PROCEEDINGS

THE ANNIVERSARY MEETING

COMMON

JUNG HOTEL

NEW ORLEANS, LOUISIANA

NOVEMBER 28, 29, 30, 1966

PREFACE

This volume records in part the technical material presented at the COMMON Meeting held in New Orleans, November 28, 29 and 30, 1966. A number of committee reports and discussion session summaries are also included. No attempt has been made to include all discussions dealing with the reorganization of COMMON since this topic will be included in subsequent issues of the Secretary's Report.

A real and permanent contribution to this COMMON Meeting is the technical material contained in this volume. Credit for this must go to the authors and the various session chairmen.

Special acknowledgment is due the Local Arrangements chairman, Mr. Earl Dobbs, and his entire team.

I would also like to thank Mrs. Linda Bower, who typed the many iterations of the agenda and assembled this proceedings.

Wiltz P. Champagne, Jr.
Program Chairman
COMMON - New Orleans, 1966

Table of Contents

Page

Preface ii

I. Agenda. 1

II. Monday Session

General Session . . . . *
Chairman, D. A. Jardine

Divisional Meetings

Minutes of Systems Division . . . . *
Chairman, James Stansbury

Minutes of Applications Division. . . . . *
Chairman, Frank Maskiell

Minutes of Administration Division. 14
Chairman, Laura B. Austin

Minutes of Installation Management Division . . . . *
Chairman, Paul A. Bickford

1800 TSX Committee. 18
Chairman, C. Pearson

1800 Systems Project I. 59
Chairman, Open

1800 Systems Project II . . . . *
Chairman, Open

1130 Systems Papers
Chairman, Peter J. Woodrow

"1130 Monitor" - Gene Lester. 61

"1130 Compiler" - Dion Johnson. . . . . *

"1130 Disk and Card Programming Techniques" - Krauf . . . . . *

1620 Systems Papers
Chairman, James Stansbury

"SYMTRAN: The Addition of Formal Algebraic Manipulative Capabilities to FORTRAN with Format" - Mary Clo Carey. . . 62

"A Processor for both SPS and FORTRAN" - James R. Oliver and Guy George 73

Page

"University of Mississippi Test Scoring Program" - Richard D. Ross 84

"A Computer Plotting Language" - James R. Oliver and John McMahon. 111

Applications Division. . . . . *
Chairman, Frank Maskiell

360 Systems Project. . . . . *
Chairman, Richard Pratt

Installation Management and Personnel Training . . . . *
Chairman, Paul A. Bickford

1620 Systems Project . . . . *
Chairman, James Stansbury

Minutes of S. S. P. Application Division . . . . *
Chairman, Frank Maskiell

1620 Application Papers
Chairman, Guy George

"Relocatable Data-Conversion Subroutines for 1620 SPS II" - W. Norris Tuttle. 128

"Teletype Input to the 1620 on an Interrupt Basis While Operating Under Monitor I" - Prof. Don Box, James H. Hughes and Hugh B. Kerr. 134

"General Card to Printer Program" - Janet Allen. 149

"General Format Conversion Program" - Robert B. Balder . . . 161

III. Tuesday Session

Utilities Project. . . . . *
Chairman, E. J. Orth, Jr.

1800 Systems Director-Advanced, Tutorial
Chairman, C. Pearson

"Systems Director" - V. Boyer. 189

1800 Papers
Chairman, C. Pearson

"Hybrid Computer Simulates Stell Process" - Everett L. Keener 236

1130 Systems
Chairman, Peter J. Woodrow

Page

"Commercial Subroutine Package (FORCOM)" - R. L. Louden 253

Languages Project . . . . *
Chairman, James Stansbury

University Education Papers
Chairman, Marv Goldberg

"Computer Requirements for the Undergraduate College" - Robert C. Bushnell 259

"Computerized Library Circulation" - Guy George 265

"USL Student Scheduling" - Jack D. Testerman and Earl K. Turner 272

360 Systems Papers
Chairman, Richard Pratt

"Scientific Computing at an Astronomical Observatory, FORTRAN Language Timings on 360/30, 360/40 and 360/50" - Robert L. Shutt. 276

"FORTRAN Debugging on the IBM 360" - James S. Taylor. . . . . . *

"1620-360 Simulation" - H. Klysen 282

Installation Management Division
Chairman, Paul Bickford

"1800 Education Plans" - G. Wolf. 283

"1800 Available RPQ's and Special Systems" - F. Schneider . . . *

1130 Systems Papers
Chairman, Peter J. Woodrow

"Small University Accounting Systems" - Peter Rhodes. . . . . . *

"1130 User Experience Panel". *

1620 Information Retrieval Papers
Chairman, James R. Oliver

"A Diagnostic Case Presentation Program" - Roger Gudobba, James L. Grisell and Peter Beckett 287

"An Information Storage and Retrieval System for Radiological Surveillance Data" - Nancy A. Paquin and Claudette Thompson. . 295

"RAGE - An Information Retrieval Language" - Jack D. Testerman and Joseph B. Tinker 318

Page

University Education Project . . . . *
Chairman, Marv Goldberg

360 Systems Project. . . . . *
Chairman, Richard Pratt

1800 TSX Committee
Chairman, C. Pearson

"1800 I/O - Advanced Tutorial" - B. Landeck. 325

1800 Systems Paper
Chairman, Open

"1800 Prospro" - H. Bass . . . . *

1130 Papers
Chairman, Peter J. Woodrow

"On-Line Debugging on the IBM 1130" - Peter J. Woodrow . . . . *

"1130 Data Presentation System, Graphic Plotting Language" - Richard E. Weber. . . . . *

1620 Math & Statistics Papers
Chairman, Jack Testerman

"A Disk-Oriented Cross Tabulation Program" - Donald L. Wright. 357

"Sampling with Unequal Probabilities and without Replacement" Ronald G. Kleibrink 378

"Estimation of Power of F Test by Patnaik's Approximation" - Sudhir N. Dalal 393

"Reliability Predictions Using an IBM 1620 Computer" - M. J. Cunniffe and W. H. Bleuel . . . . *

General Interest Papers
Chairman, Greg Payne

"Computer Assisted Painting" - James R. Oliver 401

"Computational Linguistics Program" - James R. Oliver and Sam Baty. 406

"Useful Hints for Writing High-Speed Compilers" - R. S. Milner 426

Panel on T/S vs. Stand Alone . . . . *
Chairman, D. A. Jardine

1800 Absolute Coding Examples. . . . . *
Chairman, C. Pearson

Page

1130 Continuous System Modeling Program
Chairman, Peter J. Woodrow

"Continuous System Simulation" - Robert D. Brennan 433

Open Board Meeting . . . . *
Chairman, D. A. Jardine

Engineering Papers
Chairman, Guy George

"Three-Dimensional Plotting Using the IBM 1620 and CALCOMP 564 Plotter" - R. G. Nelson 436

"DRAFT" - J. R. Birdwell . . . . *

1620 Data Processing Papers
Chairman, Mrs. Carol Hall

"The 1620 as a Data Collector" - Robert L. Shutt 457

"An Alumni Records System for the 1620" - William L. Paxton. . *

"Student Record Keeping by Computer" - James R. Oliver, Russell Schouest and Ronald DeKerlegand 458

IV. Wednesday Session

IBM 1500 Instructional System
Chairman, Frank Maskiell

"1500 Computed Assisted Instruction (CAI)" - Dr. J. L. Stone . 466

1800 Systems Project . . . . *
Chairman, C. Pearson

Contributed Program Library
Chairman, Laura B. Austin

"Contributed Program Library" - F. A. Merks. . . . . *

Conversion Project (Systems) . . . . *
Chairman, James Stansbury

1620 Application Papers-Operation Research
Chairman, James R. Oliver

"Network Analysis in Health Program Management" - Norman L. Dunfee. 473

"The Analysis and Design of a Standarized Program for Continuous Forest Inventory" - John F. Jewell 489

