoid of syntax; all constructions in LISP fall into two categories: atoms and compositions of atoms.

(2) Program and data are totally interchangeable, since they are represented in one and the same format. Therefore, in LISP it is possible for one function to construct another function as data, then execute it by indicating to the LISP system to regard it as code; alternatively, an existing function's code may be examined, modified or augmented by another function at run-time. In fact, a function is capable of self-modification if appropriate care is exercized.

(3) Memory allocation and management are automatic and transparent to the user, except where the user explicitly desires to influence them. With the exception of arrays, there are no space declarations to be made; this frees the programmer from having to worry about details, and generally allows for the unlimited growth of any given data structure. (For the most part, LISP data structures have no size or complexity constraints.) Used memory which is no longer involved in the computation is recycled automatically by a garbage collector either on demand from the user at specified points or automatically.

(4) LISP is an interpreted language. The system proper is a function of one argument, (EVAL X), such that calling EVAL with any LISP data structure as its argument causes that argument to be regarded as code and executed. However, most LISP systems include a compiler which will produce stand-alone machine code for interpreted functions. Typically, compilation provides an order of magnitude speedup which makes LISP competitive with other compiled languages, or even with well-coded assembly language. Since interpreted and compiled code may be intermixed, it is possible to retain the flexibility and power of the interpreter, while obtaining the speed required for production applications.

(5) LISP remains recursive, while also accomodating iterative algorithms via a so-called PROG feature. Both recursion and iterative programming will be illustrated later.

(6) Because of the technique LISP uses in storing local and global variables, some very powerful context-switching can be carried out, providing a fast way to enter and exit hypothetical planning environments and to cause the behavior of a program to vary as a function of its environmental context.

3.1.1. LISP Data Structure

LISP's data structure, called the S-expression, is simple, yet extraordinarily flexible, providing a substrate upon which a programmer may design his own complex data structures. An S-expression is either a so-called "atom" or a "CONS node". An atom can be regarded either as a variable, as a constant (a passive symbol), or both. There are no declarations in LISP; new atoms are simply admitted to the system as they are scanned at the input level, and atoms with the same name are guaranteed by the system to be unique (i.e. have the same internal pointer, or address).

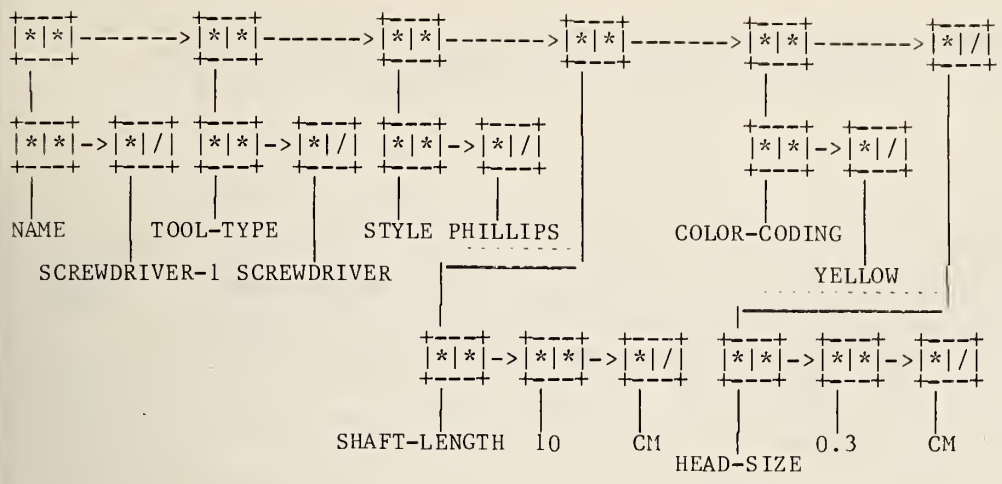
The other type of S-expression, the CONS node, provides a means of structuring atoms and other CONS nodes into hierarchical data structures. A CONS node is ordinarily implemented as a single computer word (say, 36 bits long) which contains a left pointer, called its CAR, and a right pointer, called its CDR. CONS nodes are created dynamically via the function (CONS X Y), where X and Y are any other S-expressions, or passively (as data constants) via the construction (X.Y). CONS nodes can be composed to form arbitrarily complex hierarchies, the bottommost elements of which are usually atoms (i.e. pointers to atomic S-expressions).

To illustrate, suppose we wish to characterize a particular tool, say a screwdriver, in a LISP data structure. We first decide upon a name for it, say, SCREWDRIVER-1, and what characteristics of it we wish to encode. Let us suppose the characteristics are: its type is Phillips, its color is yellow, its shaft length is 10 centimeters, and its head size is 0.3 centimeter. There are many structures in LISP which would accomodate this information; the external representation of the one we will adopt is:

```
((NAME SCREWDRIVER-1)
 (TOOL-TYPE SCREWDRIVER)
 (STYLE PHILLIPS)
 (SHAFT-LENGTH 10 CM)
 (COLOR-CODING YELLOW)
 (HEAD-SIZE 0.3 CM))
```

Here, all the symbols such as NAME, YELLOW, etc. are LISP atoms. (So too are the numbers; however numbers are not entirely equivalent with symbolic atoms.) The particular hierarchy we have adopted would be characterized as a list of lists, where each sub-list consists of an initial atom describing that sub-list's role in the structure, and a list of the information associated with that role in the description.

This structure would be graphically represented as follows:



and could be constructed passively (as a fully constant structure) via a quoted S-expression:

```
'((NAME SCREWDRIVER-1) (TOOL-TYPE SCREWDRIVER) ...)
```

or dynamically via CONS:

```
(CONS (CONS 'NAME (CONS 'SCREWDRIVER-1 NIL))
      (CONS 'TOOL-TYPE (CONS 'SCREWDRIVER NIL))
      ...
      (CONS 'HEAD-SIZE (CONS 0.3 (CONS 'CM NIL)))
)
```

Since it would be a rather harrowing experience to construct very large S-expressions dynamically in this fashion, LISP provides quite a spectrum of higher-level functions for constructing, modifying and accessing S-expressions. Some highlights of these will be covered briefly in a subsequent section. For our example, a more concise expression of code which would build this structure dynamically would be:

```
(LIST (LIST 'NAME 'SCREWDRIVER-1)
      (LIST 'TOOL-TYPE 'SCREWDRIVER)
      ...
      (LIST 'HEAD-SIZE 0.3 'CM)
)
```

Presumably, having defined this tool, we would want to record it as one available tool in a large supply of tools. Again, there would be numerous methods of doing this. One way would simply be to maintain a global list of all known tools in the system, and to add this entire description as a new tool on this list:

```
(SETQ NEW-TOOL '((NAME SCREWDRIVER-1) (TOOL-TYPE SCREWDRIVER) ...))
(SETQ MASTER-TOOL-LIST (CONS NEW-TOOL MASTER-TOOL-LIST))
```

(SETQ is one of LISP'S assignment statements.) Alternatively, we might wish to put only the name of the screwdriver on the master tool list, and associate all the remaining information on SCREWDRIVER-1's property list:

```
(PUT 'SCREWDRIVER-1 'DESCRIPTION
     '((TOOL-TYPE SCREWDRIVER) ... (HEAD-SIZE 0.3 CM)))
(SETQ MASTER-TOOL-LIST (CONS 'SCREWDRIVER-1 MASTER-TOOL-LIST))
```

3.1.2. Property Lists

Any LISP atom may have a property list (built up from CONS nodes). Conceptually, the property list allows the attachment of an arbitrary number of attribute-value pairs to the atom, thereby serving to describe the characteristics of the real-world entity the atom represents. This is a powerful feature for any programming language, since it allows "micro-descriptions" of atoms which ordinarily will not be seen by the processes that manipulate the hierarchical structures in which the atom participates. These microdescriptions can be maintained and accessed by the functions PUT, GET and REMPROP in case more detail about an atom is desired.

Properties are attached to an atom via the function (PUT <atom> <attribute> <value>), looked up via (GET <atom> <attribute>) and removed via (REMPROP <atom> <attribute>). We have seen one way to associate the screwdriver information with the atom SCREWDRIVER-1 using property lists. Another, more convenient way would be to split apart all the various attributes of this atom, making each a different entry on the property list:

```
(PUT 'SCREWDRIVER-1 'TOOL-TYPE 'SCREWDRIVER)
(PUT 'SCREWDRIVER-1 'STYLE 'PHILLIPS)
...
(PUT 'SCREWDRIVER-1 'HEAD-SIZE '(0.3 CM))
```

Then, for instance, to determine SCREWDRIVER-1's head size, we would write: (GET 'SCREWDRIVER-1 'HEAD-SIZE). If such an attribute of SCREWDRIVER-1 exists, it will be located and returned.

3.1.3. Representative LISP Data Structure Manipulating Functions

We include here a definition and brief example of several of the more standard, higher-level LISP functions that have to do with data structure creation, modification and searching.

3.1.3.1. (MEMBER X Y)

If S-expression X is a member of S-expression Y (assumed to be in the form of a list), return "TRUE", otherwise, return "FALSE".

EXAMPLE: (MEMBER 'SCREWDRIVER-1 MASTER-TOOL-LIST) would return a pointer to the atom T ("true") if SCREWDRIVER-1 is on the MASTER-TOOL-LIST, a pointer to the atom NIL ("false") otherwise.

3.1.3.2. (ASSOC X Y)

Y is assumed to be a list of lists. Y is scanned, comparing the first item of each sublist to X until a match is found, or until Y is exhausted. In case a match is found, ASSOC returns the entire sublist whose first item matched X.

EXAMPLE: (ASSOC 'HEAD-SIZE '((NAME SCREWDRIVER-1) ... (HEAD-SIZE 0.3 CM)))
would return the sublist (HEAD-SIZE 0.3 CM).

3.1.3.3. (SUBST X Y Z)

X, Y and Z are all arbitrary S-expressions. SUBST creates a new copy of Z, except that all occurrences of Y in Z are substituted with X's.

EXAMPLE: (SUBST 0.2 0.3 '((NAME SCREWDRIVER-1) ... (HEAD-SIZE 0.3 CM))) would produce a new structure for our screwdriver, identical in all respects to the original, except that its head width would be 0.2 instead of 0.3.

3.1.3.4. (APPEND X Y)

X and Y are assumed to be lists. A new list is created which is the result of appending Y onto the end of X.

EXAMPLE: (APPEND '((NAME SCREWDRIVER-1) (STYLE PHILLIPS)) '((COLOR-CODE YELLOW) (HEAD-SIZE 0.3 CM))) would produce ((NAME SCREWDRIVER-1) (STYLE PHILLIPS) (COLOR-CODE YELLOW) (HEAD-SIZE 0.3 CM))

3.1.4. LISP Data Types

In addition to atoms and CONS nodes, most LISP systems include the following other data types:

1. integer numbers
2. real numbers
3. strings
4. arrays
5. octal numbers (for bit-level manipulations)

Some LISPs (notably MACLISP [Moon74]) have highly developed numerical and trigonometric facilities and accompanying optimizing compilers geared to the efficient generation of "number crunching" software.

3.1.5. LISP Functions

A LISP "program" is a collection of functions. No main program is distinguished, and the only distinction concerning a function's "type" concerns whether or not its calling arguments are to be evaluated before it is called. Otherwise, all functions in LISP are typeless, and hence there is no need for function-related declarations.

A function is regarded as simply another type of data. As such, one typically defines a function by assigning to some atom the function as the atom's value. Strictly speaking, the function itself is nameless, and is identified by the form:

(LAMBDA <argument-list> <body>)

When a "lambda expression" is stored as the value of some atom, we can say that a function has been defined. Various LISPS's carry out such details differently, but a common format for defining a function in LISP is:

```
(DEFUN <name> <arguments> <body>)
```

DEFUN is a macro which creates the appropriate lambda expression and assigns the atom <name> this expression as its body. A function may be annihilated or altered simply by reassigning the value of the atom which represents it. Another virtue of this separability of a function from its name is that nameless functions can be created and passed as arguments to other functions without having to bother to name them if they are needed only once.

To illustrate LISP functions, suppose we wish to define a function of two arguments, (LOCATE-ALL <tool-type> <tool-list>), which, given the name of a tool type (e.g. SCREWDRIVER), and a master tool list, will search the tool list for tools of the specified type and report back a list of all tools of that type it finds. Then, framing this as a recursive function, we would write:

```
(DEFUN LOCATE-ALL (TYPE MASTER-LIST)
  (COND ((NULL MASTER-LIST) NIL)
        ((EQUAL (GET (CAR MASTER-LIST) 'TOOL-TYPE) TYPE)
         (CONS (CAR MASTER-LIST)
               (LOCATE-ALL TYPE (CDR MASTER-LIST))))
        (T (LOCATE-ALL TYPE (CDR MASTER-LIST)))))
```

that is, if (COND) the master list is (or has been reduced to) NIL, then report back "nothing"; otherwise, if the next item on the master list (its CAR) is of the correct type (as determined by the GET), then add this tool to the list to be reported (i.e. CONS it onto the front of this list) and proceed with the search on the remainder of the list (its CDR); otherwise (T...), simply proceed, without recording the current tool.

Alternatively, we could express this algorithm in iterative form via the PROG feature:

```
(DEFUN LOCATE-ALL (TYPE MASTER-LIST)
  (PROG (RESULT)
    LOOP (COND ((NULL MASTER-LIST) (RETURN RESULT))
               ((EQUAL (GET (CAR MASTER-LIST) 'TOOL-TYPE) TYPE)
                (SETQ RESULT (CONS (CAR MASTER-LIST) RESULT))))
         (SETQ MASTER-LIST (CDR MASTER-LIST))
         (GO LOOP)))
```

i.e., enter a PROG (akin to an ALGOL begin-end block), defining one temporary local variable, RESULT; then, while the master-list remains non-nil, repeatedly examine its next item, collecting those with the correct type on the RESULT list (via the SETQ, the LISP "assignment statement"), scanning to the next tool on the master list (SETQ MASTER-LIST (CDR MASTER-LIST)).

3.1.6. The PROG Feature

As just illustrated, LISP accomodates iteratively-phrased algorithms via a construction called a "PROG". A PROG has the form:

```
(PROG <local-variables> <statement-1> ... <statement-n>)
```

As a PROG is entered, the local variables (if any) are created for the scope of the PROG, and each is initialized to NIL. Next, the statements which

comprise the PROG's body are sequentially executed (evaluated) until execution either "falls off the bottom" of the PROG (an implicit exit from the PROG), or until a GO or RETURN is encountered. Statements which are atoms are interpreted as labels within a PROG, and are ignored during sequential execution. When a GO is encountered, a branch to the specified label occurs, and sequential execution proceeds from that point.

Since a PROG introduces some temporary variables which must be reclaimed as the PROG is exited, there must be some way of informing LISP that a PROG is about to be exited. The function RETURN is used for this purpose, informing the system that a PROG is being exited, and specifying what value the PROG is to pass back out to the calling environment.

PROG's may be nested and may appear at any point in a LISP program, and the PROG construction will typically result in a more efficient implementation of an algorithm than the corresponding recursive implementation. Although some feel that PROG makes LISP "impure", in reality it is the feature which is probably most responsible for LISP's widespread acceptance in the AI community and elsewhere today.

3.1.7. LISP Macros

Most LISPs support two types of macros: compile-time macros and scanner macros. A compile-time macro is nothing more than a function which, when evaluated, computes not a final result, but another S-expression which, when evaluated, will compute a final result. Thus, when a macro is encountered by the LISP interpreter, a double evaluation is performed (the first to compute the intermediate form, the second to run the intermediate form). When LISP functions are compiled into actual machine code, the compiler recognizes macros and evaluates them once to obtain the intermediate form which it then compiles. This technique is a very general and very powerful implementation of the macro concept.

Most LISP scanners are quite modular, in the sense that they can be conditioned to initiate an arbitrary computation upon encountering a given character in the input stream. In Wisconsin LISP [Norman69], for example, there is a facility called (READMAC <char> <function>), which conditions the scanner to call <function> (no arguments) whenever <char> is detected in the input stream. <function> is free to perform any computation, and whatever <function> returns is spliced into the scanner's input stream. This style of table-driven scanner makes it possible to superimpose additional syntax on LISP input, even to the point where LISP can model another language's syntax (by redefining delimiters, etc.). MLISP [Smith70] is an example of this.

3.1.8. Variable Scoping

LISP variables' values are derived as a function of the run-time environment rather than as a function of lexical environment. As a program executes, there are two times at which new variables are introduced, or "bound": (1) at function entry time (these are the function's arguments' names mentioned in the LAMBDA expression), and (2) at PROG entry time (i.e., the PROG's temporary variables). Variables are "unbound" at the corresponding exit times: when a function returns or when a PROG is exited.

At the "top-level" of LISP (when no function is currently executing), any variables which receive values are thought of as "global" to the system. Therefore, at any given moment during execution, there will be a pool of global atoms plus all the atoms introduced via LAMBDA's or PROG's on the current sequence of function calls. All these variables and their associated values ("bindings") are recorded on a structure called the "association list" (A-LIST). This, like most everything else in LISP, is a user-accessible list of CONS nodes. All variable lookups consult this list, from most recent to least recent. Since this list is dynamically maintained at run-time, the question of what variables are and are not bound (i.e. are on the A-LIST) is exclusively determined by the dynamic calling environment, rather than the

lexical scope of variables at the time functions were defined. This means that "free" variables (ones which have no binding at the current level) will assume a value at run-time which is dependent upon their definitions in functions farther up the calling hierarchy. In this manner, one function "peeks into", or borrows another's variables.

By changing the system's A-LIST pointer while inside a function, that function's entire environment can be altered. For this reason, LISP is a very powerful tool wherever hypothetical reasoning (involving switches to altered contexts) is necessary. Most other languages either lack such an ability, or make it difficult to carry out. In LISP, context switching and "taking snapshots" of contexts to which execution is to be returned are very natural operations.