Page

"Project Management - Status Simulator" - Ernest R. Johnson 504

Engineering Papers
Chairman, John Meriwether

"A Subroutine Set for Automation of Logic Circuit Design" - Peter Schneider 528

"Engineering Algorithm Decoder" - John R. Ruckstuhl, Mervin C. Budge and Larry J. LeBlanc. 543

University Education Project . . . . *
Chairman, Marv Goldberg

Impact of Standards on Computer Users
Chairman, Paul Bickford

"Impact of Standards of Computer Users" - J. Farley. 556

General Interest Papers
Chairman, Mrs. Carol Hall

"German-English Translation on the IBM 1620 Computer" - James A. Lawler and Mrs. Mary C. Kerr. 562

"A Self-Organizing Program" - James R. Oliver. 593

"A New Type of Random Number Generator" - R. G. Fryer. 597

1620 Applications Papers
Chairman, Greg Payne

"Computer Programs for Material and Process Selection" - Donald J. King. 608

"Management Simulation Games" - Dr. R. L. Jensen 615

Civil Engineering Papers
Chairman, Open

"Computer Analysis of Pipe Stress" - T. E. Bridge. 624

"On the Efficient Solution of Large Structures" - Norris L. Hickerson 656

Systems Division . . . . *
Chairman, James Stansbury

Applications Division. . . . . *
Chairman, Frank Maskiell

Page

Administrative Division . . . . *
Chairman, Laura B. Austin

Installation Management Division. . . . . *
Chairman, Paul Bickford

* This report or paper was not available at time of printing.

New Orleans COMMON Anniversary Meeting

AGENDA

November 28, 29, 30, 1966

Headquarters Room 261
IBM 1130 Room 263

Sunday, November 27, 1966

7:00 - 10:00 p. m.	Registration - Upper Mezzanine

Monday, November 28, 1966

7:00 - 8:30 a. m.	Registration - Upper Mezzanine
7:30 - 8:30 a. m.	New Member Breakfast, Dutch Treat
8:30 - 10:00 a. m.	Session M-1 M-1.1 General Session Chairman, D. A. Jardine Presidential Salon, Lower Mezzanine
10:00 - 10:30 a. m.	Coffee - Imperial Salon, Lower Mezzanine
10:00 - 11:00 a. m.	Ladies Hospitality Coffee - Pavilion Room, First Floor
10:30 - 12:00 a. m.	Session M-2 Divisional Meetings M-2.1 Systems Division Chairman, James Stansbury Presidential Salon, Lower Mezzanine M-2.2 Applications Division Chairman, Frank Maskiell Terrace Suite # 1 & 2, Sixth Floor M-2.3 Administration Division Chairman, Laura B. Austin Terrace Suite # 4, Sixth Floor M-2.4 Installation Management Division Chairman, Paul A. Bickford Meeting Room # 10, Second Floor
12:00 - 1:30 p. m.	Luncheon, included in Registration Tulane Room, Lower Mezzanine

1:30 - 3:00 p. m. Session M-3

M-3.1 1800 TSX Committee, By Invitation Only,
Chairman, C. Pearson
Meeting Room # 3, Second Floor

M-3.2 1800 Systems Project I
Chairman, Open
Meeting Room # 4, Second Floor

M-3.3 1800 Systems Project II
Chairman, Open
Meeting Room # 5, Second Floor

M-3.4 1130 Systems Papers
Chairman, Peter J. Woodrow
Meeting Room # 2, Second Floor
See below for list

M-3.5 1620 Systems Papers
Chairman, James Stansbury
Terrace Suite # 1 & 2, Sixth Floor
See below for list

M-3.6 Applications Division
Chairman, Frank Maskiell
Meeting Room # 10, Second Floor

M-3.7 360 Systems Project
Chairman, Richard Pratt
Terrace Suite # 4, Sixth Floor

M-3.8 Installation Management and Personnel Training
Chairman, Paul A. Bickford
Meeting Room # 9, Second Floor

3:00 - 3:30 p. m. Coffee - Imperial Salon, Lower Mezzanine

3:30 - 5:00 p. m. Session M-4

M-4.1 1800 TSX Committee, By Invitation Only
Session M-3.1 Continued
Meeting Room # 3, Second Floor

M-4.2 1800 Systems Project I
Session M-3.2 Continued
Meeting Room # 4, Second Floor

M-4.3 1800 Systems Project II
Session M-3.2 Continued
Meeting Room # 5, Second Floor

3:30 - 5:00 p. m. Continued	M-4.4	1130 Systems Papers Session M-3.4 Continued Meeting Room # 2, Second Floor *See below for list*
	M-4.5	1620 Systems Project Chairman, James Stansbury Terrace Suite # 1 & 2, Sixth Floor
	M-4.6	S. S. P. Application Division Chairman, Frank Maskiell Meeting Room # 10, Second Floor
	M-4.7	360 Systems Project Session M-3.7 Continued Terrace Suite # 4, Sixth Floor
	M-4.8	1620 Application Papers Chairman, Guy George Presidential Salon, Lower Mezzanine *See below for list*
6:00 - 7:00 p. m.	Cocktail Hour, Royal Salon	

Tuesday, November 29, 1966

8:30 - 10:00 a. m.	Session T-1	
	T-1.1	Utilities Project Chairman, E. J. Orth, Jr. Meeting Room # 2, Second Floor
	T-1.2	1800 Systems Director-Advanced, Tutorial V. Boyer, IBM Chairman, C. Pearson Meeting Room # 4, Second Floor
	T-1.3	1800 Papers Chairman, C. Pearson Meeting Room # 5, Second Floor Session Starts at 8:45 a. m. *See below for list*
	T-1.4	Commercial Subroutine Package (FORCOM) 1130 Systems, R. K. Louden, IBM Chairman, Peter J. Woodrow Meeting Room # 3, Second Floor
	T-1.5	Languages Project Chairman, James Stansbury Meeting Room # 9, Second Floor

8:30 - 10:00 a. m. Continued	T-1.6	University Education Papers Chairman, Marv Goldbert Meeting Room # 10, Second Floor *See below for list*
	T-1.7	360 Papers Systems Chairman, Richard Pratt Terrace Suite #4, Sixth Floor *See below for list*
10:00 - 10:30 a. m.	Coffee - Imperial Salon, Lower Mezzanine	
10:00 - 11:00 a. m.	Ladies Hospitality Coffee - Pavilion Room, First Floor	
10:30 - 12:00 a. m.	Session T-2	
	T-2.1	Utilities Project Session T-1.1 Continued Meeting Room # 2, Second Floor
	T-2.2	1800 Systems Director Session T-1.2 Continued Meeting Room # 4, Second Floor
	T-2.3	1. 1800 Education Plans, G. Wolf, IBM 2. 1800 Available RPQ's and Special Systems, F. Schneider Chairman, Paul Bickford Meeting Room # 5, Second Floor
	T-2.4	1. Small University Accounting Systems, Peter Rhodes, IBM 2. 1130 User Experience Panel Chairman, Peter J. Woodrow Meeting Room # 3, Second Floor
	T-2.5	1620 Information Retreival Papers Chairman, James R. Oliver Terrace Suite # 1 & 2, Sixth Floor *See below for list*
	T-2.6	University Education Project Chairman, Marv Goldberg Meeting Room # 10, Second Floor
	T-2.7	360 Systems Project Chairman, Richard Pratt Terrace Suite #4, Sixth Floor
12:00 - 1:30 p. m.	Lunch	