3.1.9. LISP I/O

Traditionally, input/output has been LISP's weakest link. Most systems define at least the following I/O-related functions:

(READ)	read an S-expression
(READCH)	read an individual character
(PRINT X)	print S-expression X, skipping to a new line
(PRIN1 X)	print S-expression X on the current output line
(TERPRI)	skip to beginning of new line on output

While these functions provide adequate formatting control, most LISPs are deficient in file-handling operations. (INTERLISP [Teitelman74] is the exception, with some more highly developed interfaces with the TENEX virtual operating system). We regard this deficiency as more of a historical accident than as an inherent problem of LISP (since adding these features is simply a matter of writing the code). In fact, there are efforts underway for improved multiple-file interaction and random access facilities both at MIT (MACLISP) and at Maryland (Wisconsin LISP).

3.1.10. Garbage Collection

Since LISP data structures can grow in unrestricted ways, a crucial part of any LISP system is a conceptually asynchronous process called the "garbage collector". The role of this process is periodically to take control, mark parts of storage that are still referenced by the ongoing computation, then reclaim all storage that is not so referenced (garbage). Garbage collection is an unavoidable overhead of any system with no declarations, and in which data structures can grow in unrestricted ways.

One potential disadvantage of garbage collection is that, once the system runs out of free storage, a garbage collection must occur. Since a garbage collect causes current computing activity to be suspended, if LISP is controlling a real-time process, disastrous consequencs can accrue. Such problems can normally be avoided by forcing the system into a premature garbage collect prior to entering real-time critical sections of computation. Alternatively, there is growing interest in truly asynchronous (parallel) garbage collection techniques which could obviate the problem altogether (see [Dijkstra75] for instance).

3.1.11. LISP as a Self-Contained System

LISP interpreters themselves are typically implemented in assembly language. After this basic facility has been brought up, most other supporting software can be written in LISP itself. Typical software includes

(1) A compiler which will generate (potentially quite good) machine code for LAMBDA expressions (i.e. functions) and PROGs. Typically, the LISP compiler will be written in interpreted LISP, then used to compile itself. Then the compiled version will be used as the LISP system compiler.

(2) A debug package which will permit the tracing and interactive development of functions. Typically, functions (together with their calling arguments) can be traced at entry time, and (together with their returned values) at return time. Most LISPs will also accommodate the tracing of variables (i.e. inform the user whenever a traced variable's value is about to be changed). The debugging potentials of LISP are essentially unlimited (the INTERLISP system is the most advanced to date), and are responsible (in part) for LISP's reputation as one of the best languages for the efficient and rapid development of complex software. In particular, there is no time-consuming interaction with system compilers, loaders and linkers to be contended with; a program can be developed and put into production within the confines of the LISP system itself.

(3) An S-expression editor (or system editor interface) which makes possible the convenient editing of S-expressions and maintenance of files.

3.2. MICROPLANNER

While LISP is generally accepted as the standard for computing in AI, it does not supply the user with any a-priori conceptions about intelligence. LISP is simply the blank tablet onto which the user must write his theory of intelligence or control. Not surprisingly, this resulted in numerous reinventions of the wheel in areas like database organization, problem solving, hypothetical reasoning, and language understanding. Most reinventions were at a fairly low level, but occurred often enough to warrant some investigations into some of the undercurrents of AI programming techniques.

MICROPLANNER [Sussman, Winograd, Charniak 71] is the outcropping of some of these undercurrents, particularly where automatic problem solving is concerned. MICROPLANNER was written in 1970-71 as a small-scale implementation of some ideas originally proposed by Hewitt in 1969 [Hewitt69]. The intent of the language was and is to provide some automatic mechanisms of database organization, context, and heuristic search.

MICROPLANNER is implemented entirely in LISP. Because of this, its syntax is essentially LISP's syntax, and while in the MICROPLANNER environment, the user has full access to all of LISP. To distinguish MICROPLANNER (hereafter abbreviated MP) functions from pure LISP functions, the convention is to prefix all MP functions (there are about 50 of them) with "TH" (standing, we presume, for "theorem", a key notion in MP).

The most salient features of MP are these:

(1) Computation in MP is induced by pattern, rather than by calling functions by their names. In this style of computation (often called "pattern-directed invocation"), whenever something needs to be done, a pattern which describes the need is posted to the entire system. "Entire system" normally means a large population of problem-solving experts with patterns which advertise each one's expertise. Whenever a need is posted, the system searches through the database of experts looking for those whose advertised patterns match the need. Each expert so located is then tried in turn until one succeeds, or until all have failed. This is a radically different computing paradigm from the standard paradigm of "name calling", since it makes for a very modular system where the requestor needn't know any experts by name; problems are solved by anonymous experts in the population at large.

(2) MP automatically maintains a context-sensitive database of both factual assertions and the experts just mentioned. The factual database is a collection of highly indexed n-tuples, expressed as LISP S-expressions. Any one n-tuple ("assertion"), or collection of n-tuples can be "associatively" accessed by presenting the lookup routines with a pattern containing zero or more variables. Only those facts that are deemed active in the current "context", regardless of whether they physically exist in the memory, will be located.

(3) MP does all the bookkeeping required for depth-first, nondeterministic programming. That is, anytime there is a decision of any sort in MP, the system makes a choice (either arbitrarily, or under the control of user-specified heuristics), records the alternatives for possible future reference, and then proceeds. If a failure ever causes a "backup" to that decision point, the system automatically discards the current (failing) choice, selects the next alternative, and then attempts to proceed again. In the backup process, all computations performed between the initial (bad) choice and the failure point are undone (a record of all changes to the database is maintained), and the system picks up from the decision point as though nothing had ever gone wrong. Thus, MP can be said to maintain, at least implicitly, an entire goal tree (search tree) for each problem it attempts to solve. As we will suggest later, there are both advantages and disadvantages to such automatic control.

These are the three main contributions of MP. In the following sections we will highlight and illustrate some of the specific features of this problem solving language.

3.2.1. The MICROPLANNER Database

Conceptually, the MP database is divided into two segments: facts and theorems. Theorems are further classified into three categories: "antecedent" theorems, "erasing" theorems and "consequent" theorems. Theorems will be discussed in the next section.

Both facts and theorems are entered into the database via the function THASSERT; an item is deleted from the database via the function THERASE. Facts are fully-constant LISP n-tuples. Thus, to represent our screwdriver in MP, we might augment the database as follows:

```
(THASSERT (TOOL-TYPE SCREWDRIVER-1 SCREWDRIVER))
(THASSERT (STYLE SCREWDRIVER-1 PHILLIPS))
...
(THASSERT (HEAD-SIZE SCREWDRIVER-1 0.3 CM))
```

Database lookups and fetches are accomplished via the function THGOAL. Therefore, if at some point in a MP program, we required a knowledge of SCREWDRIVER-1's head width, we could write a fetch pattern of the form:

```
(THGOAL (HEAD-SIZE SCREWDRIVER-1 (THV X) (THV Y)))
```

For our example, this would respond with "success" (i.e. a fact which matched this template was located in the database, and it would produce the side effects of binding the MP variables X and Y to 0.3 and CM, respectively. The THV form is used in MP to signal references to variables (all else is implicitly constant).

Every fact and theorem in the MP database has a context marking. Whenever a fact or theorem is THASSERTed, if such a fact is not already

physically present in the database, it is created and then marked as also being logically present. If the THASSERTed fact is present physically, but marked as logically not present, its logical status is changed to "present". If the fact is already logically and physically present, THASSERT does nothing, but reports a "failure" to store a new copy of the fact. THERASE exerts opposite effects on facts in the database: it causes a fact to be logically masked, either by changing the fact's logical context marking, or by actually physically deleting the fact (i.e. if the fact is being THERASEd at the level at which it was originally THASSERTed).

Context markings allow MP to keep track of the history of the logical status of each fact and theorem. This enables the system to back up to prior context levels, thereby restoring the database to the corresponding prior state. Thus, although there are mechanisms for making permanent database changes (e.g., after some segment of MP code is confident that what it has done is absolutely correct), normally (except at the top level), THASSERT's and THERASE's are not permanent; instead, they normally exist only for the duration of some stretch of planning or hypothetical reasoning.

3.2.2. MICROPLANNER Theorems

All reasoning processes, indeed, all computations, in MP are carried out by THANTE, THERASING and THCONSE "theorems" which are called by pattern rather than by name. The three types of theorem are indistinguishable in internal form, except with regard to the type of event to which each responds. A THANTE theorem is triggered by the THASSERTion into the factual database of any pattern which matches its invocation pattern. A THERASING theorem is triggered by the THERASEure from the database of any factual pattern which matches its invocation pattern. In the sense that these two classes of theorems respond spontaneously (not in response to any particular request), they represent a general interrupt capability. A THCONSE theorem responds to THGOAL requests whose goal patterns match its invocation pattern.

Because of this last interaction between THGOAL's and THCONSE, a THGOAL can amount to considerably more than a simple database fetch. In MP, when a THGOAL is issued, the system first attempts to locate the desired goal directly as a fact in the database. If this fails, and the THGOAL request has indicated that it is permissible to do so, MP will begin searching for THCONSE theorems whose invocation patterns match the desired goal. If any are found, each is executed in turn until one reports success (in which case the THGOAL is satisfied), or until all THCONSE theorems have failed (in which case the THGOAL fails). It is in this manner that more complex knowledge (i.e. theorems, problem solving techniques, etc.) can be automatically brought to bear on some goal if that goal is not already explicitly present in the factual database.

The forms of these three MP theorem types are:

(THANTE <optional-name> <variables> <invocation-pattern> <body>)

(THERASING <optional-name> <variables> <invocation-pattern> <body>)

(THCONSE <optional-name> <variables> <invocation-pattern> <body>)

As a brief illustration of the uses of each of these, suppose we wish to implement the following three capabilities in MP: (a) whenever a new screwdriver is defined to the system, automatically cause its name to be added to the master tool list; (b) whenever a screwdriver is deleted from the system, automatically remove its name from the master tool list, and also remove all its accompanying information; (c) whenever, during some assembly task, a THGOAL of the form: (SCREW-IN <some screw> <some threaded hole>) is announced, automatically search for, and return the name of an appropriate screwdriver for the task (based on the screw's style and head size). Task (a) will be modeled as a MP THANTE theorem, part (b) by a THERASING theorem, and part (c) by a THCONSE theorem as follows:

```
(THANTE (X) (TOOL-TYPE (THV X) SCREWDRIVER)
   (SETQ MASTER-TOOL-LIST (CONS (THV X) MASTER-TOOL-LIST)))


(THERASING (X) (TOOL-TYPE (THV X) SCREWDRIVER)
  (THPROG (ST CC ... HS HSU)
     (SETQ MASTER-TOOL-LIST (DELETE (THV X) MASTER-TOOL-LIST))
     (THAND (THGOAL (STYLE (THV X) (THV ST)))
            (THERASE (STYLE (THV X) (THV STYLE))))
     (THAND (THGOAL (COLOR-CODE (THV X) (THV CC)))
            (THERASE (COLOR-CODE (THV X) (THV CC))))
      ...
     (THAND (THGOAL (HEAD-SIZE (THV X) (THV HS) (THV HSU)))
            (THERASE (HEAD-SIZE (THV X) (THV HS) (THV HSU))))))


(THCONSE (SCREW HOLE) (SCREW-IN (THV SCREW) (THV HOLE))
  (THPROG (ST HS HSU DRIVER DST DHS DHSU)
    (THGOAL (STYLE (THV SCREW) (THV ST)))
    (THGOAL (HEAD-SIZE (THV HOLE) (THV HS) (THV HSU)))
    (THGOAL (TOOL-TYPE (THV DRIVER) SCREWDRIVER))
    (THAND (THGOAL (STYLE (THV DRIVER) (THV DST)))
           (EQUAL (THV DST) (THV ST)))
    (THAND (THGOAL (HEAD-SIZE (THV DRIVER) (THV DHS) (THV DHSU)))
           (EQUAL (THV DHS) (THV HS)))
    (THRETURN (THV DRIVER))))
```

3.2.3. Heuristic Guidance of Theorem Application

It is possible, by including special indicators in THGOAL, THASSERT and THERASE calls, to influence the order in which theorems are applied, or in fact to indicate whether or not they should be applied at all. Specifically, a THGOAL (similar remarks apply to THASSERT and THERASE) with no indicators will fail unless the requested goal can be satisfied exclusively by database fetches (no theorems will be applied). (This is the form we have been using for illustration purposes.) If there is an indicator present, it has either the form of a "filter" or a specific "recommendation list" of theorems (referenced by name). When a filter is included in a THGOAL request, only those theorems whose properties pass the filtering test (theorems can possess property lists) will be candidates for application. If the indicator has the form of a specific recommendation list, all theorems on that list will be applied first (in order) before any other theorems from the general theorem base are attempted. Both forms allow the programmer to insert limited heuristic influences. Also, since one MP theorem can create or modify another MP theorem, the filter facility provides a setting in which a collection of theorems themselves can evolve into a more structured configuration on the basis of past experience (e.g. who in the past has proven to be the most reliable expert). Although filtering and recommendations are a step in the right direction, as we will discuss later, CONNIVER provides a more flexible environment in which to encode heuristic knowledge.

3.2.4. Searching and Backup in MP

Search and backup in MP can occur for two reasons: (1) some THCONSE theorem which was run to accomplish a THGOAL fails, and another theorem must be invoked (restoring the environment to the sate at which the first theorem took over), or (2) some object to which the system has committed itself is discovered to be inappropriate, giving rise to the need of locating another candidate object and retrying. The THGOAL-THCONSE mechanism underlie the selection and backup where theorems are concerned, but object selection is handled differently, via the THPROG MP construction.

In the previous THCONSE example, the goal was to locate some screwdriver

which satisfied some set of features (in that case, the correct STYLE and HEAD-SIZE). This was accomplished by a THPROG which "conjectures" that such an object, say X, exists, then proceeds to determine whether or not this conjecture is true. In the example above, the THPROG searched for a screwdriver of type and size which matched the type and size of the particular screw which was to be inserted. For the sake of illustration, suppose the screw was of type Phillips of head size 0.3. Then, the THPROG in the example above would have performed essentially the same search as the following, more specific, THPROG:

```
(THPROG (X)
   (THGOAL (TOOL-TYPE (THV X) SCREWDRIVER))
   (THGOAL (STYLE (THV X) PHILLIPS))
   (THGOAL (HEAD-SIZE (THV X) 0.3))
   (THRETURN (THV X)))
```

i.e., introcude an initially uncommitted variable, X, to represent the object being searched for. First, obtain a candidate for X by finding something which is of TOOL-TYPE SCREWDRIVER (the first THGOAL does this). At that point, X will be tentatively bound to the first such an object found. Continue with this candidate until either all THGOAL's have been satisfied (in which case, the candidate is a success), or until some THGOAL fails (in which case, the system must back up and choose another candidate). Since some objects may pass the first THGOAL, or even two, but not all three, the system must automatically keep track of what object it is currently considering, and what other objects remain to be tested. This is the source of backups which are propagated because of bad object selections.

To keep track of theorem and object selection backups, MP maintains a decision tree, THTREE, which is essentially a record of every decision made, and what to do in case the decision leads to a failure. The strength of THTREE is, of course, that it frees the programmer from having to worry about failures: if there is a solution, it will eventually be found by an exhaustive search. The fatal weakness of THTREE is that it imposes an often undesirable depth-first ordering on the search (i.e. one subgoal must be solved in its entirety before any other subgoals can be attacked). This makes it difficult, if not impossible, to fabricate complexly intertwined solutions, since subgoals cannot communicate laterally in the tree. The MP organization is also quite awkward in its backup technique because of the depth-first organization of THTREE. Often, one small failure will cause an entire branch of THTREE to be undone, when in fact most of it was correct. It would be more desirable to be able to discard only the bad part of the tree, retaining the parts which are correct, so that wholesale resynthesis of large parts of the THTREE does not have to occur. Unfortunately, this is, again, very difficult, if not impossible to do in MP. CONNIVER has a better control structure in these respects.

3.2.5. Other Representative MP Capabilities

To complete our description of MICROPLANNER, we include two representatives of the other functions available in this language, together with a brief example of each.

3.2.5.1. (THFIND <mode> <variables> <skel> <body>)

THFIND provides a way of finding all objects in the system which satisfy a certain set of criteria. A THFIND is essentially a THPROG which is made to fail artificially after each successful location of an object which satisfies the criteria. <mode> indicates how many objects are to be located (e.g. "ALL", "(AT-LEAST <count>)",...); <variables> serve the same role as THPROG variables; <skel> specifies what form to return as each object is found; <body> contains the THGOAL's, etc. which define the criteria. THFIND returns either a failure (in case <mode> number of objects could not be found), or a

list of <skel>'s, each <skel> corresponding to one successful object thus found.

```
EXAMPLE: (THFIND ALL (X) (THV X)
             (THGOAL (TOOL-TYPE (THV X) SCREWDRIVER))
             (THGOAL (STYLE (THV X) PHILLIPS))
```

would return a list of all tools which were Phillips screwdrivers.

3.2.5.2. (THMESSAGE <variables> <pattern> <body>)

As subgoals are descended into (i.e. "on the way down" the goal tree), THMESSAGE statements have no effect. They are essentially "hooks" which will intercept failures beneath them in the goal tree as such failures propagate back up to the THMESSAGE via a (THFAIL THMESSAGE <pattern>). Upon being backed up to by a THFAIL, any THMESSAGE whose pattern matches the THFAIL pattern will take control (its <body> will be executed). Thus, the THMESSAGE-THFAIL combination provides a way of anticipating possible problems without actually checking for them beforehand. If all goes well beneath the THMESSAGE, it will never run; however, if someone gets into trouble beneath the THMESSAGE (in some way the THMESSAGE is prepared for), the THMESSAGE can correct the problem and then cause the part of the tree beneath it to be reattempted.

```
EXAMPLE:  ...       (anticipate difficulty in inserting a screw)
         (THMESSAGE (X Y) ((THV X) WILL NOT TURN IN (THV Y))
           (THGOAL (LUBRICATE (THV X)))       (attempt a remedy)
           (THGOAL (SCREW-IN (THV X) (THV Y))))  (retry)
          ...
          ...      (attempt to insert some screw in some hole)
          ...      (report a failure back up to the THMESSAGE)
         (THFAIL THMESSAGE ((THV SCREW) WILL NOT TURN IN
                                        (THV HOLE)))
          ...
```

would anticipate, detect, report, and correct a problem, then retry.