1:30 - 3:00 p. m. Session T-3

T-3.1 Utilities Project
Session T-1.1 Continued
Meeting Room # 2, Second Floor

T-3.2 1800 I/O-Advanced Tutorial, B. Landeck, IBM
Chairman, C. Pearson
Meeting Room # 4, Second Floor

T-3.3 1800 Prospro, H. Bass, IBM
Chairman, Open
Meeting Room # 5, Second Floor

T-3.4 1130 Papers
Chairman, Peter J. Woodrow
Meeting Room # 3, Second Floor
See below for list

T-3.5 1620 Math & Statistics Papers
Chairman, Jack Testerman
Terrace Suite # 1 & 2, Sixth Floor
See below for list

T-3.6 General Interest 1620 Papers
Chairman, Greg Payne
Terrace Suite # 4, Sixth Floor
See below for list

T-3.7 Panel on T/S vs. Stand Alone
Chairman, D. A. Jardine
Meeting Room # 10, Second Floor

3:00 - 3:30 p. m. Coffee - Imperial Salon, Lower Mezzanine

3:30 - 5:00 p. m. Session T-4

T-4.1 Utilities Project
Session T-1.1 Continued
Meeting Room # 2, Second Floor

T-4.2 1800 I/O-Advanced Tutorial
Session T-3.2 Continued
Meeting Room # 4, Second Floor

T-4.3 1800 Absolute Coding Examples
Chairman, C. Pearson
Meeting Room # 5, Second Floor

T-4.4 1130 Continuous System Modeling Program,
20 minutes, Robert D. Brennan, IBM
Chairman, Peter J. Woodrow
Meeting Room # 3, Second Floor
Demonstration after talk in Room 263

3:30 - 5:00 p. m. Continued	T-4.5	Open Board Meeting Chairman, D. A. Jardine Terrace Suite # 1 & 2, Sixth Floor
	T-4.6	Engineering Papers Chairman, Guy George Meeting Room # 10, Second Floor *See below for list*
	T-4.7	1620 Data Processing Papers Chairman, Mrs. Carol Hall Terrace Suite # 4, Sixth Floor *See below for list*

Wednesday, November 30, 1966

8:30 - 10:00 a. m.	Session W-1	
	W-1.1	1500 Computed Assisted Instruction (CAI), Dr. J. L. Stone Chairman, Frank Maskiell Meeting Room # 3, Second Floor
	W-1.2	1800 Systems Project Chairman, C. Pearson Meeting Room # 4 & 5, Second Floor
	W-1.3	Contributed Program Library, F. A. Merks, IBM Chairman, Laura Austin Meeting Room # 9, Second Floor
	W-1.4	Conversion Project (Systems) Chairman, James Stansbury Meeting Room # 2, Second Floor
	W-1.5	1620 Application Papers-Operation Research Chairman, James R. Oliver Terrace Suite # 1 & 2, Sixth Floor *See below for list*
	W-1.6	Engineering Papers Chairman, John Meriwether Meeting Room # 10, Second Floor *See below for list*
10:00 - 10:30 a. m.	Coffee - Imperial Salon, Lower Mezzanine	
10:00 - 11:00 a. m.	Ladies Hospitality Coffee - Pavilion Room, First Floor	

10:30 - 12:00 a. m.	Session W-2
	W-2.1 University Education Project Chairman, Marv Goldberg Meeting Room # 9, Second Floor
	W-2.2 1800 Systems Project Session W-1.2 Continued Meeting Room # 4 & 5, Second Floor
	W-2.3 Impact of Standards on Computer Users, J. Farley, IBM Chairman, Paul Bickford Meeting Room # 3, Second Floor
	W-2.4 General Interest Papers Chairman, Mrs. Carol Hall Meeting Room # 2, Second Floor *See below for list*
	W-2.5 1620 Applications Papers Chairman, Greg Payne Terrace Suites # 1 & 2, Sixth Floor *See below for list*
	W-2.6 Civil Engineering Papers Chairman, Open Meeting Room # 10, Second Floor *See below for list*
12:00 - 1:30 p. m.	Lunch
1:30 - 3:00 p. m.	Session W-3
	W-3.1 Systems Division Chairman, James Stansbury Terrace Suites # 1 & 2, Sixth Floor
	W-3.2 Applications Division Chairman, Frank Maskiell Presidential Salon, Lower Mezzanine
	W-3.3 Administrative Division Chairman, Laura Austin Meeting Room # 10, Second Floor
	W-3.4 Installation Management Division Chairman, Paul Bickford Meeting Room # 2, Second Floor
3:00 - 3:30 p. m.	Coffee - Imperial Salon, Lower Mezzanine

3:30 - 5:00 p. m.

Answers to Sound Off Held During General Session
M-1
Chairamn, D. A. Jardine
Presidential Salon, Lower Mezzanine

Section	Title and Author
M-3.4	1. "1130 Monitor" - Gene Lester - IBM - 60 min.
	2. "1130 Compiler" - Dion Johnson - IBM - 30 min.
M-3.5	1. "SYMTRAN: The Addition of Formal Algebraic Manipulative Capabilities to FORTRAN with Format" - Mary Clo Carey - University of Southwestern Louisiana - Elementary - 25 min.
	2. "A Processor for both SPS and FORTRAN" - James R. Oliver and Guy George - University of Southwestern Louisiana - Intermediate - 20 min.
	3. "University of Mississippi Floating Point Subroutines" - Richard D. Ross - University of Mississippi - Elementary - 25 min.
	4. "A Computer Plotting Language" - James R. Oliver and John McMahon - University of Southwestern Louisiana - Intermediate - 20 min.
M-4.4	1. "1130 Compiler", continued for M-3.4 - Dion Johnson - IBM - 30 min.
	2. "1130 Disk and Card Programming Techniques" - Krauf - IBM - 60 min.
M-4.8	1. "Relocatable Data-Conversion Subroutines for 1620 SPS II" - W. Norris Tuttle - General Radio Company - Intermediate - 20 min.
	2. "Teletype Input to the 1620 on an Interrupt Basis While Operating Under Monitor I" - Prof. Don Box, James H. Hughes and Hugh B. Kerr - Tennessee Technological University - Intermediate - 20 min.

Section	Title and Author
M-4.8, contd.	3. "General Format Conversion Program" - Robert B. Balder - Rockville, Maryland - Intermediate - 25 minutes
	4. "General Card to Printer Program" - Janet Allen - PIONEER Computing Department - Elementary - 25 minutes
T-1.3	1. "Hybrid Computer Simulates Stell Process" - Everett L. Keener - Applied Research Laboratory, U. S. Steel - Elementary - 30 minutes
T-1.6	1. "Computer Requirements for the Undergraduate College" - Robert C. Bushnell - Oberlin College - Elementary - 30 minutes
	2. "Computerized Library Circulation" - Guy George University of Southwestern Louisiana - Intermediate - 20 minutes
	3. "USL Student Scheduling" - Jack D. Testerman and Earl K. Turner, Jr. - University of Southwestern Louisiana - Intermediate - 30 minutes
T-1.7	1. "Scientific Computing at an Astronomical Observatory, FORTRAN Language Timings on 360/30, 360/40 and 360/50" - Robert L. Shutt - National Center for Atmospheric Research - Elementary - 15 minutes
	2. "FORTRAN Debugging on the IBM 360" - James S. Taylor - Systems Analysis Department - Intermediate - 45 minutes
	3. "1620-360 Simulation" - H. Klissen - IBM - 30 minutes
T-2.5	1. "A Diagnostic Case Presentation Program" - Roger Gudobba, James L. Grisell, and Peter Beckett - Detroit, Michigan - Intermediate - 30 minutes

Section	Title and Author
T-2.5 Contd.	2. "An Information Storage and Retrieval System for Radiological Surveillance Data" - Nancy A. Paquin and Claudette Thompson - Rockville, Maryland - Elementary - 30 minutes
	3. "RAGE - An Inforamtion Retrieval Language" - Jack D. Testerman and Joseph B. Tinker - University of Southwestern Louisiana - Intermediate - 30 minutes
T-3.4	1. "On-Line Debugging on the IBM 1130" - Peter J. Woodrow - Princeton, New Jersey - Intermediate - 45 minutes
	2. "1130 Data Presentation System, Graphic-Plotting Language" - Richard E. Weber - IBM Manufacturing Industry Development - Advanced - 45 minutes
T-3.5	1. "A Disk-Oriented Cross Tabulation Program" - Donald L. Wright - Georgetown University - Intermediate - 30 minutes
	2. "Sampling with Unequal Probabilities and Without Replacement" - Ronald G. Kleibrink - University of Texas, Medical Branch - Intermediate - 15 minutes
	3. "Estimation of Power of F Test by Patnaik's Approximation" - Sudhir N. Dalal - University of Texas, Medical Branch - Intermediate - 15 minutes
	4. "Reliability Predictions Using an IBM 1620 Computer" - M. J. Cunniffe and W. H. Bleuel - Rochester, New York - Intermediate - 30 minutes
T-3.6	1. "Computer Assisted Painting" - James R. Oliver - University of Southwestern Louisiana - Elementary - 20 minutes
	2. "Computational Linguistics Program" - James R. Oliver and Sam Baty - University of Southwestern Louisiana - Elementary - 20 minutes

Section	Title and Author
T-3.6, Contd.	3. "Useful Hints for Writing High-Speed Compilers" - R. S. Milner - University of the West Indies - Intermediate - 45 minutes
T-4.6	1. "Three-Dimensional Plotting Using the IBM 1620 and CALCOMP 564 Plotter" - R. G. Nelson - Bell Telephone Laboratories Intermediate - 30 minutes
	2. "DRAFT" - J. R. Birdwell - Dow Chemical Company - Elementary - 45 minutes
T-4.7	1. "The 1620 as a Data Collector" - Robert L. Shutt - Sacramento Peak Observatory - Intermediate - 30 minutes
	2. "An Alumni Records System for the 1620" - William L. Paxton - Bucknell University - Intermediate - 30 minutes
	3. "Student Record Keeping by Computer" - James R. Oliver, Russell Schouest, and Ronald DeKerlegand - University of Southwestern Louisiana - 20 minutes
W-1.5	1. "Network Analysis in Health Program Management" - Norman L. Dunfee - Rockville, Maryland - Intermediate - 30 minutes
	2. "The Analysis and Design of a Standarized Program for Continuous Forest Inventory" - John F. Jewell - Michigan Technological University - Intermediate - 30 minutes
	3. "Project Management - Status Simulator" - Ernest R. Johnson - General Motors Institute - Intermediate - 20 minutes
W-1.6	1. "A Subroutine Set for Automation of Logic Circuit Design" - Peter Schneider - Watson Research Lab, IBM - Intermediate - 50 minutes

Section	Title and Author
W-1.6 Contd.	2. "Engineering Algorithm Decoder" - John R. Ruckstuhl, Mervin C. Budge, and Larry J. LeBlanc - University of Southwestern Louisiana - Intermediate - 50 minutes
W-2.4	1. "German-English Translation on the IBM 1620 Computer" - James A. Lawler and Mrs. Mary C. Kerr - The Tennessee Technological University - Intermediate - 40 minutes
	2. "A Self-Organizing Program" - James R. Oliver - University of Southwestern Louisiana - Intermediate - 20 minutes
	3. "A New Type of Random Number Generator" - R. G. Fryer - Sylvania Electronic Systems Williamsville, New York, 14221 - Elementary - 30 minutes
W-2.5	1. "Computer Programs for Material and Process Selection" - Donald J. King - Flint, Michigan - Intermediate - 45 minutes
	2. "Management Simulation Games" - Dr. R. L. Jensen - Emory University - 30 minutes
W-2.6	1. "Computer Analysis of Pipe Stress" - T. E. Bridge - Philadelphia, Pennsylvania Technical - 45 minutes
	2. "On the Efficient Solution of Large Structures" - Norris L. Hickerson - Tennessee Technological University - Elementary - 20 minutes

Administration Division
November 28, 1966

The Administration Division held an organizational meeting at which the objectives and scope of the Division and projects were discussed. There were 18 people in attendance.

By direction of the Executive Vice-President under the full Executive Board, the Administration Division will have broad responsibilities for all activities which are necessary to operate COMMON as an organization. The Division will carry out these responsibilities through projects under the following categories:

Program Library
Reference Manual
Meeting Plans
Communications

Each project is further defined below.

A. Program Library - Scope and Objectives.

1. Act as a steering committee in all matters pertaining to program information distribution from the IBM Distribution Center with the following guidelines:

   a. Determine program and documentation standards for customer contributed programs.

   b. Maintain clear channels with IBM for customers to obtain contributed programs directly.

   c. Establish procedures for review of contributed programs and their subsequent certification or decertification.

2. Scope will cover program information distribution for 1620, 1130, 1800, and 360. COMMON members from the 360 area will work with representatives from GUIDE and SHARE and IBM in a joint effort. The representatives for the other machines will work with IBM Program Information Department representatives.

B. Reference Manual - Scope and Objectives.

1. Present to all member installations the information necessary to understand the organizational structure, the obligations and advantages of membership, and the standards set forth by COMMON.

2. Scope of the Reference Manual will cover:

   a. By-Laws.

   b. Lists of officers and Executive Board members.

c. Membership list.

d. List of current projects and their directors and the latest progress reports of project activity.

C. Meeting Plans - Scope and Objectives.

1. To coordinate the planning of all COMMON meetings with the objective of presenting a unified program of interest to all members and fostering the continuation of existing projects and initiation of new projects.

2. This project will guide the individual Program and Arrangements chairmen and assist them in every way possible.

D. Communications - Scope and Objectives.

1. The scope of this project is twofold, communications with members and inter-user group communications.

2. For communications with members the project shall be responsible for distributing through the International Secretary all correspondence, project reports, and submitted articles or items deemed to be of interest to the general membership. This will also include announcements from the Executive Board, ballots, and information from other users groups.

3. Inter-user group communications will be handled by a member of the Executive Board and will include receiving and giving of information pertinent to other users groups in our relations with IBM and professional groups such as ACM and the American Standards Association. Information deemed to be pertinent to the membership will be forwarded by this project to the International Secretary for inclusion in the Secretary's distribution.

The following Pro tem chairmen were selected for this Division:

Contributed Program Library - PREP Forms
Miss Gaye Baber (1454)
Research Division
National Education Association
1201 16th Street, N.W.
Washington, D. C. 202 223-9400

Reference Manual
B. Roswell Russell (3363)
College of Wooster
Wooster, Ohio 44691

Inter Users Group Communications

J.U.G. Inter Library Exchange

Walter A. DeLegall
Schering Corporation
Bloomfield, N. J.