3.3. CONNIVER

The most recent stage in the evolution of the LISP family of languages was the result of McDermott's and Sussman's development of a language called CONNIVER [McDermott, Sussman 73]. CONNIVER's development was principally motivated by the control structure deficiencies of MP, as suggested in the earlier discussion of THTREE. Although there were some improvements in the database and pattern-directed invocation control (e.g. the pattern matcher is more sophisticated), the most significant feature of CONNIVER is its ability to maintain numerous computations in states of suspended animation, then to switch among them, working on many subgoals or alternate strategies in unison rather than one at a time. In such an environment, partial computations need not be undone simply because some small aspect of the problem solving has gone awry.

CONNIVER is less a programming language than it is a collection of ideas about control structure. (The language apparently has never been used for more than one or two significant programming tasks [Fahlman73]). Because of this, our discussion will omit most references to syntax, and highlight only the aspects of CONNIVER's control structure which are unusual or unique to it.

3.3.1. Frames, Au-revoir and Adieu

In a conventional programming language (MP included), one function calls another function either by name or pattern and waits until the called function returns control. In a conventional language, once a function returns, that copy of it dies; the function may be called anew, but the new call will cause a new "copy" of the function to begin. No memory of a function's current status can be preserved across call-return sequences. This type of control is usually carried out under the control of push-down stacks which record calling arguments and return addresses; calling a function causes stacks to be pushed, while returning from a function causes stacks to be popped, annihilating all control information.

In CONNIVER, things are quite a bit different. To call a function in CONNIVER is to create a so-called "frame" for the called function, rather than to push information onto a central stack. A function's frame will contain all the information needed to characterize the function at any moment (e.g. from what A-LIST it derives values for its free variables, to whom it is to return when it has finished, etc.) There are two important features of a frame. First, it is a user-accessible LISP data structure. This means that a function may alter its own or another function's frame in arbitrary ways, causing free variables to be looked up on some other function's A-LIST, or causing the identity of the function to which control is to be returned to be altered. Second, because there is no central stack which is chronologically pushed and popped at function entry/exit, execution control is free to meander from one function to the next without permanently closing any function. Thus, at any moment, there can be numerous suspended functions which may be resumed at the point at which they last relinquished control, or in fact, at an arbitrary labeled point within them.

As one might expect, this ability makes the context marking technique for items in the database more complex than in MP. In particular, since control may eventually be returned to any suspended function (the system in general has no way of knowing whether or not it actually will be), every fact in the database must have markings which specify for every suspended function, F, whether or not that fact is supposed to be logically present while F is running. To accomplish this type of marking, the MP context scheme was generalized from a stack-like arrangement to a tree of contexts. Basically, every fact lives on some branch of the tree, and functions have access to limbs of the tree. Although there is considerable overhead, the system manages to mask and unmask facts in the database in synchrony with the meandering of execution control from one function to the next.

To distinguish the permanent return of a function from the case where a function merely relinquishes control, reserving the option to continue, CONNIVER defines two methods of returning: ADIEU (final, permanent return) and AU-REVOIR (suspension). One very important application of the AU-REVOIR feature is in the (often costly) generation of alternatives. Rather than calling a function (such as THFIND in MP) to generate all possible candidates before any detailed filtering tests are applied (a procedure which may waste an inordinate amount of time in the initial collecting phase), in CONNIVER it is possible to call a "generator" function which will locate and return candidates one at a time, suspending itself across calls. This makes for a more intimate form of interaction between the generating and testing functions than is possible in MP, and can lead to more efficient searches because of this intimacy. To facilitate the use of generators, CONNIVER has some rather elaborate machinery for maintaining "possibilities lists", including a function, TRY-NEXT, which controls the extraction of possibilities from such lists.

Computation in CONNIVER is similar in most other regards to computation in MP. The counterparts of THANTE, THERASING and THCONSE theorems are, respectively, IF-ADDED, IF-REMOVED and IF-NEEDED "methods". Except for differences in syntax, and a more general pattern-directed invocation scheme, these three functions are the same as the MP versions. CONNIVER counterparts of MP's database and goal-statement functions, THASSERT, THERASE and THGOAL are, respectively, ADD, REMOVE and FETCH.

3.4. Efficiency of the LISP Language Family

Being an interpreted language, LISP is slower than, say, FORTRAN, by between one and two orders of magnitude. But compiled LISP can be competitive with a good FORTRAN compiler. Thus, in our opinion, LISP provides the best of both worlds, in the sense that the interpreter provides for easy program development and debugging, while the LISP compiler can transform debugged code into production-level efficiency.

MICROPLANNER and CONNIVER, on the other hand, are inherently less efficient, primarily because of the control structures they superimpose on LISP. The fatal flaw with MP is its backup system, which can be extremely slow; compilation will not typically remedy the problem. CONNIVER is slow for similar reasons: in addition to data structures, processes must also be garbage collected, and the elaborate context system must be maintained. In our opinion, although these two languages contain many noteworthy features, neither would be suitable as it stands for production use.

3.5. Standardization of the LISP Language Family

There are LISP systems for the following machines: PDP-10, PDP-11, UNIVAC 1106, 1108, 1110, CDC 6500, 6600, IBM 360, 370, SIGMA 5, and others. (Because it is a relatively easy language to implement, we would anticipate no significant development problems for any machine, including microcomputers.) Since LISP's syntax is nearly non-existent, there is exactly one dialect. Although there are minor differences in the semantics of how functions are defined, and how variables' values are accessed, such "incompatibilities" can normally be ameliorated in about one day's worth of macro-writing. Because of this, LISP can be characterized as a language which is fairly standard and transportable. Finally, most LISP systems have an accompanying compiler, usually written in LISP itself.

4. Related Languages

4.1. AL

AL is a high-level programming system for specification of manipulatory tasks, which has been developed at Stanford Artificial Intelligence Laboratory [Finkel74]. AL is a SAIL-like language and includes large runtime support for controlling devices.

Trajectory calculation is a crucial feature of manipulatory control. AL contains a wide range of primitives to support efficient trajectory calculations. As much computation as possible is done at compile-time and calculations are modified at run-time only as necessary.

Besides a dimensionless scalar data type (i.e. REAL), AL recognizes and manipulates TIME, MASS and ANGLE SCALARs, dimensionless and typed VECTORs, and ROT (rotation), FRAME (coordinate system), PLANE (region separator) and TRANS (transformation) data types. Proper composition of variables of these types gives a simple means of performing calculations of any type of movement.

Also included are PL/1-like ON-conditions, which allow monitoring of the outside world, and concurrent processes.

Example:

```
PLANE p1;
  .
  .          { statements initializing p1 }
  .
SEARCH yellow                    { SEARCH is a primitive which causes
                                   a hand to move over a specified
                                   area.  yellow is a hand }
   ACROSS p1                     { hand moves across plane }
   WITH INCREMENT = 3*CM         { every 3 cm }
   REPEATING
      BEGIN                      { do at every iteration }
         FRAME set;
         set_yellow;             { yellow is also coord system of hand }
         MOVE yellow XOR - Z*CM
                                 { move hand 1 cm down from current
                                   position along Z-axis }
            ON FORCE(Z) > 3000*DYNES
               DO TERMINATE;     { keep in touch with real world }
         MOVE yellow TO set DIRECTLY;  { move the hand back to where
                                         it was in a straight line }
      END;
  .
  .
  .
```

4.2. MLISP

MLISP (meta-LISP) is a high-level list-processing language developed at Stanford University [Smith70]. MLISP programs are translated into LISP programs which are then executed or compiled. The MLISP translator itself is

written in LISP.

MLISP is an attempt to improve the readability of LISP programs as well as alleviate some alleged shortcomings in the control structure of LISP (e.g. no explicit iterative construct). Since run-time errors are only detected by the LISP system (when actually executing the program), users frequently find themselves debugging the translated LISP code. This somewhat defeats the purpose of any high-level language.

All LISP functions are recognized and translated in MLISP, but the Cambridge prefix notation of LISP has been replaced by standard prefix, infix and function notation. Instead of (PLUS X Y) one may write X + Y, and (FOO 'A B C) becomes FOO('A, B, C).

MLISP also provides a powerful set of iterative statements and a large number of "vector operators." Vector operators are used to apply standard operators in a straightforward manner to lists. Thus, in MLISP, <1, 2, 3> +@ <6, 5, 4> yields <7, 7, 7>. +@ is the vector addition operator and <A, B, C> is equivalent to (LIST A B C) in LISP.

Example:

Given a list of the form <obj1, obj2, ..., objn>, this function will return a list of the form <<obj1, holder1>, ..., <objn, holdern>> where holderi is either PLIERS, VISE or NOTHING accordingly as needed to hold the object. % ...% is an MLISP comment.

```
EXPR HOLD-LIST(OBJ-LIST);                    % EXPR starts a regular func %
   BEGIN
      NEW S;                                 % local declaration %
      RETURN                                 % RETURN is a unary operator %
         FOR NEW OBJ IN OBJ-LIST COLLECT
                                             % OBJ is local to the FOR loop. %
                                             % OBJ will be bound in turn %
                                             % to each element of OBJ-LIST. %
                                             % COLLECT indicates that the %
                                             % result of each iteration is %
                                             % to be APPENDed to the previous %
                                             % result and this whole list %
                                             % returned as the result of %
                                             % the FOR. %
            IF (S GET(OBJ, 'SIZE)) LEQUAL 5
               THEN
                  <<OBJ, 'PLIERS>>
               ELSE
                  IF S LEQUAL 10
                     THEN
                        <<OBJ, 'VISE>>
                     ELSE
                        <<OBJ, 'NOTHING>>
   END;
```

4.3. POP-2

POP-2 is a conversational language designed by R. M. Burstall and R. J. Popplestone and developed at the University of Edinburgh [Burstall71].

POP-2 features an Algol-like syntax and draws heavily from LISP.

Integers, reals, LISP-like lists and atoms (called 'names'), function constants (lambda expressions), records, arrays, extensible data types and run-time macros are supported. A unique feature of the POP-2 system is the heavy use of a system stack, which the user may easily control to enhance the efficiency of programs.

A full complement of list-manipulation, numeric and storage-management functions are available.

Example:

Suppose one wanted to obtain a list of all machinery not currently functioning. A useful function would be,

```
COMMENT  returns a list of all elements of argument list x1
         which satisfy argument predicate p ;

FUNCTION sublist x1 p;              { arguments are x1 and p }
   VARS x;                          { declaration of local, no type }
   IF null(x1) THEN nil             { just like LISP }
      ELSE hd(x1) -> x;             { hd(a) = (car a) }
           IF p(x)
              THEN x::sublist(tl(x1), p)
                                    { tl(a) = (cdr a), x::1 = (cons x 1) }
              ELSE sublist(tl(x1), p)
           CLOSE
   CLOSE
END;
```

A call might then look like,

```
sublist(machine-list,
              LAMBDA m; not(functioning(m)) END);
```

which might return,

```
[punch-press1 drill-press2 unit10]
```

which is a POP-2 list.

5. Examples

5.1. Introduction

A common example will be used to illustrate the distinguishing features of the four major languages. With only minor variations the program-segments use the same algorithm. The program-segments appear out of context and are not meant to indicate the most eficient (or preferred) implementation of the problem in each language, but merely to illustrate the languages' major attributes.

Problem statement:

Given two distinct assemblies (say A1 and A2), attempt to unscrew A1 from A2, and indicate success or failure accordingly. The "world" of the example is assumed to include:

(1) Two hands (LEFT and RIGHT) capable of moving, grasping, twisting and sensing force and motion.

(2) A fixed number (possibly zero) of PLIERS

(3) A fixed number (possibly zero) of VISEs

(4) A fixed number of "assemblies"

For each PLIERS and VISE, the data base contains an assertion of the form, "PLIERS (VISE) # n is at location (X, Y, Z) and is of capacity C cm." In addition, for each assembly the data base contains an assertion of the form, "assembly A is at location (X, Y, Z) and is of size S cm." One of the distinguishing features of the languages is their method of representing this knowledge.

Each example assumes the existence of the following routines. They are described below in ALGOL-like notation.

ATTACHED(A1, A2) - TRUE if and only if the assembly represented by A1 (hereafter simply called A1) is attached to the assembly represented by A2 (likewise called A2). The routine has no side effects.

MOVE(HAND, LOCATION) - Moves HAND (either LEFT or RIGHT) to LOCATION (but see PLANNER's description of MOVE).

TWIST(HAND, DIRECTION) - Twists HAND (LEFT or RIGHT) in given DIRECTION (CLOCKWISE or COUNTER-CLOCKWISE). The DIRECTION is oriented looking down the length of the arm. Except for SAIL, all programs assume a routine called TWIST-BOTH, to cause both hands to twist at once.

GRASP(HAND, OBJECT) - Causes HAND (LEFT or RIGHT) to grasp OBJECT, which must be within some fixed range of HAND (i.e. the hand must MOVE to the OBJECT first).

ATTEMPT(OBJ1, OBJ2, A1, A2) - Attempts to do the actual unscrewing of assembly A1 from A2 using objects OBJ1 and OBJ2 (here, VISEs or PLIERS). Returns TRUE if and only if the attempt is successful.

Each program proceeds as follows:

(1) Try to unscrew the assemblies using the hands. This entails obtaining the location of the assemblies, moving the hands to their locations, grasping and then twisting.

(2) If the objects are no longer attached, return "success."

(3) It is assumed that the hands weren't strong enough, it is proposed to try two pairs of PLIERS. A search ensues for a suitable set of available PLIERS (i.e. large enough to hold the assemblies). If one set of PLIERS fails, the search is continued for another set, with the hope that the differences among PLIERS (grip, size, etc.) will eventually lead to success.

(4) The PLIERS failed. It is decided to hold one of the assemblies in a VISE. A search identical in nature to that in step (3) occurs

(5) At this point all attempts have failed. An appropriate message is output and "failure" is returned.

5.2. SAIL

5.2.1. Sample Program

```
1
2   INTEGER PROCEDURE BIGENOUGH(ITEMVAR HOLDER, HOLDEE);
3
4           " RETURN TRUE IFF OBJECT HOLDER IS LARGE
5             ENOUGH TO HOLD OBJECT HOLDEE "
6
7   BEGIN
8
9           INTEGER ITEMVAR C, S;
10
11          C _ COP(CAPACITY XOR HOLDER);
12          S _ COP(SIZE XOR HOLDEE);
13          RETURN(DATUM(C) GEQ DATUM(S))
14
15  END;
16
17
18
19
20  INTEGER PROCEDURE UNSCREW(ITEMVAR A1, A2);
21
22     " ATTEMPT TO DISASSEMBLE ASSEMBLY A1 FROM A2, BY UNSCREWING "
23
24  BEGIN
25
26          DEFINE RUNME = 1;
27
28          ITEMVAR V1, PL1, PL2, P1, P2;
29
30          INTEGER FLAG;
31
32          IF NOT ATTACHED(A1, A2) THEN RETURN(1);  " DON'T BOTHER "
33
34          MOVE(LEFT, LOCATION XOR A1); MOVE(RIGHT, LOCATION XOR A2);
35          GRASP(LEFT, A1);  GRASP(RIGHT, A2);
36
37          " GET BOTH HANDS TWITSTING AT ONCE "
38
39          SPROUT(P1, TWIST(LEFT, COUNTER!CLOCKWISE), RUNME);
40          SPROUT(P2, TWIST(RIGHT, COUNTER!CLOCKWISE), RUNME);
41          JOIN({P1, P2});
42          IF NOT ATTACHED(A1, A2) THEN RETURN(1);
43
44          " HANDS NOT STRONG ENOUGH, TRY PLIERS "
45
46          FOREACH PL1, PL2 |
47                  ISA XOR PL1 EQV PLIERS AND (BIGENOUGH(PL1, A1))
48               AND ISA XOR PL2 EQV PLIERS AND (PL1 NEQ PL2)
49               AND (BIGENOUGH(PL2, A2)) AND (ATTEMPT(PL1, PL2, A1, A2))
50          DO RETURN(1);
51
52          " EITHER THERE WEREN'T ANY PLIERS LARGE ENOUGH,
53            OR THE PLIERS WEREN'T STRONG ENOUGH.  TRY A
54            VISE ON ONE SIDE "
55
56          FOREACH V1, PL1 |
57                  ISA XOR V1 EQV VISE AND (BIGENOUGH(V1, A1))
58               AND ISA XOR PL1 EQV PLIERS AND (BIGENOUGH(PL1, A2))
59               AND (ATTEMPT(V1, PL1, A1, A2))
60          DO RETURN(1);
61
```

```
62          " ALL ATTEMPTS FAILED "
63
64          OUTSTR("CAN'T UNSCREW " & CVIS(A1, FLAG) & " & "
65                  & CVIS(A2, FLAG) & ('15 & '12));
66          RETURN(0)
67
68  END;
```

5.2.2. Commentary

2. In SAIL, FALSE = 0, TRUE <> 0. BIGENOUGH is really a BOOLEAN procedure.

9. C and S are items whose DATUM is assumed to be of INTEGER type.

11. COP(<set>) returns the first item of <set>. We are assuming there is only one triple of the form CAPACITY XOR <object> EQV <capacity> for each <object>.

13. C and S were needed since DATUM(COP(<set>)) is illegal. SAIL must know at compile-time what the type of a DATUM is. GEQ is a numeric test for greater than or equal.