Meeting
Administration Division
November 28, 1966

Attendees

Charles E. Maudlin, Jr.	Indiana State University, Terre Haute, Indiana
Gaye M. Baber	National Education Assoc., Washington, D.C.
Robert B. Balder	U. S. Public Health Service, Div. of Radiological Health, Rockville, Md
Norman L. Dunfee	U. S. Public Health Service, Div. of Radiological Health, Rockville, Md
Walter A. DeLegall	Schering Corporation, Bloomfield, N.J.
Maxwell Marks	IBM Corporation, White Plains, N.Y.
Robert H. Wilkin	Hooker Chemical, Niagara Falls, N.Y.
J.A.N. Lee	University of Mass., Amherst, Mass.
B. Roswell Russell	College of Wooster, Wooster, Ohio
P. Lonergan	IBM Corporation, White Plains, N.Y.
Arthur F. Hallam	Firestone Tire and Rubber, Akron, Ohio
Fred Caprez	Tacoma City Light
Carol A. H. Hall	Louisiana State Univ., Baton Rouge, La.
James R. Oliver	Univ. of Southwestern La., Lafayette, La.
John F. Keller	Loyola Univ., New Orleans, La.
Joyce E. Stout	Dow Chemical, Midland, Michigan
Stanley G. Knight	Trunkline Gas, Houston, Texas
Robert C. Allen	University of Victoria, Victoria, B.C., Canada

PRELIMINARY REPORT OF THE TSX REVIEW COMMITTEE

Presented at:
COMMON Meeting
New Orleans, Louisiana

November 30, 1966.

Members:

Wayne Barnes	-	SHARE
Dick Edsall	-	IBM
Cliff Foerster	-	IBM
Max Felix	-	COMMON
Charls Pearson	-	COMMON
Sterling Weaver	-	IBM
Gio Wiederhold	-	SHARE

INTRODUCTION

This is a preliminary report of the Time-Shared Executive System for the IBM 1800 Data Acquisition and Control System by the joint TSX Review Committee (TSXRC). COMMON, IBM and SHARE participated in this review. GUIDE was invited to participate but did not elect to do so. The report has been reviewed by IBM for accuracy.

The reader should be aware that the comments in the following sections, for the most part, are concerned with functional aspects of the system rather than performance characteristics. The report is critical in tone since the Committee felt its job was to discover and report potential deficiences in TSX rather than iterate and praise the desirable features, of which there are many.

TABLE OF CONTENTS

	Page
Introduction	2
Philosophy	4
1800 TSX System Generation	5
1800 TSX System Director	8
1800 TSX Nonprocess Monitor Supervisor	13
1800 TSX Disk Utility Program (DUP)	16
1800 TSX Assembler Language	19
1800 TSX FORTRAN	21
1800 TSX Core Load Builder	25
1800 TSX Subroutine Library	26
1800 TSX Simulator	29
1800 TSX Diagnostics	34
1800 TSX Languages	37
1800 TSX Manuals	39

PHILOSOPHY

Early in the Committee's investigation of TSX, three important subjects were discussed. These three subjects continually re-appeared. The Committee believes that they have a major impact on the design of an operating system. As such, the Committee strongly recommends that Users Groups provide definite input on these areas.

The first involves the trade-off between Multiprogramming and Core Exchange methods. The TSX Committee agrees that multi-programming, as a design objective, is necessary for some applications and extremely desirable for the user who is willing to buy a 32K multi-disk drive system. The Committee's definition of multiprogramming is:

Several independent programs residing concurrently within a single main computer storage that makes maximum use of all available machine cycles by transferring control between programs based on some form of priority and the availability of both process and system I/O devices.

For systems with 8K or 16K memory, the core exchange method, such as TSX, is adequate for many applications and utilizes core more efficiently. However, this is generally at the expense of throughput due to the problem of overlapping I/O.

Secondly, there is the distinction between process and non-process jobs. The Committee agrees there are installations where it is difficult to make a distinction between process and non-process jobs and also there are times when non-process jobs may have higher priorities than process jobs. However, other installations require a more rigorous discipline and distinction in the area.

The manner in which Time-Sharing is utilized in a particular installation determines, to a great extent, whether a distinction between process or non-process should be made.

The third subject involves the type of installation and/or applications, of which there are many. For example:

1) Dedicated installations
2) Integrated installations
3) Mature applications
4) Research oriented applications
5) Laboratory data acquisition.

The type of application for which the system is intended controls many of the design considerations. The Committee feels that system modularity is the key concept involved here, but questions whether it is the answer to all problems of this type.

The TSX Review Committee recognizes the need to establish a more effective relationship between User Groups and IBM so that a match between user requirements and future operating systems can be achieved.

1800 TSX SYSTEM GENERATION

INTRODUCTION

This report is based on a review of Phase I TSX Systems Generation. Any changes in generation of TSX Phase II are anticipated to be minor.

System Generation is a process, or series of steps, that generate the TSX system for the 1800. The main purpose is to process certain routines and build the various components which comprise the TSX system and construct the system on the resident disk file. Ideally the only role that the user should have to perform in System Generation would be to define his physical system parameters and assign interrupt levels. The actual building or generation of the TSX system should be handled through the 1800 by a series of system programs that require little or no intervention by the user.

METHOD

The method used in generation of 1800 TSX is to provide the user with a set of detailed procedures that will provide step-by-step instructions from card box to on-line processor. There are approximately 80 (at last count) of these detailed steps that must be performed by the user. In addition, the procedures involve much card manipulation, hand filing of cards, and also involve settings of data, sense, and program switches. The format of the relocatable object decks used for constructing TSX is identical to the format of object decks derived from the TSX FORTRAN and Assembler.

PREPARATION

It is advisable to thoroughly review the entire written procedure before attempting System Generation. The step-by-step instructions, dictionary of all error messages, as well as the formats for all control cards, are presented in the 1800 Operating Procedures, C26-3754. Since the control cards define the entire system to TSX, including interrupt levels, considerable care should be exercised in the planning and preparation of these cards.

REQUIREMENTS

Machine - 8192 words of core, 2310 disk, 1442 card read-punch, 1053 or 1443 or 1816 printer.

Time - System Generation time varies considerably based on individual specifications. Initial system generation requires a minimum of approximately 3 hours:

a) 1/2 hour system load time.
b) 1-1/2 hours Task Assembly (including 1443 output).
c) 1 hour System Director assembly.

Subsequent skeleton builds involve the time required for the recompiling or assembling of user subroutines on line and approximately a 5 minute off-line time to perform the actual skeleton build function prior to new process cold start.

RESTRICTIONS

Skeleton Modification

While relocatable programs can be deleted and replaced on line by the Disk Utility Program (DUP) there is no known method of modifying any features of the TSX skeleton on line. Changes in the skeleton area (including user written skeleton interrupt routines) will require an off-line skeleton build.

System Area Modification

Any modifications (including IBM distributed modifications) of the following system areas must be done off-line using the system loader.

FORTRAN	Core Load Builder
Assembler	Error Detection Prog.
Disk Utility Prog.	Supervisor
Simulator	Cold Start Prog.

However, modifications to the subroutine library may be performed on-line through the Disk Utility Program.

Logical Unit Number

The logical unit numbers (LUN) associated with FORTRAN I/O statements must be assigned during System Generation. There is no way of reassigning the LUN at program execution time.

CONCLUSION

1. Initial System Generation involves many manual procedures that the user must perform. The more manual intervention required the greater the chance of error.

2. Modification to the skeleton area and many of the system programs requires the user to go off-line. This will present a major problem to some totally on-line users (off-line means not controlling process).

3. Logical unit numbers (LUN) cannot be assigned at program execution time. This restriction should be removed to facilitate exchange of programs between users.

RECOMMENDATIONS

1. Possible development of an IBM System Program for multi-drive users that would build a user 1800 TSX system from an IBM supplied TSX Nucleus and user defined system parameters.

2. Modification and rebuilding of the TSX system should be possible on line under control of TSX.

3. A control card method should be developed to assign logical unit numbers at program execution time.

1800 TSX SYSTEM DIRECTOR

INTRODUCTION

The system director directs the handling of interrupts, mainline process programs and error routines and makes the system available to the non-process monitor.