20. UNSCREW is a BOOLEAN procedure which returns TRUE (non-zero) if it succeeds in unscrewing the objects.

26. This is a macro definition. Whenever RUNME is seen (by the SAIL compiler) it will be replaced by the constant 1, (see 39. for use)

39. SPROUT is a SAIL function which causes activation of its second argument (a procedure/function call) as a process. The first argument is an item whose DATUM will be set by SPROUT to contain information about the SPROUTed process (see 41. for use). The third argument to SPROUT determines the status of the current and the created process. RUNME (bit 35 set) indicates that the current and new process are to be run in parallel by the SAIL scheduler.

47. BOOLEAN tests in a FOREACH must be enclosed in parentheses.

48. Notice (PL1 NEQ PL2) to insure that two distinct pairs of pliers are found.

50. If the body of the FOREACH is entered, all went well so we just return success.

64. CVIS is a SAIL function which will return a character string 'name' associated with an item. FLAG is set by CVIS to indicate an error.

5.3. LISP

5.3.1. Sample Program

```
1
2       (DEFUN UNSCREW (A1 A2)
3
4         ?  ATTEMPT DISASSEMBLY OF OBJECT A1 FROM A2, BY UNSCREWING
5
6       (PROG (PL1 PL2 V1 IN)
7
8             (COND [(NOT (ATTACHED A1 A2)) (RETURN T)])
9
10            (MOVE 'LEFT (GET A1 'LOCATION))
11            (MOVE 'RIGHT (GET A2 'LOCATION))
12            (GRASP 'LEFT A1) (GRASP 'RIGHT A2)
13            (TWIST-BOTH 'COUNTER-CLOCKWISE)
14            (COND [(NOT (ATTACHED A1 A2)) (RETURN T)])
15
16          ?  HANDS NOT STRONG ENOUGH, TRY PLIERS
17
18            (COND [(FOREACH PL1 IN PLIERS-LIST (BIGENOUGH PL1 A1)
19                            PL2 IN PLIERS-LIST (AND (NOT (EQ PL1 PL2))
20                                                    (BIGENOUGH PL2 A2))
21                      DO (ATTEMPT PL1 PL2 A1 A2))
22                     (RETURN T)]
23
24          ?  PLIERS NOT LARGE ENOUGH OR NOT STRONG ENOUGH.
25          ?  TRY A VISE ON 1 SIDE
26
27                   [(FOREACH V1 IN VISE-LIST (BIGENOUGH V1 A1)
28                             PL1 IN PLIERS-LIST (BIGENOUGH PL1 A2)
29                      DO (ATTEMPT V1 PL1 A1 A2))
30                     (RETURN T)]
31
32            ?  ALL ATTEMPTS FAILED
33
34                   [T (PRIN1 "CAN'T UNSCREW ") (PRIN1 A1)
35                      (PRIN1 " & ") (PRIN1 A2) (TERPRI)
36                      (RETURN NIL)])
37      ))
38
39
40
41
42      (DEFUN BIGENOUGH (HOLDER HOLDEE)
43
44        ?  RETURN T IFF OBJECT HOLDER IS LARGE ENOUGH TO
45        ?  HOLD OBJECT HOLDEE
46
47        (NOT (LESSP (GET HOLDER 'CAPACITY)
48                    (GET HOLDEE 'SIZE)))
49      )
50
51
52
53
54
55      (DEFSPEC FOREACH (LAMBDA (OBJ1 IN1 LIST1 PRED1
56                                OBJ2 IN2 LIST2 PRED2
57                                DO TRY)
58
59        ?  MIMIC SAIL FOREACH IN SIMPLE CASE
60
61      (PROG (TEMP1 TEMP2)
```

```
62
63          (SETQ TEMP1 (EVAL LIST1))
64      LOOP1
65          (COND [(NULL TEMP1) (RETURN NIL)])    ? RAN OUT
66          (SET OBJ1 (CAR TEMP1))
67          (SETQ TEMP1 (CDR TEMP1))
68          (COND [(NOT (EVAL PRED1)) (GO LOOP1)])  ? FAILED 1ST TEST
69          (SETQ TEMP2 (EVAL LIST2))
70      LOOP2
71          (COND [(NULL TEMP2) (GO LOOP1)])
72          (SET OBJ2 (CAR TEMP2))
73          (SETQ TEMP2 (CDR TEMP2))
74          (COND [(NOT (EVAL PRED2)) (GO LOOP2)]
75                [(EVAL TRY) (RETURN T)]           ? IT WORKED
76                [T (GO LOOP2)])
77      )))
78
79
80
81
82      (DEFMAC FOREACH (LAMBDA (OBJ1 IN1 LIST1 PRED1
83                               OBJ2 IN2 LIST2 PRED2
84                               DO TRY)
85
86        ?  MACRO VERSION OF FOREACH
87
88      (LIST 'PROG '(L1 L2)
89          (LIST 'SETQ 'L1 LIST1)
90      'LOOP1
91          '(COND [(NULL L1) (RETURN NIL)])
92          (LIST 'SETQ OBJ1 '(CAR L1))
93          '(SETQ L1 (CDR L1))
94          (LIST 'COND (LIST (LIST 'NOT PRED1) '(GO LOOP1)))
95          (LIST 'SETQ 'L2 LIST2)
96      'LOOP2
97          '(COND [(NULL L2) (GO LOOP1)])
98          (LIST 'SETQ OBJ2 '(CAR L2))
99          '(SETQ L2 (CDR L2))
100         (LIST 'COND (LIST (LIST 'NOT PRED2) '(GO LOOP2))
101                     (LIST TRY '(RETURN T))
102                     '(T (GO LOOP2))))
103     ))
```

5.3.2. Commentary

2. The main function. Returns T iff disassembly was successful.

13. Unlike SAIL, LISP does not support concurrency. We thus assume a primitive function to get both hands twisting.

18. FOREACH is an iterative special form which mimics a simple SAIL FOREACH. FOREACH will try pairs of pliers until the given predicates succeed or it runs out of pliers (and returns NIL). Note that the arguments to a special form need not be quoted.

19. Check to insure that distinct pairs of pliers are found.

34. PRIN1 is a LISP function which loads its argument into the stream output buffer.

35. TERPRI is a LISP function which dumps the output buffer.

47. Return T if capacity >= size.

55. DEFSPEC defines a special form (sometimes called a FEXPR). A special form is identical to a LISP function except that its arguments are passed unevaluated.

63. EVAL is necessary since the argument was passed unevaluated.

66. Note SET and not SETQ. OBJ1 needs to be evaluated to get the intended atom (SET evaluates its first argument, SETQ does not).

68. Note EVAL (see 61.).

72. Note SET (see 64.).

82. This is an alternative macro version of FOREACH. It expands into a PROG which is similar in nature to the special form FOREACH.

5.4. PLANNER (MICROPLANNER)

5.4.1. Sample Program

```
1
2       (THCONSE UNSCREW (A1 A2)
3          (UNSCREW (THV A1) (THV A2))
4
5        ?  ATTEMPT DISASSEMBLY OF OBJECT A1 FROM A2, BY UNSCREWING
6
7       (THOR
8            (THNOT (ATTACHED (THV A1) (THV A2)))
9            (THAND
10                 (THGOAL (MOVE LEFT (THV A1)) (THTBF THTRUE))
11                 (THGOAL (MOVE RIGHT (THV A2)) (THTBF THTRUE))
12                 (GRASP 'LEFT (THV A1)) (GRASP 'RIGHT (THV A2))
13                 (TWIST-BOTH 'COUNTER-CLOCKWISE)
14                 (THNOT (ATTACHED (THV A1) (THV A2)))
15           )
16
17         ?  HANDS NOT STRONG ENOUGH, TRY PLIERS
18
19           (THPROG (PL1 PL2)
20                 (THGOAL (ISA (THV PL1) PLIERS) (THTBF THTRUE))
21                 (THGOAL (BIGENOUGH (THV PL1) (THV A1)) (THNODB)
22                                    (THUSE BIGENOUGH) (THTBF THTRUE))
23                 (THGOAL (ISA (THV PL2) PLIERS) (THTBF THTRUE))
24                 (THNOT (EQ (THV PL1) (THV PL2)))
25                 (THGOAL (BIGENOUGH (THV PL2) (THV A2)) (THNODB)
26                                    (THUSE BIGENOUGH) (THTBF THTRUE))
27                 (ATTEMPT (THV PL1) (THV PL2) (THV A1) (THV A2))
28           )
29
30         ?  NO PLIERS LARGE ENOUGH, OR NO PLIERS STRONG ENOUGH.
31         ?  TRY A VISE ON 1 SIDE
32
33           (THPROG (V1 PL)
34                 (THGOAL (ISA (THV V1) VISE) (THTBF THTRUE))
35                 (THGOAL (BIGENOUGH (THV V1) (THV A1)) (THNODB)
36                                    (THUSE BIGENOUGH) (THTBF THTRUE))
37                 (THGOAL (ISA (THV PL) PLIERS) (THTBF THTRUE))
38                 (THGOAL (BIGENOUGH (THV PL) (THV A2)) (THNODB)
39                                    (THUSE BIGENOUGH) (THTBF THTRUE))
40                 (ATTEMPT (THV V1) (THV PL) (THV A1) (THV A2))
41           )
42
43         ?  NOTHING WORKED, JUST FAIL
44
45           (THNOT (THDO
46              (PRIN1 "CAN'T UNSCREW ") (PRIN1 (THV A1))
47              (PRIN1 " & ") (PRIN1 (THV A2)) (TERPRI)
48           ))
49           (THFAIL THEOREM)
50      ))
51
52
53
54
55      (THCONSE BIGENOUGH (HOLDER HOLDEE C S)
56         (BIGENOUGH (THV HOLDER) (THV HOLDEE))
57
58         ?  SUCCEEDS ONLY IF OBJECT HOLDER IS LARGE ENOUGH TO HOLD
59         ?  OBJECT HOLDEE
60
61      (THGOAL (CAPACITY (THV HOLDER) (THV C)) (THTBF THTRUE))
```

```
62      (THGOAL (SIZE (THV HOLDEE) (THV S)) (THTBF THTRUE))
63      (THCOND [(NOT (LESSP (THV C) (THV S)))
64                   (THSUCCEED)]
65              [T (THFAIL THEOREM)])
66
```

5.4.2. Commentary

2. Defines and asserts a consequent theorem with name UNSCREW.

3. This is the pattern on which to invoke this theorem if needed. ((UNSCREW ASSEMBLY1 ASSEMBLY2) for instance.)

7. THOR sequentially executes each of its arguments until one succeeds, and then the THOR succeeds. The THOR is used here to prevent undesired backup.

8. (THNOT p) is defined as (COND [p (THFAIL)] [T (THSUCCEED)]) .

9. THAND succeeds if and only if all of its arguments succeed. Unlike THOR, backup may occur among the arguments of a THAND.

10. Attempt to move the left hand to object A1. There may be several experts (theorems) on moving hands, PLANNER will try as many as it needs (a powerful concept). (THTBF THTRUE) is a theorem base "filter" which is satisfied by every theorem.

19. THPROG behaves in a similar manner to THAND except that local variables may be declared.

20. Attempt to find a pliers.

21. See if the pliers is large enough. (THNODB) indicates to PLANNER not to bother searching the data base. (THUSE <theorem>) tells PLANNER to try <theorem> first.

24. Make sure we have two distinct pairs of pliers.

45. THDO executes its arguments and then succeeds. However, we know we have failed, so THNOT is used to generate a failure from THDO. This is all necessary since PRIN1 returns its first argument, which being non-NIL would cause the THOR to succeed.

49. Generate explicit failure of the theorem.

5.5. CONNIVER

5.5.1. Sample Program

```
1
2       (CDEFUN UNSCREW (A1 A2)
3
4         ?  ATTEMPT TO DISASSEMBLE A1 FROM A2, BY UNSCREWING
5
6                  "AUX" (LOC1 LOC2 GEN1 GEN2 V1 PL1 PL2)
7
8               (COND [(NOT (ATTACHED A1 A2)) (RETURN T)])
9
10              (PRESENT '(LOCATION !,A1 !>LOC1))
11              (PRESENT '(LOCATION !,A2 !>LOC2))
12              (MOVE 'LEFT LOC1) (MOVE 'RIGHT LOC2)
13              (GRASP 'LEFT A1) (GRASP 'RIGHT A2)
14              (COND [(NOT (ATTACHED A1 A2)) (RETURN T)])
15
16              ?  HANDS NOT STRONG ENOUGH, TRY PLIERS
17
18              (CSETQ GEN1 !"((*POSSIBILITIES) *IGNORE
19                        (*GENERATOR (NEXT-OBJ 'PLIERS '(BIGENOUGH $ A1)))))
20      :PLOOP1
21              (CSETQ PL1 (TRY-NEXT GEN1 '(GO 'TRY-VISE)))
22              (CSETQ GEN2 !"((*POSSIBILITIES) *IGNORE
23                             (*GENERATOR (NEXT-OBJ 'PLIERS
24                                                   '(AND (NOT (EQ PL1 $))
25                                                         (BIGENOUGH $ A2))))))
26      :PLOOP2
27              (CSETQ PL2 (TRY-NEXT GEN2 '(GO 'PLOOP1)))
28              (COND [(ATTEMPT PL1 PL2 A1 A2) (RETURN T)]
29                    [T (GO 'PLOOP2)])
30
31              ?  NO PLIERS LARGE ENOUGH, OR PLIERS NOT STRONG
32              ?  ENOUGH.  TRY A VISE ON ONE SIDE.
33
34      :TRY-VISE
35              (CSETQ GEN1 !"((*POSSIBILITIES) *IGNORE
36                           (*GENERATOR (NEXT-OBJ 'VISE '(BIGENOUGH $ A1)))))
37      :VLOOP
38              (CSETQ V1 (TRY-NEXT GEN1 '(GO 'NO-CAN-DO)))
39              (CSETQ GEN2 !"((*POSSIBILITIES) *IGNORE
40                        (*GENERATOR (NEXT-OBJ 'PLIERS '(BIGENOUGH $ A2)))))
41      :PLOOP3
42              (CSETQ PL1 (TRY-NEXT GEN2 '(GO 'VLOOP)))
43              (COND [(ATTEMPT V1 PL1 A1 A2) (RETURN T)]
44                    [T (GO 'PLOOP3)])
45
46              ?  ALL ATTEMPTS FAILED
47
48      :NO-CAN-DO
49              (PRIN1 "CAN'T UNSCREW ") (PRIN1 A1)
50              (PRIN1 " & ") (PRIN1 A2) (TERPRI)
51              (RETURN NIL)
52      )
53
54
55
56
57      (CDEFUN BIGENOUGH (HOLDER HOLDEE)
58
59        ?  RETURN T IFF OBJECT HOLDER IS LARGE
60        ?  ENOUGH TO HOLD OBJECT HOLDEE
61
```

```
                "AUX" (C S)

    (PRESENT '(CAPACITY !,HOLDER !>C))
    (PRESENT '(SIZE !,HOLDEE !>S))
    (NOT (LESSP C S))
    )



    (CDEFUN NEXT-OBJ (TYPE PRED)

      ?  GENERATOR TO RETURN NEXT OBJECT OF 'TYPE'
      ?  WHICH SATISFIES 'PRED'

                "AUX" (OBJ TEMP)

       (CSETQ TEMP (FETCH '(ISA !>OBJ !,TYPE)))
    :LOOP
       (TRY-NEXT TEMP '(ADIEU))
       (COND [(CVAL (SUBST OBJ '$ PRED))
              (NOTE OBJ)
              (AU-REVOIR)])
       (GO 'LOOP)
    )
```

5.5.2. Commentary

2. CDEFUN defines a function to CONNIVER.

6. "AUX" <list> defines local variables.

10. PRESENT is a CONNIVER function which searches the data base for an item which matches its pattern argument. If one is found, PRESENT sets the indicated variables (marked with !< or !>) and returns the item. !,A1 indicates the current CONNIVER value of A1. !>LOC1 indicates that LOC1 is to be bound if possible.

18. GEN1 is being assigned a TRY-NEXT possibilities list. !" tells CONNIVER to do a "skeleton expansion" of the following list (which is necessary to CONNIVER's internals). The (*POSSIBILITIES) and *IGNORE are syntatic markers to TRY-NEXT whose function we can ignore. (*GENERATOR <func-call>) indicates to TRY-NEXT to use <func-call> to generate additional possibilities if needed.

19. NEXT-OBJ will continue to generate objects of type PLIERS which satisfy the predicate (2nd argument). It will generate one PLIERS at a time. (BIGENOUGH $ A1) is a skeleton predicate which NEXT-OBJ will use to screen each possibility. The current candidate is substituted for $ before the predicate is CVALuated (CONNIVER's form of EVALuation).

21. When GEN1 contains no more possibilities, TRY-NEXT will execute (GO 'TRY-VISE). Unlike LISP, GO evaluates its argument here.

24. Check to insure that two distinct pairs of pliers will be found.

64. See 10.

66. RETURN is not necessary since the value of a CONNIVER function is the last expression evaluated.

72. This starts the definition of the generator. Note that NEXT-OBJ looks like a regular function to CONNIVER until it is called.

79. FETCH is a CONNIVER primitive which returns a possibilities list of all items in the data base which match its pattern argument. !>OBJ indicates that OBJ should be bound by TRY-NEXT to each possibility in turn.

81. TRY-NEXT binds OBJ from the possibilities list TEMP and removes the current possibility. If there is no current possibility, (ADIEU) is evaluated which causes termination of the generator.

82. The desired predicate is CVALuated after substituting the current object into the skeleton. (SUBST A B C) is a LISP function which returns a list which is the result of substituting A for every occurrence of B in list C.

83. (NOTE OBJ) is a CONNIVER function which places the current value of OBJ onto the current possibilities list.

84. (AU-REVOIR) returns control from NEXT-OBJ but leaves the generator in a suspended state. When TRY-NEXT returns control to NEXT-OBJ, execution will resume at (GO 'LOOP).

6. Recommendations

In principle, either SAIL or LISP could provide an excellent basis for real-time planning and execution control of a large automated shop. However, each language possesses features which facilitate certain types of operations. In particular, SAIL is generically better at the low level control of I/O devices, and has more extensive abilities for interacting with the operating system (especially where file manipulations are concerned). LISP, on the other hand, is more flexible at the higher planning levels and where system development and debugging are concerned.

We would therefore envision an "ideal" system as one which merged all the desirable features of these two language classes. The merger would probably adopt LISP program and data structure format, augmented where necessary to accomodate SAIL file-related operations, and possibly LEAP. SAIL features would be implanted in this environment, and, at the implementor's discretion, an ALGOL-like front-end syntax (such as MLISP's) could be grafted onto the front of the system to make LISP more tractable to humans.

Our recommendation therefore is that some development effort be invested in the merger of LISP and SAIL. Such a merger would provide the foundations for more complex control as found in MP and CONNIVER, without having to cope with other, more cumbersome parts of these systems.

Such a merger should take care to preserve the following desirable features of SAIL and LISP:

(1) Data structures should accomodate complex symbolic information as well as string and numeric information. As in LISP, data structures should be free to grow in unrestricted ways, and storage declarations should be optional to the user.

(2) Program and data should, as in LISP, be in one and the same format. Such a representation underlies (a) a strong macro facility, (b) rapid editing, modification and debugging of programs, and (c) self-modifying and self-extending systems. The last ability would, for example, enable the system, given the description of a new type of tool, automatically to synthesize the programs for controlling the tool from a library of sub-functions.

(3) There should be strong I/O and file manipulating facilities, as are found in SAIL. A good random-access file system is imperative for even moderately large databases. The system should have both high and low level control over input and output formatting which provides control down to the bit level of the machine.

(4) There should be a highly-developed interrupt subsystem. With the merger of SAIL's bit-wise interrupt control, and LISP's symbolic abilities, such a system as is described in [Rieger 76] could be efficiently implemented, serving as the network protocol for a large collection of highly autonomous processes where the synthesis and control of many parallel events is important.

(5) For software development and debugging, the language should be interpreted. Yet, the language should be compilable for production usage. LISP currently satisfies these requirements.

(6) The system should provide for a large, context-sensitive, associative database. This would involve some new engineering to coordinate a MP-like database with an efficient random-access file system. [McDermott75] surveys some ideas on this topic.

(7) There should be some degree of automatic problem-solving control which includes a CONNIVER-like context-switching and process-suspending mechanism. There should also be accomodations for SAIL-like parallel process control, and emphasis should be placed on inter-process communications protocols. Most of the ideas already exist in CONNIVER and SAIL, but the ideas need to be synthesized into a unified system.

Such a merger could be carried out either at the concept level (in which case, NBS would supervise the development of a new language which implements these features), or by modifying and extending a good existing LISP system. We would consider the second alternative more feasible and appropriate.

7. Bibliography

[Baumgart72] Baumgart, B. G. "Micro-Planner Alternate Reference Manual," Stanford AI Lab Operating Note No. 67, April 1972.

[BBNEXEC] Bolt, Beranek and Newman. "TENEX Executive Manual," Cambridge, Massachusetts, April 1973.

[Beech70] Beech, D. "A Structured View of PL/1," ACM Computing Surveys, March 1970, pp. 33-64.

[Bobrow74] Bobrow, D. G. and Raphael, B. "New Programming Languages for Artificial Intelligence," ACM Computing Surveys, September 1974, pp. 153-174.

[Burstall71] Burstall, R. M., Collins, J. S. and Popplestone, R. J. Programming in POP-2, The Round Table and Edinburgh University Press, 1971.

[Church41] Church, A. The Calculi of Lambda Conversion, Princeton University Press, Princeton, New Jersey, 1941.

[COBOL74] COBOL. "American National Standard Programming Language COBOL," X3.32 - 1974, American National Standards Institute, Inc., New York, 1974.

[CODASYL71] CODASYL Data Base Task Group. "April 1971 Report," ACM, New York, 1971.

[DEC] DEC. "DEC System-10 Data Base Management System Programmer's Procedures Manual," Document DEC-10-APPMA-B-D, Maynard, Massachusetts.

[Dijkstra75] Dijkstra, E.W.D., Lamport, L., Martin, A.J., Scholten, C.S., Steffens, E.F.M. "On-the-fly Garbage Collection: An Exercise in Cooperation," Burroughs, Plataanstraat 5, NL-4565 NUENEN, The Netherlands, EWD496-0.

[Fahlman73] Fahlman, S. "A Planning System for Robot Construction Tasks," MIT AI Memo 283, 1973.

[Feldman69] Feldman, J. A. and Rovner, P. D. "An ALGOL-Based Associative Language," Communications of the ACM, August 1969, pp. 439-449.

[Feldman71] Feldman, J. A. and Sproull, R. F. "System Support for the Stanford Hand-Eye System," Second International Joint Conference on Artificial Intelligence, London, September 1-3, 1971.

[Finkel74] Finkel, R., Taylor, R., Bolles, R., Paul, R. and Feldman, J. "AL, A Programming System for Automation," Stanford Artificial Intelligence Laboratory, Memo AIM-243, November 1974.

[Hewitt69] Hewitt, C., "PLANNER: A Language for Proving Theorems in Robots," Proc. IJCAI-1, 1969

[Leslie72] Leslie, W.H.P. (editor). Numerical Control Programming Languages, North-Holland Publishing Company, London, 1972.

[Levin65] Levin, M.I. "LISP 1.5 Programmer's Manual," The M.I.T. Press, Cambridge, Massachusetts, 1965.

[McCarthy60] McCarthy, J. "Recursive Functions of Symbolic Expressions and their Computation by Machine," Communications of the ACM, April 1960, pp. 184-195.

[McDermott72] McDermott, D. V. and Sussman, G. J. "The Conniver Reference Manual," AI Memo No. 259, MIT Project MAC, May 1972.

[Moon74] Moon, D.A. "MACLISP Reference Manual," Project MAC - Massachusetts Institute of Technology, Cambridge, Massachusetts, 1974.

[Naur60] Naur, P. (Editor). "Revised Report on the Algorithmic Language ALGOL 60," Communications of the ACM, May 1960, pp. 299-314.

[Norman69] Norman, E. "LISP," University of Wisconsin Computing Center, Madison, Wisconsin, April 1969.

[Parsons74] Parsons, F. G., Dale, A. G. and Yurkanan, C. V. "Data Manipulation Language Requirements for Database Management Systems," Computer Journal, May 1974, pp. 99-103.

[RAPIDATA] RAPIDATA Corporation. "A FORTRAN DML Implementation for DBMS-10," Fairfield, New Jersey.

[Rieger76] Rieger, C.J. "Spontaneous Computation in Cognitive Models," Department of Computer Science, University of Maryland, TR-459, July 1976.

[Reiser75] Reiser, J. F. "BAIL--A debugger for SAIL," Stanford Artificial Intelligence Laboratory, Memo AIM-270, October 1975.

[Reiser76] Reiser, J. F. (Editor). "SAIL," Stanford Artificial Intelligence Laboratory, Memo AIM-289, August 1976.

[Samet76] Samet, H. "The SAIL Data Base Management System," Computer Science Department, University of Maryland, College Park, Maryland, Unpublished, 1976.

[Siklossy76] Siklossy, L. Let's Talk LISP, Prentice-Hall, Inc., 1976.

[Smith70] Smith, D. C. "MLISP," Stanford Artificial Intelligence Project, Memo AIM-135, 1970.

[Stacey74] Stacey, G. M. "A FORTRAN Interface to the CODASYL Database Task Group Specifications," Computer Journal, May 1974, pp. 124-129.

[Sussman72] Sussman, G., Winograd, T., and Charniak, E., "MICROPLANNER Reference Manual," M.I.T. AI-TR-203a, 1971

[Taylor76] Taylor, R. W. and Frank, R. L. "CODASYL Data-Base Management Systems," ACM Computing Surveys, March 1976, pp. 67-103.

[Teitelman74] Teitelman, W. "INTERLISP Reference Manual," XEROX Palo Alto Research Center, Palo Alto, California, 1974.

[TOPS10] DEC. "DECSYSTEM-10 Operating Systems Command Manual," DEC-10-OSCMA-A-D, Digital Equipment Corporation, Maynard, Massachusetts, May 1974

[Weissman67] Weissman, C. "LISP 1.5 Primer," Dickinson Publishing Company, 1967.

[Wilcox76] Wilcox, C. R. "MAINSAIL Language Manual," SUMEX, Stanford University, May 1976.

8. Summary Chart

	Flow of Control	Data Types	Declarations	Scope	I/O Facilities	Development Aids	Linkage to Other Processors	General Comments
LISP	sequential, labels	s-expressions (atoms of type real, arbitrary precision integer, string, symbol, array)	implicit for globals, explicit for locals, typeless	dynamic	frequently meager file interaction, READMACRO	structure editor, run-time macros, breakpoints	possible (done at interpreter level)	extremely consistent, flexible, general purpose, usually interpreted (can be compiled for efficiency), program and dat are indistinguishable
PLANNER	pattern-directed, sequential, labels, backtracking	s-expressions (same as LISP)	typeless	same as LISP	paltry	similar to LISP including monitoring of GOALS and data base activity	same as LISP	written in LISP, non-procedural, special purpose, interpreted, data base contains arbitrary n-tuples
CONNIVER	pattern-directed, sequential, suspended processes, labels, frames	s-expressions (same as LISP)	typeless	dynamic, frame	paltry	similar to LISP including a frame debugger	same as LISP	written in LISP, non-procedural, complex control and data structures, special purpose, interpreted
SAIL	processes, coroutines, sequential, FOREACH, iteration, backtracking, labels	integer, real, string, array, structure, pointer, list, set, itemvar, context, procedure, item	statically typed declarations except for run-time items	static, block-structured	excellent - from low level to high level	RAID, BAIL, macros, conditional compilation, REQUIRE feature	easy, can use libraries, in-line assembly language statements	compiled, general purpose, extensive run-time library, associative data base consisting of triples, procedural, backtracking, network data base interface, **
POP-2	sequential, labels, iteration	real, integer, string, symbol, list, record, array, pair	explicit, typeless	dynamic	similar to LISP but has file I/O	similar to LISP	same as LISP	interpreted, not as consistent as LISP, general purpose, stack based, capability of returning more than one result (without using a list as in LISP), data and program are almost identical
MLISP	sequential, labels, iteration	s-expressions (same as LISP)	typeless	dynamic	same as LISP	same as LISP but user must debug in LISP and not in MLISP	same as LISP	translated into LISP, ALGOL-like syntax, readable

** compile-time facility control of extenranl devices, interprogram communication

APPENDIX D

A SURVEY OF SIMULATION LANGUAGES

Prepared for National Bureau of Standards

by

Mr. Benjamin Clymer, P.E.
Consulting Analytical Engineer

Introduction

The purpose of this Appendix is to bring together the ideas in this report which relate to simulation languages and to present those ideas in tutorial form as constituting an important cross-section of the subject of integrated computer aided manufacturing.

For the benefit of those readers who are not familiar with the subject at hand, a few basic definitions are offered at the outset of the discussion:

(1) "Software" is the hierarchy of programming tools (including languages) by which a user of a computer exerts command control over the hardware. (A longer but but better definition is given in (108, pp 177-178))*.

(2) "Problem-oriented languages" are among the higher-echelon languages; they employ a vocabulary appropriate to the application problem.

(3) A "simulation language" is a problem-oriented language intended for application to problems involving simulation. A simulation language is built upon a more general-purpose language, such as FORTRAN. A simulation language saves the user's time in writing and debugging a computer program, but the program takes longer to be executed (run) than would a program written in a lower-level language.

(4) "Simulation" is the "establishment of a mathematical-logical model of a system and the experimental manipulation of it on a ... computer" (4). Simulation produces a "dynamic portrayal of the states of a system over time" (4).

(5) Some of the instances of a "system" which are relevant here are an aircraft factory, an engineering department, an aircraft, or any subset of one of these.

(6) A "model" is a set of equations and/or verbal statements which together define selected states and behaviors of the corresponding system, and which are capable of being rendered as a simulation program that puts the model into specified situations.

(7) A "discrete" system, model, or language, is one which is composed of or deals with discrete objects and discrete events affecting them. The events usually occur at irregular times.

(8) A "continuous" system, model, or language is one which undergoes or expresses continuous processes, usually without sudden discontinuities.

(9) A "combined" system, model, or language has some discrete and some continuous content.

There is a regrettably sharp dichotomy of simulation people into discrete and continuous camps (1). Few persons can deal with both, so the "combined" camp is still small. The discrete camp contains people from probability and statistics, industrial engineering, operations research, and management science, whereas the continuous camp is populated from differential equations, analog computation, physical sciences, and recently the soft sciences.

There are many simulation languages. In order to focus this Appendix upon the language which are most relevant, emphasis has been placed upon the following: GPSS, SIMSCRIPT, CSSL, CSMP, DYNAMO, and GASP IV. Less concern has been given to languages developed and used mainly outside this country (such as SIMULA), or implemented only on old or unusual or small computers (such as PACTOLUS), or languages which are themselves old and to various degrees superseded (such as MIDAS and DAS).

---

*Numbers in parentheses are numbers of references in Section 4 of this Appendix.

The level of the discussion herein is somewhat abstract. Details of the design of particular simulation languages are not brought in at all. Comparisons of languages are made only by citing opinions in the literature, there being more than enough of these gratuitous opinions to serve the purpose here.

1. Tutorial Survey of Simulation Languages

1.1 GPSS

1.1.1 Overview of Language

GPSS is a highly structured discrete simulation language. It is designed to give useful results easily and quickly (5). The user does not need to know much programming; in fact, a statistical report is produced automatically (3).

GPSS is very well documented (3, 5, 6, 52-54).

The user of GPSS/360 invokes up to 43 "blocks", each of which is a subprogram in assembly language specifying an action by the computer. Examples of GPSS blocks are link, assign, tabulate, terminate and queue. When a model has been programmed, the blocks are assembled into operational instructions to the computer by the software (3,6).

GPSS is the most widely used discrete simulation language (6). Its flow orientation makes it "an effective and efficient tool" (112) for queueing problems (6).

1.1.2 Applications

The literature concerning applications of discrete simulation languages is severely truncated by proprietary interest on the part of the organizations which have developed most of the practical applications. Therefore, the published papers have tended to be of academic orientation: too theoretical, and too out of touch with real-life applications (1).

Table 1 presents a sampling of representative papers concerning applications of GPSS to systems of concern herein. All of the applications in Table 1 are directly or by analogy relevant to CAD/CAM in the aircraft industry. The analogies are self-evident, and several of them are noted in the literature. Many application classes are omitted for lack of relevance, as judged by the writer.

1.1.3 Standardization Status

Some opportunity for standardization is available to users, and initiative has been shown in this direction. For example, a steel company has developed a "standardized" GPSS macro which simulates overhead crane movements within simulations of larger portions of a steel mill. "The primary purpose of the routine is to eliminate the need to rewrite simulations of crane movements, thus providing a substantial reduction in model development time" (12).

1.1.4 Evolution and Derivatives

GPSS was first released by IBM in 1961. As of 1971 the evolution had led to GPSS V, then the latest version (5). GPSS/360, version 1, is upward-compatible with GPSS V (87).

Some typical extensions to GPSS, which have been developed by users, may be noted:

(1) To avoid what had been a relatively large effort, an extension was proposed (35) to deal efficiently with parameters dependent upon simulation time.

(2) GPSS with added graphics, denoted by GAPSS, was developed and applied to shop scheduling problems (44).

(3) An extension of GPSS/360, version 5, called APPLES, was designed to be used by production engineers who were not skilled in GPSS. The applications were to production lines for electronic and electromechanical aerospace products. Standard data forms were used to define a problem (20).

(4) Norden Division of United Aircraft Corp. has described some of its extensions to GPSS: dynamic on-line display, data bases for input, interactivity with user, and a combined language gotten by coupling GPSS to CSMP via GPSS HELP blocks and by putting GPSS in control (1).

1.1.5 Implementation Status

GPSS, having been issued by IBM originally, has of course been implemented on system 360 and subsequent computers. The IBM versions of GPSS are implemented in assembly language, whereas the Univac versions are in FORTRAN. Most major computers in this country have a version of GPSS (5).

1.2 SIMSCRIPT

1.2.1 Overview of Language

SIMSCRIPT was developed by Rand Corp. and was released in 1962. Meanwhile, SIMSCRIPT has become the most widely used discrete simulation language after GPSS (5). It provides the power and freedom of a general-purpose programming language (18), but the user has to have some programming expertise; he must specify the statistics he wants, and it takes effort to document a model (5). SIMSCRIPT is modular, which facilitates debugging (6), even in the large models for which it is designed (5). SIMSCRIPT is parsimonious in its memory requirements, and it is relatively inexpensive to run (3).

SIMSCRIPT is well documented (3, 5, 6, 18, 50, 51, 99).

SIMSCRIPT programs consist of a vector of the attributes of entities and a collection of event generation and interaction routines (99).

A SIMSCRIPT program is translated into FORTRAN, whence it is compiled and assembled. SIMSCRIPT I.5 eliminates the step of translation (6).

1.2.2 Applications

Only one relevant paper concerning an application of SIMSCRIPT to manufacturing or analogous systems was encountered in the writer's search. This statistic is in contrast to the number of entries in Table 1 for GPSS. The application found concerned a "flexible manufacturing system" (FMS), which consists of automated transfer lines and numerical control (NC) machines (98). Large cost savings are claimed compared with conventional job shops.

1.2.3 Standardization Status

No information on this subject was encountered in the search.

1.2.4 Evolution and Derivatives

Many languages have stemmed from the original SIMSCRIPT:

(1) SIMSCRIPT II, dating from 1968, is radically different. It has its own compiler, which is written in SIMSCRIPT II. It requires programming competence (5). It is said to be the most comprehensive discrete simulation language (6). It is even dignified with the description "higher order programming system" (5).

(2) SIMSCRIPT II Plus is marketed by Simulation Associates Inc. (6).