METHOD

The system director resides in core at all times as part of the skeleton. It is read from disk only during a cold start or EAC re-load operation. Primary entry is from internal and external hardware interrupts and calls from the user's programs. The principle components are (see also Figure 1):

a) MIC Master Interrupt Control
b) PSC Program Sequence Control
c) TSC Time Sharing Control
d) ITC Interval Timer Control
e) EAC Error Alert Control
f) Mainline Core Load Queue Table
g) Level Work Areas

A. The MIC (Master Interrupt Control) is a re-entrant routine. It directs all hardware interrupts (internal, I/O and external) and programmed interrupts to the desired routines. Control returns to MIC as long as unserviced interrupts exist.

Interrupts are directed by fixed words in lower core (addr. 11 and up) to the individual Level Work Areas in the skeleton (104 words - one area for each interrupt level) where indicators are set and the index registers are saved. Control is then transferred to MIC which first saves the accumulator and the status word for the interrupt level already in process. The ILSW for the interrupt level to be serviced is sensed. A branch is made via the Level Work Area through the Interrupt Branch Table (IBT) residing within each core load in transient core. From the IBT I/O interrupts are directed to the required routine. For Process interrupts, MIC is re-entered to sense the PISW. A skeleton resident Interrupt Core Load Table (ICLT) is associated with each PISW. The ICLT indicates for each bit in the PISW one of the four possible methods for servicing the interrupt and it contains the required addresses.

1) In-core-with-skeleton
2) Out-of-core interrupt core load
3) In-core-with mainline
4) Record

Methods 3) and 4) may require different information in the

ICLT from one core load to another. The Program Sequence Control PSC makes the necessary changes to the ICLT when the core load is read into core. It obtains the information from the Interrupt Status Table (IST) which resides with each process mainline.

The number of interrupt levels to be serviced must be defined at system generation. Not all the levels defined must be used. The user can, and in fact should, make allowance for future expansion. Should it be necessary in the future to change the number of defined interrupt levels, all core loads without exception will have to be reassembled. This is because the Interrupt Branch Table (IBT) and the Interrupt Status Table (IST) which reside with the core loads have to be amended accordingly. If the ICLT contains no specific information, the interrupt is recorded by default.

Recorded interrupts can be interrogated and reset by the QIFON call in any user written routine, interrupt or mainline. They can also be reset by the execution of a CALL CLEAR.

Whenever an out-of-core interrupt core load is being processed, TSX will mask all other interrupt levels for which out-of-core servicing has been specified. There is no software queueing facility for out-of-core interrupt core loads. The core exchange operations wait pending completion of all I/O operations which depend on parameters or data areas in transient core.

B. The PSC (Program Sequence Control) is responsible for the orderly transfer of control from one core load to the next. A core load may also temporarily be saved on disk pending the processing of another core load (CALL SPECL, CALL BACK). All PSC functions are restricted to process mainline core loads.

The next core load to be executed is indicated to PSC by the execution of CALL CHAIN or CALL SPECL statements or by the VIAQ subroutine.

Though the queue table itself is part of the System Director the queue statements QUEUE, UNQ, QIFON AND VIAQ are designed as subroutines which can be kept in skeleton or with the mainline at the user's discretion. Processing of mainline is not suspended as a result of queuing a higher priority mainline.

C. Multiprogramming was not part of the objectives of TSX. Some degree of time sharing is achieved by the core exchange method under TSC (Time Sharing Control) which is entered from the execution of a CALL SHARE. This statement may be part of the user's process program intended for special applications where time sharing is desired without the use of the queueing technique. The VIAQ also contains this statement for execution when the queue table is empty. As a result, the process core load which is in progress or has

just been completed, is saved on disk and control is transferred to the nonprocess monitor or the nonprocess core load if one has been previously interrupted and stored on disk. The computer remains in the nonprocess mode for a user specified time unless a CALL ENDTS is executed by an interrupt routine. As a result, a complete core exchange again is made and the process program is pursued at the instruction following the CALL SHARE. When located in the VIAQ subroutine, it may simply result in a loop back to CALL SHARE, if the queue is still empty, followed by another core exchange. All waits are performed in the nonprocess mode.

The programmer of the nonprocess core load has no control over the occurrence of a core exchange during the execution of his program.

D. The ITC (Interval Timer Control) services all interrupts involving the three machine timers A, B and C, the nine programmed timers and the programmed real time clock. The programmed timers and the real time clock are based on timer C. It is reset by subtraction rather than by loading of a fixed value. Accurate time is therefore kept even when the response to the timer interrupt itself may be delayed. Timer C also services the "no-response routine" for the 1053/1816 Printers in the skeleton I/O. As an option it services the Operation Monitor during nonprocess execution. Periodic interrupts are available from interval timers rather than from a real time clock. The programmed timers interrogate the ICLT, but only skeleton count subroutines are entered into. If there is no such routine the condition is recorded.

E. The EAC (Error Alert Control) program is called to process all error conditions. The user has the option at system generation time to specify that core be saved under certain error conditions. A user written error subroutine can be included with each process core load. This subroutine is entered before EAC's error decision subroutine. The latter analyses the error, prints an error message and indicates one of the four possible recovery procedures:

Continue through the I/O routine
Reload the System Skeleton and Cold Start Program
Call the User Specified Restart Core Load
Continue Through the Interrupt Level.

PROBLEMS AND SUGGESTIONS

1. The Interrupt Branch Tables (IBT) incorporated in each individual core load (non process, process, interrupt and combination) play vital roles in the servicing of all interrupts. Being located in variable core, they cannot

be storage protected and thus are vulnerable to destruction by programming bugs. The fact that the IBT's depend on the number of interrupt levels defined at system generation time requires the user to plan ahead by defining levels to the system which will not be used initially but possibly at a later date.

A new system generation is required for the definition of additional levels and all core loads have to be rebuilt. The user may thus be required to remain off-line until the required core loads are again available. It has been suggested to include a basic Interrupt Branch Table IBT in the skeleton with all branches to in-skeleton routines storage protected. Branches to interrupt routines residing with mainline may be filled in at core load time and reset on exit from that core load. Their format should be compatible with changes in the number of defined interrupt levels.

2. The Interrupt Status Tables (IST) which reside with all process mainline, and combination core loads depend also on the number of interrupt levels defined.

3. QIFON is the only means offered by TSX to interrogate recorded interrupts. The desired decision may not always be the queueing of a core load. It is suggested to add a routine to test the recorded interrupt and execute a branch if on.

4. A suggested additional function to TSX is to have in-skeleton "load-on-call" servicing of disk resident interrupt routines. A read-in area in the skeleton should be provided for each level of interrupt for which this type of servicing is desired.

5. Some users feel that they are restricted by TSX offering only one level of core exchange for interrupt handling. The reasons for only one level were considerations of disk storage requirements and exchange time.

6. The core load queue is restricted to one entry for each individual core load with the same priority.

7. The VIAQ routine calls SHARE if the QUEUE is empty. Whenever the time period, specified for SHARE by the user at system generation, is up a core exchange takes place before TSX checks whether any core loads have been queued. There is a delay in the attention to newly queued core loads even after execution of the ENDTS routine until the program timers are updated.

   It is suggested to provide a more efficient method for time sharing than SHARE when queueing techniques are used, which avoids unnecessary core exchanges and give immediate

attention to core loads entered into an empty queue.

8. It is realized that the distinction between process and nonprocess work cannot always be clearly made. The programmers should therefore be provided with some option to prevent suspension of "nonprocess core loads" during critical phases.

9. Consider the following types of core loads:

   a) Core loads of very short duration, which must be executed within a matter of seconds at random or periodic intervals.

   b) Core loads of several seconds duration at random or periodic intervals of one minute or longer, which must be executed within their interval.

   c) Core loads of several minutes duration at low frequencies.

   d) Nonprocess monitor jobs.

   e) Background process core loads of lower priority than monitor jobs.