(3) SIMSCRIPT II.5, which is marketed by Consolidated Analysis Centers Inc., contains additional improvements (6,100).

TABLE 1

Relevant Applications of GPSS

Degree of Relevance	Class of Application	References
Essentially Directly	Material handling system	12
	Machine shop or job shop	48, 32
	Assembly line tree, manufacturing system, or factory	9, 83, 73, 21
	Steel mill	12
	General queuing network	96
More or Less Indirectly, by Analogy to CAD/CAM	Computer, or net or computers or microprocessors	112, 111, 107, 82, 113
	Telephone system	56
	Health care system	60, 27
	Bank real-time system	112
	Airline schedule	77

(4) SIMFOR, which is based on a set of subprograms coded in FORTRAN, provides the basic statements of SIMSCRIPT (105).

(5) SIMPL/1 is a SIMSCRIPT-like extension of PL/1 (95). It is said to be "easy to learn" and "convenient" (96), although it has none of the "amenities" of SIMSCRIPT or GPSS (95).

(6) DESPL/1 (Discrete Event Simulation Programming Language / Version 1) is a close relative of SIMSCRIPT II. It functions as a preprocessor to the PL/1 compiler (55).

(7) CSP (Computer Simulation Program) is said to be a modular, flexible, and efficent tool for all stages of the design of a computer. CSP I was written in SIMSCRIPT I.5 in 1972; then CSP II was written in SIMSCRIPT II.5 in 1975 (115).

(8) Another extension of SIMSCRIPT enables it to control DYNAMO by calling events relating to the differential equation integrations at the end of every time interval. In this version DYNAMO uses trapezoidal integration (18).

1.2.5 Implementation Status

In 1971 SIMSCRIPT compilers were available for the IBM 360, RCA Spectra 70, and many other computers (5).

1.3 CSSL

1.3.1 Overview of Language

CSSL is a family of languages for continuous system simulation, including currently MIMIC, CSMP, CSSL III, and ACSL (92). Some of the versions of CSSL are HYTRAN (for analog-digital hybrid simulation programming), SL-1, and S/360.

CSSL had its origins in languages which sought to make the digital computer less formidable for analog users to learn. These languages have been well surveyed (116).

CSSL is a statement-oriented language for dynamic continuous system simulation (119).

CSSL III is marketed by Programming Sciences Corp. and by Control Data Corp. (92). The difinitive references are (117, 118). CSSL III has seven options for its integration subroutine. It is designed for ease of batch processing by novice programmers. CSSL III programs are translated into FORTRAN for compilation and execution (92).

1.3.2 Applications

In the writer's search no references were found to papers dealing with manufacturing applications or even analogous applications. Rather, numerous applications (not cited herein) turned up for a variety of engineering design problems involving essentially continuous systems.

There is a rather tenuous analogy between an assembly line and a set of chemical reactions. With a sufficiently high rate of production the assembly line could be modeled with the differential equations of chemical kinetics. A piece of scrap would correspond to a chemical byproduct. Similarly, it is possible to model a health care system as a continuous system (28).

1.3.3 Standardization Status

In 1965 Simulation Councils (now the Society for Computer Simulation) established a Simulation Software Committee. It developed specifications for CSSL's which were published in the December 1967 issue of the SCi journal "Simulation". Currently the committee has been reestablished and is exploring possible updates in the specifications. Several versions of CSSL, such as Raytheon's RSSL (102), have been voluntarily made to conform to these specifications.

1.3.4 Evolution and Derivatives

Several derivatives of CSSL have evolved:

(1) GCSSL is an interactive graphics extension of CSSL III (59, 92, 93). The authors would use ACSL as a starting point, if they could do it again.

(2) The originators of Raytheon's RSSL, Mitchell and Gauthier Associates, have more recently developed and have been marketing ACSL ("Advanced Continuous Simulation Language") (58, 91, 102). ACSL has such features as interactive output displays (plots and/or tables), several options for the integration algorithm, a strong analog "flavor", and array capability. A program can be prepared from a block diagram, FORTRAN statements, or a mixture of the two.

(3) DARE P is a batch mode, ANSI FORTRAN IV based variant of CSSL (66).

(4) AHCSSL is intended for analog-digital hybrid computer users (90).

(5) CSSL IV is proprietary with Young Lee and Associates, Torrance, California. It is said to be an improved version of CSSL III, having a number of enhancements and extensions which enable it to run much faster.

1.3.5 Implementation Status

CSSL III has been available on the Control Data 6000 series computers, as well as on the IBM 360, XDS Sigma 7, and Univac 1108 (59, 119).

ACSL is implemented on all major CDC, Univac, and IBM computers.

1.4 CSMP

1.4.1 Overview of Language

CSMP/(Continuous System Modeling Program) was developed by IBM (119, 120). It is based on IBM's DSL/90 digital simulation language. Problems can be prepared from either a block diagram or a set of differential equations. CSMP can also accept most FORTRAN statements.

1.4.2 Applications

Relevant applications papers which were found in the writer's search are presented in Table 2.

1.4.3 Standardization Status

No information on this subject was encountered in the search.

1.4.4 Evolution and Derivatives

Many derivatives have sprung from CSMP:

(1) PRODYC is a language for chemical plant design studies. It is capable of representing the plant's control system, and as of 1971 it could represent six different processes, each with its own subroutine. An executive routine ties the submodels together. PRODYC generates a CSMP program. (34)

(2) A precompiler to convert a chemical kinetics model into a CSMP program is CHEMCSMP (23).

(3) Among interactive versions of CSMP/360 is (85).

(4) Another on-line interactive version of CSMP/360 with graphic output has been reported in (76).

TABLE 2

Relevant Applications of CSMP

Class of Application	References
Mechanical impact	42
Landing gear design	38
Control system design	40, 41
Aerospace component design	36
Inventory control	30
Aerodynamics	17
Optimization (2-point boundary value problem)	19
Adaptive manufacturing system	22

(5) An extension of CSMP offering printer or plotter graphic output is (46).

(6) Other graphics additions to CSMP are described in (43).

(7) A graphic CSMP is presented in (39).

(8) An extension of CSMP, which incorporates some features of MIMIC, is given in (37).

(9) CSMP III has graphics capabilities (31).

1.4.5 Implementation Status

CSMP, being an IBM product, is available on the 360, 370, and 1130. The latter version is only block oriented (119, 121).

1.5 DYNAMO

1.5.1 Overview of Language

DYNAMO is a special-purpose compiler for continuous system simulation. It was developed originally to implement the "Industrial Dynamics" methodology, as Forrester called systems modeling and simulation initially (later "Systems Dynamics"). The definitive references for the first version of DYNAMO are (62, 122).

DYNAMO has error detection and diagnostic capabilities (18).

DYNAMO is "brilliantly limited" (18). For example, it uses only Eulerian (rectangular) integration with fixed step size, even in the DYNAMO II User's Guide version of 1970 (4). The result is a simple language with, of course, limitations.

1.5.2 Applications

DYNAMO has been especially popular for simulation of company, city, or the world, following Forrester. Company simulation, strongly influenced by Industrial Dynamics (122), have tended to concentrate on the aggregated dependent variables which are of concern in upper echelons of management, to the virtual exclusion of particular factories as such (62).

DYNAMO has been applied to the dental care delivery system for this country (101).

1.5.3 Standardization Status

The DYNAMO languages have been developed and marketed only by Pugh Roberts Inc. Hence some future standardization is more possible than it might otherwise be.

1.5.4 Evolution and Derivatives

The latest version if DYNAMO III (101).

1.5.5 Implementation Status

The original DYNAMO compiler was designed for use on the IBM 704. It was quickly extended to the IBM 709 and 7090 (122). More recent versions are available for virtually any computer.

1.6 GASP IV

1.6.6 Overview of Language

GASP IV is a "combined" language intended for simulation of systems having both discrete and continuous aspect. It is preeminent in a new field. Of all combined languages, only GASP IV is well documented and receives vendor support (103).

The definitive references for GASP IV are (4, 8, 7, 63-65).

GASP IV was developed and is marketed by Pritsker and Associates (4). It was built upon GASP II, which is only discrete, by including some continuous elements (64).

GASP IV is written in ANSI FORTRAN IV. Therefore, it is approachable by many people and is potentially widely portable to different computers (4).

Some of the features in GASP IV include statistical calculations, random deviate generation, report generation, and collection of performance data for the modeled system, in addition to exercising the model (4).

GASP IV can use three independent variables: time and two space dimensions. Hence it can model factory equipment in terms of their actual floor locations (7).

Integration in GASP IV is done by a variable-step-size fourth-order Runge-Kutta algorithm (123, 124). Step size is adjusted to make any "time event" (occurring at a preknown time) occur at the end of a step (64).

The structure of GASP IV is a two-echelon hierarchy. The top level is the executive function, which switches among modes (initialization, state variable updating, monitoring, etc.). The lower level is the event selection function, which sequences the execution of event routines. Control passes back and forth between these two echelons (4).

The user has to keep in mind 98 GASP system variables and 41 GASP and user subprogram names. His responsibilities consist of writing the following in FORTRAN:

(1) an initialization routine

(2) a subroutine for each type of discrete event in the model

(3) a subroutine defining state and derivative equations for the continuous state variables

(4) a subroutine defining conditions on state variables when events trigger (69).

1.6.2 Applications

There have been quite a few applications of GASP IV to problems relevant to CAM. Pritsker's book (4) presents five examples, which are mostly discrete. Wortman (7) states that GASP IV can be used for problems involved in automated manufacturing, including inventory strategies, logistics, queueing, scheduling, materials handling, plant design, etc. There has been an application to an inventory problem with backorders (104). A system consisting of four parallel production lines, seven warehouses, and many products with variable demand, has been studied with GASP IV (14).

A computation center has been modeled with SCOPE 4(103). It is written in FORTRAN, and it calls GASP IV subroutines.

The plants comprising a crop have been simulated with GASP IV (109). The model included growth, utilization of various nutrients, light, water, etc. The problem is somewhat analogous to a manufacturing "plant", which likewise produces new growth (products) by employing raw materials.

1.6.3 Standardization Status

There is a GASP IV user's group in ACM SIGSIM (7). Such groups tend to generate extensions and embellishments, but this one might be able to undertake some efforts related to standardization.

1.6.4 Evolution and Derivatives

Although GASP IV is quite new, it has already spawned some descendants. An interactive version has been developed (104). It was implemented initially on a minicomputer (General Automation SPC 16/65).

1.6.5 Implementation Status

GASP IV is or soon will be implemented on all major computers in this country (4).

1.7 Comparative Evaluations

1.7.1 Discrete Languages

1.7.1.1 GPSS and SIMSCRIPT

The popularity of GPSS is due in part to its simple structure, which facilitates learning it. Moreover, it is powerful enough for many applications, even though it is less flexible than competitive languages (99). The other popular discrete languages in this country, in decreasing order, are SIMSCRIPT, DYNAMO, and SIMULA (1, 5).

A point-by-point comparison of GPSS and SIMSCRIPT is offered in Table 3. Clearly each has substantial advantages for its own class of users and applications. However, the greater popularity of GPSS offers the most working model for the effort expended. In the case of simulation of computers GPSS receives at least one vote: "Contrary to the more conventional view of the matter, we believe that GPSS is superior to any of the specialized computer system simulation packages for the most complex computer system simulation applications" (112).

1.7.1.2 SIMULA

Since SIMULA has not been included in Sections 1.1-1.6, a brief overview will be given here as a preliminary to comparing it with other languages.

ALGOL is a subset of and is the base of SIMULA (5,99). SIMULA has an elegance, richness, and flexibility of structure which recommends it to many users. In fact the structure concept, and expressive power of GPSS and SIMSCRIPT are subsumed in SIMULA so that each of these two languages could be rendered in SIMULA (99). Further information is available in the definitive references (49, 67). The latter contains five papers concerning SIMULA itself as well as abstracts of 34 papers in English.

The SIMULA Development Group serves as a standardizing body. It consists of representatives of SIMULA implementations on various computers. SIMULA is said to be available in the form of compilers for computers of nearly every major US manufacturer, with the possible exception of Burroughs.

The most recent version of SIMULA, available since 1972, is SIMULA 67.

The following comparisons of SIMULA with other discrete simulation languages will be noted:

(1) Like SIMSCRIPT, SIMULA gives the user nothing automatically in the way of analysis and reporting of results; he must program to get them (6).

(2) Also like SIMSCRIPT, SIMULA is harder for an engineer to learn and use (1).

(3) In a comparison made with a military simulation problem (45), SIMULA was found to save personnel time and cost, although it increased computer time and cost, relative to use of FORTRAN.

(4) One comparison (29) found SIMULA 67-A to be better than SIMSCRIPT II in both compiling and executing. It was granted, however, that SIMSCRIPT II has a powerful language structure, better readability, and excellent documentation.

TABLE 3

Comparison of GPSS and SIMSCRIPT

Feature	GPSS	SIMSCRIPT
Degree of structure (5)	highly structured	less structured
Ease of learning by nonprogrammer (1, 5, 6)	relatively easy	harder; requires competence
Convenience features (5)	includes many	less convenient
Effort to develop significant simulation (5)	moderate	considerably more
Comprehensiveness (6)	less	most of all languages
Freedom in arithmetic and logic (6)	less afforded	greater freedom
Execution time (6)	longer	moderate
Automatic analysis and reporting (6)	much provided	none

(5) One critic feels that SIMULA does not offer the user as sophisticated a tool as GPSS or SIMSCRIPT II (5).

(6) The scenario approach to modeling is convenient for a SIMULA user, since a scenario model can be transferred directly to SIMULA coding (99). However, this approach is also fundamental to the structure of GPSS programs.

1.7.1.3 GASP II

GASP ("General Activitiy Simulation Program") was orignially developed by Kiviat at US Steel, and from it was developed GASP II at Arizona State University (4). It includes a number of special subroutines in FORTRAN which, when called, perform for the user the fundamental actions of the program, such as timing, set manipulation, random deviate generation, statistical summary, I/O, etc. (3).

GASP II is generally considered to be competitive with GPSS and SIMSCRIPT. One user chose GASP II "because its lower run costs, smaller core requirement, and shorter compilation times more than offset the greater flexibility of GPSS and SIMSCRIPT II" for a munitions manufacturing study (74). Another user chose GASP II for modeling a job shop with up to nine machines. A 62-item bibliography of the application of discrete languages to job shop scheduling simulation is included in this paper (61).

1.7.1.4 Other Discrete Languages

Numerous other discrete languages are documented in the literature as "also rans" (3). For example, SOL, OPS 3, and WASP are said (5) to be less sophisticated than the front-running languages. One user selected SIMPL/1 over GPSS V for his problem because the people concerned already knew PL/1, a fact which obviated a psychological obstacle (96). The same argument might have led him to SPL, which also has PL/1 as its base language (5). Some of the other lesser known languages are CSL (72), Application Program (81), ESP (5), and ECSL (78). A total of 38 languages are tabulated in (3).

1.7.1.5 FORTRAN

It is generally considered that simulation languages are preferable to FORTRAN or other general purpose languages, because programming is a smaller and easier job. This simplicity of specifying actions in the program outweighs the disadvantage of longer execution time (6). Reitman (1) asserts that FORTRAN programs take 3 to 20 times as long to write and debug. Nevertheless, FORTRAN IV has been used to simulate a job shop (71).

1.7.1.6 Classifications of Discrete Languages

Kay (3) discerns three families of discrete languages:

GASP family, including FORSIM
GPSS family, including GPS
SIMSCRIPT family, including SIMULA and CSL.

Fishman (6) uses a different three-fold classification:

Event-oriented, including GASP II and SIMSCRIPT
Activity-oriented, including CSL, FORSIM, and GSP
Process-oriented, including GPSS, SIMULA, SPL, and SIMPL

These orientations distinguish ways of organizing the sequencing of events.

1.7.2 Continuous Languages

There have been many comparisons of the continuous simulation languages already mentioned herein with each other and with general purpose analog and hybrid analog-digital hardware (33, 11, 15, 16, 40, 47). Generally, the use of continuous languages on a digital computer has taken most of the market away from analog and hybrid systems,

since less experience is required on the part of the user. However, hybrid computers show great savings in computer time for problems having a large bandwidth (ratio of highest to lowest natural frequency or reciprocal time constant) (108).

There are not great differences among continuous languages of the same software generation. Such differences as there are are shown in Table 4.

A competitor to DYNAMO is NUCLEUS (2, 125). It too uses Eluerian integration. It has been applied widely to problems in engineering planning. The continuous simulation part of NUCLEUS is machine-independent; the other portions have been implemented on the CDC 6000 series and CYBER 73 computers. NUCLEUS is marketed by Battelle Columbus Laboratories. It has not been tested on the same problem against other languages.

1.7.3 Combined Languages

The combined languages which predated GASP IV have been surveyed by Oren (57). They include SSL, CLASS, DYSYS, GEST (86), and GSL (84). CLASS includes features selected from SIMSCRIPT II, GPSS, and CSSL. GSL ("Generalized Simulation Language") is a FORTRAN-based cross between CSSL and SIMULA which was developed at Case Western Reserve University, drawing upon the earlier work of Fahrland.

GASP IV seems to have seized the lead since its introduction in 1973.

2. Needs for Development

2.1 Language Development Needs

A number of needs for language development can be identified:

(1) There is a lack of language protability between different computers (105).

(2) There is excessive difficulty of debugging a program, and it is virtually impossible to completely debug a language (5).

(3) Programs have low efficiency in compile or execute (105).

(4) Many languages need improvement in the interaction between man and computer. "The role of the computer must change from that of a hostile assistant attempting and succeeding in showing the human to be a relatively inept provider of instructions. Instead the computer must become a patient teacher" (5).

(5) There is need for languages to become more deserving of the name "program generating system" by employment of increasing automation of programming (79, 80).

2.2 Standards Development Needs

Reitman asserts flatly that "Standardization of simulation languages is premature" (5). He does, however, push for machine independence, without which the Air Force would find it impossible to implement CAD/CAM.

One potent force operating to thwart standardization is the pressure favoring proliferation of simulation languages. Sammet (88) points out that "As long as people find it fun to develop languages, as long as they want something which is specifically tailored exactly to their needs, and as long as they are going to find picayune faults with the existing languages, there is very little that technical progress can do to reduce the number of languages". The same conclusion is stated independently by Thompson and Dostert (89).

On the other hand, there is a force tending to throttle higher level language development. According to Reitman (1), "The need for software standardization and the

TABLE 4

Relative Merits of Selected Simulation Languages

	ACSL	CSSL III	CSMP/360	MIMIC
Object Language	Fortran	Fortran	Fortran	Assembly
Macro Processor	Extensive	Extensive	Usual	No
Free Form Coding	Unrestrict.	Unrestrict.	Unrestrict.	No
Run-time Options	Extensive	Average	Average	Nominal
Stiff Integration Alg.	Yes	No	Yes	No
Vector Integration	Yes	No	Yes	No
Symbolic Input	All names	All names	Some	None
Run-time Procedures	Yes	No	No	No
Translation Speed	Fast	Slow	Average	Average
Dynamic Storage	Yes	No	No	No
Full Syntax Analysis	Yes	No	No	Yes
Fully buffered I/O	Yes	No	Yes	Yes
Time-sharing use	Yes	No	No	No
Program size	Average	Large	Average	Small

establishment of elaborate groups organized to support the computer itself have all worked against the use of simulation languages, in fact, all languages except FORTRAN or COBOL. We must avoid the trap of standardization into a rigid set of languages unsuited to problems with certain characteristics. Simulation languages require, if not establishment support, then at least no establishment hindrance."

One benefit of standardization is that the approved languages would be more exercised and thus would have fewer undiscovered bugs.

2.3 Application Development Needs

In recent years there has been growth in the development of corporate models (70). However, these models contain little or no detail in their treatment of production activities. This is as it should be, since a model should not be too many echelons of detail deep. Rather, what is needed is a hierarchy of corporate models. The lower echelons of models in this hierarchy should be the CAD/CAM software.

The state of the art for accuracy of workload simulation is an error or prediction of the order of 5 to 25% (110, 114). There is need for improvement in realism of prediction. To be sure there is a point of diminishing returns of effort to improve accuracy.

As factories evolve, there will be an increasing amount of on-line real-time computing hardware and software on the lower echelons of control. These controls will need to be simulated for purposes of design of these and higher echelons of computer control.

2.4 Hardware Development Needs

While it is not necessary to develop new hardware for CAD/CAM, the possibility deserves to be considered, in order to make the most of simulation languages.

A net of conventional computers is already a reality in most aircraft companies. Simulation languages should be designed to operate in the environment afforded by a network of computers.

Another hardware development which seems inevitable is the interconnection of a computing system, complete with graphic terminals, to a plant-wide data communication system, in order that real-time supervision can be accomplished, at first semi-automatically, and perhaps later fully automatically in a growing sector of responsibility for the computer. Simulation languages, operating faster than real-time, are a valuable tool in such a situation.

Another possibility to be on the alert for is a new type of parallel hardware, which would be eminently suited to the CAD/CAM job. One emerging candidate is HEP (106), which offers a hierarchy of synchronized parallel processors. The hierarchy could represent all or a major portion of a factory. Prototype hardware yields an expectation of an average instruction time of 10 nanosecs.

3. Recommendations

3.1 Applications of Simulation in CAM

The most promising strategy for organizing the total CAD/CAM task is to apply the hierarchical structure principle deliberately wherever feasible.

The computers themselves, by means of which the automated factory is to be scheduled and controlled, could well be organized with an operating system designed to manage a hierarchy of computers. Moreover, that hierarchy of computers would need to be simulated for purposes of system design.

In addition, a hierarchy of models of the factory would be desirable, since no single model could have the depth and bandwidth to deal with the entire factory at once in detail. Conceivably the models in lower echelons would be rendered in logic statements or block diagrams, suited to discrete languages, while some, at least, of the higher echelon models could be continuous (sets of differential equations). There might be opportunity in the top and middle of the hierarchy to apply "combined" modeling and programming. Thus to the hierarchy of models there could correspond a hierarchy of simulation language programs and a hierarchy, somewhat less elaborate, of simulation languages. Specific languages can be chosen for each level as appropriate. The principles for such hierarchical organization of very large and complex systems have been described (24-26).

Each language and program should have an internal hierarchical organization too. Most modern simulation languages do have a hierarchical structure consisting of three echelons. The top level is for simulation timing and control. The bottom level is to call library subroutines. The middle echelon executes simulation oriented routines. (6)

3.2 Language Standardization

It seems desirable to try to exert a minimum of unnecessary constraints against future ingenuity, giving a possible role and place to any deserving future development. To implement this objective would require the broadest and most flexible of structures in any standardization concept. Hierarchical organization of unspecified entities would meet this need. Different companies could use different hierarchies, according to considerations applicable to their individual situations. Hence language standardization is not recommended. Project standards, particularly on systems structure and documentation, will be needed and will be adequately for the Air Force program.

3.3 Language Choice

The success of the Air Force ICAM program will not depend upon which specific language or languages are chosen as long as the system structure is adequately specified and documented. Discrete, continuous, and combined languages are discussed above that are supported by all major hardware vendors, so machine independence is not a major problem.

4. References

(1) Reitman, J., "The Engineering Role of Simulation", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 6, No. 3, March 1974, pp 208-213.

(2) George Juras, Personal Communication, Batelle Columbus Laboratories, 19 Nov. 76.

(3) Key, I. M., "An Over-the-Shoulder Look at Discrete Simulation Languages", Proc. 1972 Spring Joint Computer Conf., Vol. 40, pp 791-798, May 16-18, 1972, Atlantic City, N.J., AFIPS Press, Montvale, N.J.

(4) Pritsker, A.A.B., "The GASP IV Simulation Language", Wiley-Interscience, New York 1974.

(5) Reitman, J., "Computer Simulation Applications: Discrete-Event Simulation for Synthesis and Analysis of Complex Systems", Wiley-Interscience, New York, 1971.

(6) Fishman, "Concepts and Methods in Discrete Event Digital Simulation", John Wiley & Sons, N.Y., 1973.

(7) Wortman, D. B., "Simulation with GASP IV: A Combined Discrete - Continuous Simulation Language", 9th Annual Sim'n Symp., Record of Proceedings, Tampa, Fla., March 17-19, 1976, p 47-59.

(8) Pritsker, A.A.B., "The GASP IV User's Manual", Pritsker & Associates, Inc., Lafayette, Ind., 1973.

(9) Eldin, H.K., and Schroer, B.J., "A Utility Program for Assembly-Line Design", Simulation Councils Proceedings Series, Vol. 3, No. 1, June 1973, pp 5-11.

(10) Moore and Clayton, "GERT Modeling and Simulation: Fundamentals and Applications", Petrocelli/Charter, New York 1976.

(11) Rossnagel, B.L., "A Comparison of 1130 CSMP and the EA1 680 Hybrid Computer for Various Engineering Problems", Darcom Intern Training Center, Texarkana, Texas, April 1976, 82 pp, Report No. Darcom ITC-02-08-76-014.

(12) Hawes, B.W., Jr., "A GPSS Subroutine for the Simulation of Overhead Crane Movements", 9th Annual Sim'n Symp, Tampa, Fla., Mar 17-19, 1976, Record of Proceedings, pp 205-212.

(13) Kay, I. M., Kisko, M., and Van Houweling, D.E., "GPSS/SIMSCRIPT $EM DASH$ The Dominant Simulation Languages", Record of Proc. of 8th Annual Sim'n Symp, Tampa, Fla., Mar 12-14, 1975, p 141-154.

(14) Duncavage, T. D., Reklaitis, G. V., and Woods, J. M., "GASP IV Simulation of Inventory Control and Production Scheduling for a Multi-Product Plant", Proc. 5th Annual Pittsburgh Conf. on Modeling and Simulation, University of Pittsburgh, Pittsburgh, PA, April 24-26, 1974, Part II, pp 1170-1176, Publ, by ISA, Pittsburgh, PA, 1974.

(15) Capehart, B.L., "Dynamic, CSMP, and State Variables, A Comparative Study for Simulation of Dynamic Operations Research Models", Proc. SCSC, Vol. 1, July 19-21, 1971, p 31ff.

(16) Nilsen, R.N., and Karplus, W.J., "Continuous-System Simulation Languages: A State-of-the-Art Survey", Annal. Ass'n. Internationale Calcul Analogique, Vol. 16, No.1, Jan. 1974, p 17-24.

(17) Kumar, A., Cheng, S.C., and Birta, L.G. "Use of the S/360 CSMP Package in Solving an External Boundary-Layer Flow Problem" Simulation, Vol. 20, No. 5, p 163-167, May 1973.

(18) Jones, R.D., "Combining Discrete and Continuous Simulation Dynamo into Simscript", Proc. Annual Winter Sim'n. Conf., Washington, D.C., Jan. 14-16, 1974, Vol. 2, p 747-748.

(19) Storm, J.E., "CSMP III in Optimum Systems Control", Proc. 1973 Summer Computer Simulation Conference, p 429-435, July 17-19, 1973, Montreal, Canada.

(20) Moore, P.J., and Hoggatt, A.C., "Autonetics Planned Production Line Evaluation Simulator (APPLES)", Proc. 1973 Winter Simulation Conf., p 892, Jan 17-19, 1973, San Francisco, California, publ, by AFIPS Press. Abstract only.

(21) Patel, M.M., Panchal, J.M., Coughlin, M.T., "Cycle-Time Simulation for a Multiproduct Manufacturing Facility", Proc. 1973 Winter Simulation Conf., Jan 17-19, San Francisco, California, AFIPS Press, Montvale, N.J., 1973.

(22) Anastasiu, S., and Soceanu, A., "An Application of CSMP Language to Adaptive Systems", Autom. and Electron. (Rumania), Vol. 17, No. 2, Mar-Apr 1973, pp 97-102, in Italian.

(23) Clark, R.L., and Groner, G.F., "A CSMP/360 Precompiler for Kinetic Chemical Equations", Simulation, Vol. 19, No. 4, Oct. 1972, pp 127-132.

(24) Clymer, A.B., "The Modeling and Simulation of Big Systems", Proc. Simulation and Modeling Conf., Pittsburgh, PA, April 21-22, 1969, pp 107-118.

(25) Clymer, A.B., "The Modeling and Simulation of Hierarchical Continuous Systems", Keynote Address, Proc. Conf. on Applications of Continuous System Simulation Languages (1st SCSC), June 30 and July 1, 1969, San Francisco, California pp 1-16.

(26) Clymer, A.B., and Bledsoe, L.J., "A Guide to the Mathematical Modeling of an Ecosystem", in "Simulation and Analysis of Dynamics of a Semi-Desert Grassland", Wright, R.G., and Van Dyne, G.M., Editory, Range Science Dept., Science Series No. 6, Colorado State University, Fort Collins, Colorado, Dec. 1970, p I-75 to I-99, inc.

(27) Caltagirone, R., and Lev, B., "Evaluation of Health Care Delivery by Computer Simulation", Bull, Oper. Res. Soc. of America, Vol. 20, supp. 2, B/399, 1972.

(28) Clymer, A.B., "Next-Generation Models in Ecology", in Pattern, Ed., "Systems Analysis and Simulation in Ecology", Vol. 2, Academic Press, New York, 1972, pp 533-569.

(29) Tognetti, K.P., and Brett, C., "Simscript II and Simula 67-A Comparison", Australian Comput. Journal, Vol. 4, No. 2, May 1972, pp 50-57.

(30) Valisalo, P.E., Sivazlian, B.D., and Maillot, J.F., "Experimental Results on a New Computer Method for Generating Optimal Policy Variables in (S, S) Inventory Control Problem", Proc. 1972 Spring Joint Computer Conf., Vol. 40, pp 199-203, May 16-18, 1972, Atlantic City, N.J., publ. by AFIPS Press, Montvale, NJ.

(31) Caskie, R.E.M., and Mason, R.E.A., "Some Factors Influencing Development of CSMP III", Canadian Information Processing Society, Canadian Computer Conference Session 1972, June 1-3, 1972. Montreal.

(32) Skjerseth, P.J., "The Use of GPSS for Operational Planning a Numerically Controlled Job Shop," Bull. Ops. Res. Soc. Amer., Vol 20, Suppl. 1, B-183, Spring 1972.

(33) Delle Donne, P.E., and Capehart, B.L., "Some Modeling and Simulating Considerations in Environmental Systems Analysis", Bull. Ops. Res. Soc. Amer., Vol. 20, Suppl. 1, B-149, Spring 1972.

(34) Ingels, D.M., "A System for Simulating Chemical Process Dynamics and Control", University of Houston, Texas. Order No. 71-9531, University Microfilms, Ann Arbor, Michigan.

(35) Schmitz, P., "On the Simulation of Time Dependent Queuing Discipline Using the GPSS Simulation Language", Angew. Inform. (Germany), No. 1, Jan. 1971, pp 31-34, in Germany.

(36) Anon., "Design for Precision", Comput. Rep. (USA), Vol. 7, No. 2, April 1971, pp 8-11.

(37) Carver, M.B., and Likeness, S.L., "FORSIM. A FORTRAN Oriented Simulation Program for the Continuous Transient Solution of Systems of Ordinary Differential Equations", Report No. AECL - 3902, Atomic Energy of Canada, Ltd., Chalk River, Ontario, May 1971.

(38) Biehl, F.A., "Aircraft Landing Gear Brake Squeal and Strut Chatter Simulation", Proc. Conf. on Applications of Continuous System Simulation Languages, pp 201-230, June 30-July 1, 1969, San Francisco, California.

(39) Sturcke, E.H., "Interactive Graphics in Aircraft Landing and Take-off Studies", Proc. Conf. on Applications of Continuous System Simulation Languages, pp 241-246, June 30-July 1, 1969, San Francisco, California.

(40) Murrill, P.W., and Smith, C.L., "Application of Simulation to the Generalized Optimization of Process Control Systems", Proc. Conf. on Applications of Continuous System Simulation Languages, pp 193-196, June 30-July 1, 1969, San Francisco, California.

(41) Hinchley, E.M., "Digital Simulation in Control System Design of the Gentilly Nuclear Generating Station", Proc. Conf. on Applications of Continuous System Simulation Languages, pp 121-128, June 30-July 1, 1969, San Francisco California.

(42) Tramposch, H., and Jones, H.A., Jr., "Impact Problems Efficiently Solved with 1130 CSMP", Simulation, Vol. 14, No. 2, Feb. 1970, pp 73-79.

(43) Schroer, B.J., "Expanded Capabilities of CSMP Graphics on the IBM 1130 Digital Computer", Simulation, Vol. 14, No. 5, May 1970, pp 205-214.

(44) Bell, T.E., "Simulation Analysis with Interactive Computer Graphics ", Instruments and Control Systems, Vol. 43, No. 11, Nov. 1970, pp 109-115.

(45) Palme, J., "A Comparison Between Simula and FORTRAN", BIT (Sweden), Vol. 8, No. 3, 1968, pp 203-209.

(46) Packer, T.J., "An Extended Version of the 1130 CSMP", Simulation, Vol. 13, No. 5, Nov. 1969, pp 231-232.

(47) Chubb, B.A., "Economic Evaluation of the CSMP Digital Computer Simulation Language", Simulation, Vol. 14, No. 3, March 1970, pp 101-103.

(48) Jain, S.K., "A Simulation-Based Scheduling and Management Information System for a Machine Shop", Interfaces, Vol. 6, No. 1, Part 2, Nov. 1975, pp 81-96, TIMS.

(49) Dahl, O.J., and Nygaard, K., "SIMULA - A Language for Programming and Discription of Discrete Event Systems, Introduction and Users Manual", Norwegian Computing Center, 1967.

(50) Markowitz, Hausner, and Karr, "SIMSCRIPT - A Simulation Programming Language", Printice-Hall, Englewood Cliffs, N.J., 1968.

(51) Kiviat, Villanueva, and Markowitz, "The SIMSCRIPT II Programming Language" Printice-Hall, Englewood Cliffs, N.J., 1968.

(52) Gordon, G., "General Purpose Systems Simulation Program", Proc. Eastern Joint Computer Conf., Vol. 20, 1961.

(53) IBM, "IBM General Purpose Systems Simulation Program V Users Manual", IBM Form SH20-0851, 1970.

(54) Bobillier, Kahan, and Probst, "Simulation with GPSS and GPSS V", Prentice-Hall, Englewood Cliffs, N.J., 1976.

(55) Reddy, Y.V., and Bryan, R.H., "DESPL/1 - A Discrete Simulation Language Based on PL/1", Proc. 1973 SCSC, July 17-19, Montreal, Canada, pp 100-112.

(56) Luk, R.H., and Rabideau, G.F., "A Simulation Approach to the Evaluation of Two Telephone Switchboard Systems", IEEE Trans. on Systems, Man, and Cybernetics, Vol. 6, No. 4, April 1976, pp 310-315.

(57) Oren, T.I., "Digital Simulation Languages for Combined Systems, An Overview", Proc. 1973 SCSC, July 17-19, 1973, Montreal, Canada, pp 346-353.

(58) Mitchell, E.E.L., and Gauthier, J.S., "A Table Driven Continuous System Simulation System", Proc. 1973 SCSC, July 17-19, 1974, Montreal, Canada, pp 121-125.

(59) Herring, T.L., and Kiraly, L.J., "Graphical CSSL - A New Tool for Scientific Computation Using Interactive Computer Graphics", Proc. 1973 SCSC, July 17-19, 1973, Montreal, Can., pp 63-65.

(60) Flaugh, R.S., and Delporto, R.W., "Simulation of Automated Materiel-Handling Systems Using GPSS", Simulation, Aug. 1971, pp 65-70.

(61) Ashour, S., and Vaswani, S.D., "A GASP Simulation Study of Job-Shop Scheduling", Simulation, Jan. 1972, pp 1-10.