   On-line core loads (a) whose repeat frequency period or permissive time delay is considerably shorter than the execution time of core loads (b) must be able to suspend servicing the latter.

   A similar requirement exists for core loads (b) to suspend (c).

   However, TSX permits servicing of on-line core loads at two distinct levels only, namely INTERRUPT and PROCESS core loads. The servicing of "background" process core loads has not been part of the objectives for TSX.

   As such, TSX does not have the ability to service all of the above core loads as core loads in a true priority fashion. In order to handle the situation, the user must make core loads (a) skeleton interrupt subroutines. To obtain true priority execution, the user must include in skeleton all interrupting programs except one (i.e. one level of interrupt core-load is provided).

   To alleviate the problem, some users feel multi-level exchange is required while other users want a multiprogramming capability or even a combination of the two.

10. It is suggested that the Interval Timer Control (ITC)

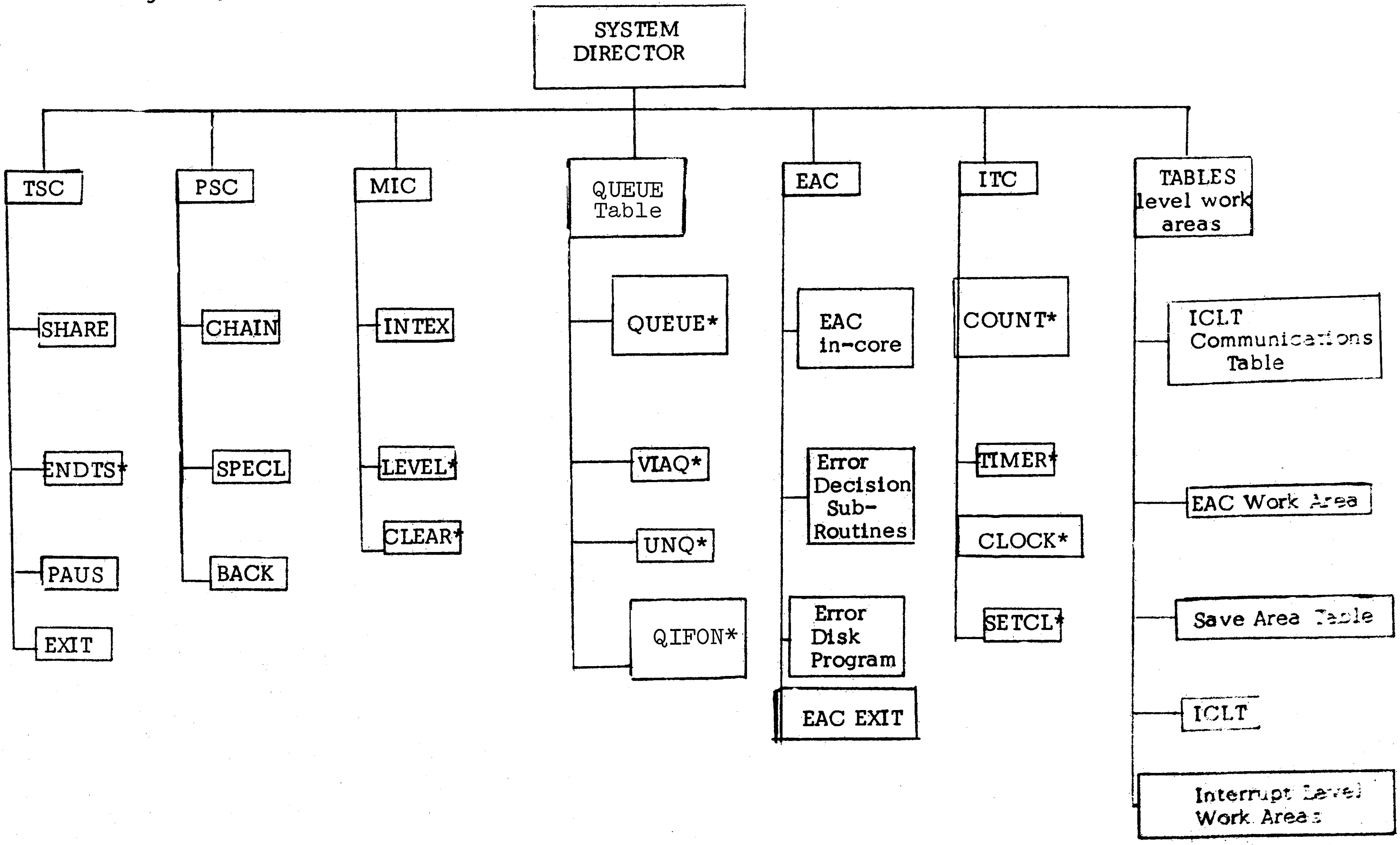
provide as an optional routine a job scheduler. This should be able to queue core loads at fixed times or at periodic intervals with a specified offset to the full hour or the full minute. Specifications to be accepted from control cards and able to be modified when the system is on-line.

Similarly, a calendar routine would be useful for some users.

11. When the EAC initiates a skeleton reload the ICLT and the printer message table are saved and restored. The same philosophy is not maintained in respect to INSKEL COMMON and the queue table. To let the system continue on-line may depend on the availability of meaningful data from INSKEL COMMON and the regular execution of queued core loads. It should be studied how TSX could insure this.

12. A method should be investigated to provide job accounting routines.

Figure 1.



* Not Always in Skeleton

1800 TSX NONPROCESS MONITOR SUPERVISOR

INTRODUCTION

The Nonprocess Monitor (NPM) Supervisor directs the execution of all nonprocess core loads either IBM supplied as part of the TSX package (FORTRAN, Assembler, Core Load Builder, Disk Utilities and Simulator) or user written. It normally operates under the System Director's Time Sharing Control. It can also be run as a dedicated Monitor under TASK.

The functions of the NPM Supervisor are to analyze monitor control record cards; call and transfer control to the requested core load; perform the JOB initialization, PAUS and END OF ALL JOBS functions. It also analyzes control record cards following the *STORECI and *SIMULCI for the core load builder.

METHOD

The NPM Supervisor, including all monitor programs, must reside on logical disk drive zero where it occupies 21 sectors. The first 168 words of sector zero on this disk contains the Non-process Communications Area which provides the logical links between the monitor programs and the users programs. It resides in the 1 8 words at the high end of core. This area slso contains the loader for the monitor programs. There are some unused words in which the sector address and word count for additional system programs could be located.

Entry to the NPM Supervisor occurs through Console Interrupt, from the system director's SHARE routine or from the Disk Utility program.

Analysis of monitor control record cards extends over columns one to five only except for the JOB card. All control records are printed on both the system and the list printers. Invalid control records result in an error message and cause an abort. Blank cards are bypassed and not stacker selected. The card read routine in the Skeleton is used if present and is designed to recognize cards with a / in column one which turns control over to the NPM Supervisor.

The JOB card resets the abort indicator and the effective address for the nonprocess working storage on disk. It can also specify which of logical disk drives 1 and 2 are expected to be running and checks the labels on their disk packs when indicated.

The END card directs the NPM Supervisor into a wait state.

Manual intervention is possible by:

1. Depressing the 1800 CONSOLE INTERRUPT with PROGRAM SWITCH 7 ON to:

a) Enter the monitor initially (Open the Card file)
b) Abort a job in progress prematurely and search for the next JOB card.

2. Depressing the 1800 START to continue after a //PAUS card.

   This still results in the desired continuation of the nonprocess function even though the computer may be in the process or interrupt mode.

3. Stop and restart the 1442 card reader if a job is in progress.

The NPM Supervisor operates in several phases. The largest one occupies 3692 words of core. Analysis is performed at card read speed for most control cards.