(62) Pugh, A.L., III, "Dynamo User's Manual", MIT Press, Cambridge, Mass., 1963.

(63) Pritsker, A.A.B., and Hurst, N.R., "A Manual for GASP IV", Purdue University, Lafayette, Indiana, March 1973.

(64) Pritsker, A.A.B., and Hurst, N.R., "GASP IV - A Combined Continuous-Discrete FORTRAN - Based Simulation Language", Simulation, Sept. 1973, pp 65-70. (The paper following treats one example in detail).

(65) Hurst, N.R., "GASP IV - A Combined Continuous/Discrete FORTRAN Based Simulation Language", Ph.D. thesis, Purdue University, 1973.

(66) Lucas, J.J., and Wait, J.V., "DARE P - A Portable CSSL - type Simulation Language", Simulation, Jan. 1975, pp 17-28.

(67) Norwegian Computing Center, "NCC Publications on Simulation and Simulation Languages", Oslo, Norway, April 1974, 17 pp.

(68) Des Roches, J.C., "Survey of Simulation Languages and Programs", Mitre Corp., Report MTR-2040, Bedford, Mass., July 1971.

(69) Pritsker, A.A.B., "The GASP IV Simulation Language", John Wiley & Sons, New York, 1974.

(70) Naylor, T.H., and Jeffress, S., "Corporate Simulation Models: A Survey", Simulation, June 1975, pp 171-176.

(71) Johnson, M.M., and Schneider, M.H., "Simulation of Production Flow Times in a Job Shop", Sci. Proc., Vol. 3, No. 2, Dec. 1973, pp 47-54.

(72) Milner, D.A., "The Computer in Numerical Control of Machine Tools", Sc. Proc., Vol. 3, No. 2, Dec. 1973, p 27-33.

(73) Curry, G.L., Wadsworth, R.B., and Dolbey, B.I., "Simulation Analysis of Design Parameters for a Proposed Production Facility", Sci. Proc., Vol. 3, No. 2, Dec. 1973. pp 3-7.

(74) Brown, T.G., and Morris, R.S., "A Simulation Model for the Analysis of Multistation Parallel - Channel Manufacturing Processes", Sci. Proc., Vol. 3, No. 2, Dec. 1973, pp 95-102.

(75) Pristker, A.A.B., and Kiviat, P.J., "Simulation with GASP II", Prentice-Hall, Englewood Cliffs, N.J. 1969.

(76) Birta, L.G., Raymond, J., and Haqqani, M.A.M., "SIMPAK - An Interactive/Graphics Program for Continuous - Simulation System", Simulation, Oct. 1976, pp 115-122.

(77) Raqazzini, J.F., Jr., and Tapiero, C.S., "Optimum Shuttle Scheduling - a GPSS Simulation", Simulation, Sept. 1976, -- 97-104.

(78) Clementson, A.T., "Extended Control and Simulation Language", University of Birmingham Institute for Engineering Production, Birminghan, England, 1973.

(79) Davies, N.R., "On the Information Content of a Discrete - Event Simulation Model", Simulation, Oct. 1976, pp 123-128.

(80) Davies, N.R., "A Modular Interactive System for Discrete Event Simulation Modeling", Proc. 9th Hawaii Intern. Conf. on System Sciences, Jan. 1976, p 296.

(81) Giffler, B., "DPS: A Dynamic Planning and Scheduling System", TIMS/ORSA Bull., No. 1, 1976, p 116-117.

(82) Bowdon, E.K., Mamrak, S.A., and Salz, R.F., "A Simulation Tool for Performance Evaluation of the IBM 360/75", Intern. Journal of Computer and Information Sciences, Vol. 3, No. 1, 1974.

(83) Hutchinson, G.K., and Wynne, B.E., "A Flexible Manufacturing System", Ind. Eng. (USA), Vol. 5, No. 12, Dec. 1973, pp 10-17.

(84) Golden, D., and Schoeffler, J.D., "Combined Continuous and Discrete Event Simulation", Proc. 1972 SCSC, pp 329-334, June 14-16, 1972, San Diego, CA.

(85) Josephson, P.D., "Interactive Continuous System Simulation Language", Proc. 1972 SCSC, pp 44-61, June 14-16, 1972, San Diego, California.

(86) Oren, T.I., "GEST: General System Theory Implementor - A Combined Digital Simulation Language", Ph.D. Dissertation, University of Arizona, Tucson, Ariz., 1971.

(87) Schriber, T.J., "Simulation Using GPSS", John Wiley & Sons, New York, 1974, pp 533.

(88) Sammet, J.E., "An Overview of Programming Languages for Specialized Application Areas", Proc. 1972 SJCC, Vol. 40, May 16-18, 1972, pp 299-311.

(89) Thompson, F.B., and Dostert, B.H., "The Future of Specialized Languages", Proc 1972 SJCC, Vol. 40, May 16-18, 1972, pp 313-319.

(90) Mawson, J.B., "A Continuous System Simulation Language for an Advanced Hybrid Computing System (AHCSSL)", AICA International Symposium on Simulation Languages for Dynamic Systems, London, England, Sept. 8-10, 1975 (available as reprint from Electronic Associates Inc., West Long Branch, NJ, 07764).

(91) Mitchell, E.E.L., and Gauthier, J.S., "Advanced Continuous Simulation Language (ACSL) User Guide/Reference Manual", Mitchell and Gauthier Accocs., Concord, Mass. 01742.

(92) Herring, T.L., "On the Design and Use of Graphics - Oriented Continuous System Simulation Languages (CSSL)", Proc. ACM Symp. on Graphic Languages", April 26-27, 1976, Miami Beach, Fla., Florida Internil. Univ., pp 123-128.

(93) Herring, T.L., "User's Guide to Graphical CSSL - A New Tool for Scientific Computation Using Interactive Computer Graphics" NWL Tech. Report TR 2905, Naval Weapons Lab., Dahlgren, Va., Jan. 1973.

(94) Young, R.E., and Pritsker, A.A.B., "GASPL/1: A PL/1 Based Simulation Language", Proc. 1974, Winter Simulation Conf., Jan 14-16, 1974, Washington, D.C., pp 24-35.

(95) Rettenmayer, J.W., "SIMPL/1: A Simulation Programming Language", Proc. 1974 Winter Simulation Conf., Jan 14-16, 1974, Washington, D.C., pp 3-12.

(96) Sutherland, D.R., and Tendolkar, N.N., "Exploring the Use of SIMPL/1 for Modeling Queuing Networks", Proc. 1975 SCSC, July 21-23, 1975, San Francisco, Cal., pp 92-97.

(97) Stewart, J.P., "Simulation Languages SIMPL/1, "Simuletter IV/2, ACM, Jan. 1973, pp 84-87.

(98) Wynne, B.E., and Hutchinson, G.K., "Simulation of Advanced Manufacturing Systems", Proc. 1974 Winter Simulation Conf., Jan. 14-16, 1974, Washington, D.C., pp 39-44.

(99) Houle, P.A., and Franta, W.R., "On the Structural Concepts of Simula and Simulation Modeling", Proc. 1974 SCSC, July 9-11, 1974, Houston, Texas, pp 55-60.

(100) Kiviat, P.J., Villanueva, R., and Markowitz, H.M., "The Simscript II. 5 Programming Language", Consolidated Analysis Centers, Inc., 1973.

(101) Hirsch, G.B., Bergan, T.A., Goodman, M.R., and Pugh, A.L., III, "A Dynamo III Simulation Model of the Dental Care Delivery System", Proc. 1975 SCSC, July 21-23, 1975, San Francisco, California, pp 1056-1061.

(102) Gauthier, J.S., and Mitchell, E.E.L., "Large Scale Simulation in the Raytheon Scientific Simulation Language", Proc. 1975 SCSC, July 21-23, 1975, San Francisco, California, pp 133-137.

(103) Alexander, E.L., Jr., "Scope 4: A Simulation Approach to Computer Performance Evaluation Using GASP IV", Proc. 1976 SCSC, July 12-14, 1976, Washington, D.C., pp 6-10.

(104) Fox, M., and Pritsker, A.A.B., "Interactive GASP IV", Proc. 1976 SCSC, July 12-14, 1976, Washington, D.C., pp 3-5.

(105) Toth, P., "Considerations on the Efficiency of Discrete Simulation Languages", Proc. 1976, SCSC, July 12-14, 1976, Washington, D.C., pp 14-19.

(106) Gilliland, M.C., Smith, B.J., and Calvert, W., "HEP: A Semaphore - Synchronized Multiprocessor with Central Control", Proc. 1976 SCSC, July 12-14, 1976, Washington, D.C., pp 57-62.

(107) Brauch, C.J., "Simulation of a Large Computing Facility as a Multiproduct Firm", Proc. 1975 Symp. on the Simulation of Computer Systems, Aug. 12-14, 1975, Boulder, Colo., pp 243-249.

(108) Karplus & Bekey, "Hybrid Computation", John Wiley & Sons, New York, 1968.

(109) Miles, G.E., Peart, R.M., and Pritsker, A.A.B., "CROPS: A GASP IV Based Simulation Language", Proc. 1976 SCSC, July 12-14, 1976, Washington, D.C. pp 921-924.

(110) Segal, R., and White, N., "Representation of Workloads in a Network Environment", Proc. 1976 SCSC, July 12-14, 1976, Washington, D.C., pp 815-818.

(111) Griffith, W.G., Ingerman, D., and Price C.E., "A Simulation Model of Univac's DMS - 1100 -- More Than Just A Performance Evaluation Tool", Proc. 1975 Symp. on the Simulation of Computer Systems, Aug. 12-14, 1975, Boulder, Colo., pp 91-98.

(112) Traub, A.C., Jr., and Zachman, W.F., "A GPSS Model of Complex On-Line Computer System", Proc. 1973 Symp. on the Simulation of Computer Systems, June 19-20, 1973, Gaithersburg, Md., pp 17-37.

(113) Pasahow, E.J., "Simulation of a Real Time Microprocessor Network", Proc. 1975 Symposium on Simulation of Computer Systems", Aug. 12-14, 1975, Boulder, Colo., pp 67-71.

(114) Roth, P.F., "Simulation of Computers: A Tutorial Introduction", Proc. 1975 Symposium on Simulation of Computer Systems, Boulder, Colo., Aug. 12-14, 1975, pp 3-5.

(115) Iwata, H.Y., and Cutler, M.M., "CSP II - A Universal Computer Architecture Simulation System for Performance Evaluation", Proc. 1975 Symp. on Simulation of Computer Systems, Aug. 12-14, 1975, Boulder, Colo. pp 197-206.

(116) Brennan, R.D., and Linebarger, R.N., "A Survey of Digital Simulation: Digital Analog Simulator Programs", Simulation, Dec. 1964, reprinted in McLeod, J., Ed., "Simulation: the ;Modeling of Ideas and Systems with Computers", McGraw-Hill, New York, 1968, pp 244-258.

(117) Anon., "CSSL III User's Guide/Reference Manual", Rev. A, Programming Sciences Corp., Los Angeles, Cal. 1971.

(118) Anon., "Control Data 6000 Computer Systems Continuous System Simulation Language III (CSSL III) User's Guide", Version 1, Publ. no. 17304400, Control Data Corp., Systems Publications, Sunnyvale, California.

(119) Sammet, J.E., "Roster of Programming Languages for 1973", Computing Reviews, Apr. 1974, ACM (A condensed version of the paper was given in Simultter V/4, pp 33-35.)

(120) IBM, "System/360 Continuous System Modeling Program: User's Manual", GH 20 - 0367, IBM Data Processing Div., White Plains, NY.

(121) IBM, "Introduction to 1130 Continuous System Modeling Program II", GH 20 - 0848, IBM Data Processing Div., White Plains, NY.

(122) Forrester, J., "Industrial Dynamics", MIT Press and John Wiley & Sons, 1961.

(123) England, R., "Error Estimates for Runge - Kutta Type Solutions to Systems of Ordinary Differential Equations:, Comput. J., Vol. 12, May 1969, pp 166-170.

(124) Shampine, L.F., and Walls, H.A., "Efficient Runge - Kutta Codes", Sandia Labs., SC -- RR -- 70-615, Sept. 1970.

(125) Goodman, F.K., and Juras, G.E., "NUCLEUS" A Simulation and Modeling System Applied to Demographic, Economic, and Electricity Demand Forecasting Problems", Proc. 1976 Conference on Modeling and Simulation, Pittsburgh, Pa., in press.

APPENDIX E - DATA BASE MANAGEMENT FILE STRUCTURES

FILE STRUCTURE AND ACCESS METHOD

- Sequential
- Random
- List

COMPLEX STRUCTURES

- Indexed Sequential
- Tree
- Network
- Sets

File Structure and Access Method

With present commerical data base management systems, many of the characteristics of their operations are a result of the particular file structure and access method used. We will describe here, the various methods used with their advantages and limitations. These will be general remarks and do not indicate that some of the limitations cannot be resolved by clever alterations, however, these additional correction factors are usually expensive in terms of main memory, storage space, or retrieval time overhead.

We will partition the structure/access methods into three types - sequential, random, and list.

1) SEQUENTIAL - Here, the record is contiguous, its location is based on the value a record's key has relative to other records. (Storage devices tend to be tapes and cards.)

   Advantages - very rapid access to the next file.

   Limitations - a new file has to be written for each update to a record or if a new additional record is inserted, retrieving records out of their normal sequence is virtually impossible, if a file is to be retrieved by more than one key (e.g. a water pump specification may be under the key-engine parts and the key-aluminum parts), then duplicate files have to be created leading to much data redundancy.

2) RANDOM - records are stored and retrieved on the basis of a predictable relationship between the key of the record and the address of the location where the record is stored (Storage devices are drums and discs). Three general methods are used to determine the address:

   a) Direct Address - the address of the Jones' record (for example) i.e., the disc, track, and sector location (number 3469, for example) is known by the programmer and is supplied at storage and retrieval times.

      Advantages - allows equally fast access to all records.

      Limitation - additional effort is required to maintain these direct addresses.

   b) Dictionary lookup - both the address and the record key are stored in a dictionary (table or index). To locate the "Jones" record, the dictionary is scanned for a match on this name. Then the location address is picked up and the record retrieved.

      Advantage - the system maintains the actual address.

      Limitation - additional time required to scan the dictionary, and additional storage space required for the indices.

   c) Calculation or Randomization - the record key is converted through some kind of hash code process into an address.

      Advantage - can retrieve all records equally fast without having to search a data file or index file, and records can be sorted, retrieved, and updated in place without effecting other records in the storage media.

Limitation - may not yield a unique address for each record, therefore, if overlap occurs - causes a condition called overflow therefore have to use pointers indicating where the overflow record is stored.

3) LIST - The basic concept here is to separate the logical organization from the physical organization. The next logical record desired can be "pointed" to, and need not be the next physical record as in sequential organization. Thus, new records can be placed in any space that is available. There are three basic types of list organization:

a) Simple list - pointers are used to cause a record to be a member of as many lists as desired under any number of different keys (like our water pump example).

Advantage - there is no duplication of the record in the data base and, therefore, no multi-updates, in addition, the record can be stored anywhere in the file where there is space.

Limitation - additional space required for pointers, user's system must take into consideration the length of the lists as well as the number of lists in which a record participates - these factors increase the file maintenance overhead time since if a record is deleted, all of the lists it was involved in have to be readjusted to bypass it.

b) Inverted list - makes available every data item as a key. Such an organization requires a table or index of all data values in the system and contains the addresses of all record locations where those values occur.

Advantage - allows access to all data with equal ease - this gives good query and reporting capabilities - good at handling hierarchical data structures.

Limitation - the index table required can be as large or larger than the data itself, as with the simple list system above, there can be much maintenance required in storing and updating data in large tables - this system has difficulty in handling, requests of records located in different branches and/or levels of a hierarchical structure, or located in network type data structures.

c) Ring - the last record in a list points (by a pointer) back to the first (forming a ring structure).

Advantage - very powerful as it provides retrieve and process capabilities in both directions while allowing branching to to other logically related ring structures.

Limitation - again heavy record pointer overhead - these searches can be quite time consuming if the data base is not "tuned" properly - e.g. if the pointers send you back and forth to different discs for each record in the list to be searched.

Compex Structures

In addition, to the simpler methods of storing and relating records described above more complex relational structures can be defined.

1) INDEXED SEQUENTIAL - The file is organized so that records can be accessed either by use of an index or in a sequential fashion

(of the indices or the data records). Indices containing record keys and addresses may exist for each record in the file.

Advantages - it provides some of the speed of retrieval of the sequential file (once you are at the right location) by using indicies to increase the speed of entering the file at the proper place.

Limitations - still large maintenance problems of sequential files with the additional maintenance overhead of indices.

2) TREE - several layers of indices or records are used to establish a tree-branched hierarchy. Indices may be organized as lists or in sequence, with either method's characteristics.

Advantage - convenient in maintaining large dictionaries - allows the data to be structured to represent rather complex data relationships.

Limitation - only a single entry point into each hierarchical relationship - therefore, can require a long time to search the hierarchy for one piece of data. Does not represent a network related data structure, since no branches of the tree touch.

3) NETWORK - specialized form of a hierarchy where all the branches can be interconnected.

Advantage - permits the storage and retrieval mechanisms of the data management system to start with any record in the file and move in multidirections throughout the hierarchy. The network structure allows the data to accurately model real world manufacturing and business relationships.

Limitation - updating and record deletion can involve large maintenance due to the involved relationships.

4) SETS - this is the CODASYL concept of relating data records. Each set type consists of one record type declared as owner and plus one or more record types declared as members. Connection between the owner record and the member records is made by chains (embedded pointers) or pointer arrays (indices). Both tree and network structures can easily be built from these sets. At the least, sets are connected by one way pointers, but the user may also choose to declare two-way pointers for given sets. These sets can then be searched in either direction with equal efficiency. In addition, the member records can have pointers to the owner record, to avoid stepping through the chain to obtain the owner.

☆U.S.Government Printing Office: 1978 — 757-080/294