RESTRICTIONS AND PROBLEMS

1. The NPM Supervisor, as well as all programs in the monitor package, (FORTRAN, Assembler, Disk Utility, Core Load Builder and Simulator) are limited to the use of the one 1442 Card reader with the lower logical unit number. However, this restriction does not apply to the user's programs.

2. Control records can be entered from cards only. It is agreed that this may be satisfactory to the majority of TSX users.

3. There is no END OF FILE control record. The user must devise his own ways to determine the end of data input.

4. Because the high end of core is overwritten by the Non-process Communications Area, the FORTRAN COMMON area cannot be passed from one monitor function to another except by the use of fixed files. NPM core loads can pass COMMON to another one by the CALL LINK.

5. TSX does not provide a way to split the NPM Supervisor and Disk Utilities from the Assembler, FORTRAN and Simulator. In multidisk installations it may be desirable to keep the latter group on a separate (off-line) disk pack to conserve space on the on-line system disk.

6. Individual programs or phases of the monitor package can be reloaded separately, but only when the system is off-line.

7. The absence of utilities for cards, paper tape and magnetic tape has been noted. Of these, the lack of routines to list and reproduce cards will be most severely felt. The monitor package also offers no routine for the loading of

card resident core image programs directly into core without going through disk. There is no facility for spooling card I/O to disk for unattended operation of time consuming nonprocess programs.

8. When voltage process interrupt is disconnected or turned off, interrupts may occur because of the zero voltage level. Since nonprocess programs cannot CALL MASK, the continuous voltage interrupt cannot be masked out when using the stand alone nonprocess system.

9. Message stored previously by an aborted job are not cleared from the message buffer.

10. There is no provision for source input from magnetic tape or disk for ASM and FORTRAN.

CONCLUSIONS

A considerable part of the NPM Supervisor is dedicated to the analysis of control records for the core load builder. All supervisor functions are relatively simple and executed efficiently.

The restrictions should be carefully reviewed.

Consideration should be given to the complete separation of the FORTRAN Assembler and Simulator programs from the NPM Supervisor. They could be subjected to the same TSX functions like the user's nonprocess programs.

1800 TSX DISK UTILITY PROGRAM (DUP)

INTRODUCTION

The DUP is a group of routines for the maintenance of data and programs on disk packs while the TSX system is on-line. A few functions, however, can only be executed when the system is off-line. One option provides a listing of the contents on the disk packs.

METHOD

DUP is immediately linked to the Nonprocess Monitor (NPM) Supervisor. Like the latter, it must reside on logical disk drive zero where it occupies 68 sectors. It is kept on disk in "wrap-around" address format and operates in about 18 different phases. The largest phase occupies 3692 words of core storage, always at the high end of core. The mode of operation remains the same for all core sizes. Principal entry to DUP is from the NPM Supervisor. Other entries can occur from Assembler and FORTRAN to perform the temporary store of the newly compiled program and to complete the program header information.

DUP uses the card I/O routines in the skeleton if present. Blank cards are skipped and stacker selected when searching for control cards. Other non-DUP or non-monitor cards result in an error message. All DUP control records and messages are printed on both the system and list printer.

Control cards use a column oriented fixed format which, as far as possible, is identical to the IBM 1130 system.

Pre-requisites for most DUP functions to communicate with a disk pack are:

a) Sector Addresses.
b) Numeric Lable in Word 0 of Sector 0.
c) Disk Communication Area (balance of Sector 0).
   This gives information on the size and location of work storage areas and on the Location Equivalence Tables for the relocatable program area (LET) and for the fixed area (FLET) containing core loads and data.
d) Valid entries in LET/FLET.

DUP assumes exclusive responsibility for spotting and deleting programs, core loads and data areas on disk packs and for making corresponding entries in LET/FLET. The disk area for relocatable programs and LET can be packed to eliminate unused spaces which have resulted from deletions. No packing is possible in the fixed disk area for core loads and data (FLET). Unused spaces in this area are used for storage of new core loads or data, provided the space is sufficient. FLET is searched for this purpose and the first space large enough is chosen. The search is not extended to find the smallest possible space. Adjacent void areas are combined into one. No consideration is applied to disk cylinder boundaries.

RESTRICTIONS AND PROBLEMS

1. Practical guide lines for estimating the requirements for the LET/FLET areas are lacking in the reference manuals. Once defined, the allocated space cannot be altered without loss of all data on the disk pack.

2. The *DEFINE CONFG function is a System Generation function and must be done off-line.

3. Some users desire the means by which they can specify the location or re-define the size of core loads or data areas in the fixed area on the disk with appropriate entry in FLET.

   Leaving this responsibility as it is, to DUP, may be satisfactory for fairly dedicated applications where few changes and additions to the initial set of core loads and data areas are anticipated. In applications where frequent changes and additions are made, it will lead to eventual fragmentation of the fixed disk areas. In other words, core loads and data areas will be interspersed by small blocks of unused disk sectors which are too small to store programs. Core loads and data areas may unnecessarily be spread over cylinder boundaries. Increased access times may result. The implementation of packing techniques for the core load area is very difficult.

4. There is a very real danger of destroying valuable information on a disk by the *DLABL function if the pack label was left off on the preceding job card.

5. It has been suggested that TSX provide a default routine which causes a simple abort if the user does not specify a re-start core load.

6. The printer output from the *DUMP and *DUMPDATA options lacks addresses. It is restricted to hexadecimal format. To enhance the value of these options, absolute disk addresses and relative addresses should be provided on the margin. Fixed point decimal and floating point data output format are desirable.

7. The printer output from the *DUMPLET routine is only legible to the trained eye. There are no headings. Improvements to the format and the addition of a version with output (LET * FLET combined) in alphabetical order from one or more disks would help to make it a very valuable tool.

8. Once an interrupt core load has been assigned to an interrupt level and bit, the user cannot get rid of it. He is required to maintain a dummy core load for this purpose. Deletion of a mainline may cause similar problems.

9. To many users the *STOREDATA control card fails to imply the true functions of this routine which is not restricted to data. It can be used for core image programs as well as for allocating data space. Storing of actual data or programs is practically limited to re-loading of cards obtained by the *DUMPDATA option. It would help the users if the dump would also punch a ready-made *STOREDATA card complete with all parameters. It has been suggested that these functions be identified by unique control card names which describe the functions more clearly.

10. The following routines are absent from DUP:

    a) DUMP and LOAD routines with parameters in terms of disk blocks.
    b) COPY routine for all or part of a disk.

11. With the exception of all system relocatable subroutines, TSX system programs are not accessible to DUP functions although FORTRAN, Assembler and Simulator can be removed once the system has been built.

12. Just as it is possible to specify at core load build time (*STORECI) that the core load is to be executed as an interrupt core load when the specified interrupt occurs, it should be possible to specify automatic priority queueing of a mainline core load. The QIFON routine is inadequate for that purpose and depends on high frequency interrogation.

13. The maximum size of fixed data files is 65K words on the *STOREDATA card.

1800 TSX ASSEMBLER LANGUAGE

INTRODUCTION

There has been considerable discussion over the past years in regard to desired assembler functions. Many committees have made recommendations outlining the desired approach in the development of assemblers. There have been several highly sophisticated assemblers developed which operate efficiently in a relatively small amount of core.

The purpose of this review however, is not to discuss assembler criteria but to evaluate the current 1800 Assembler as specified in the IBM Manual C26-5882.

The purpose of the 1800 Assembler as defined in the manual is to generate binary instruction code from mnemonic symbols and use labels for other fields of an instruction. The conversion is one for one; i.e. one machine language instruction is produced for each symbolic instruction. While this approach may be inadequate for an effective assembler, it nevertheless defines what the 1800 Assembler accomplishes. It should be pointed out at this time that the 1800 Assembler was designed to operate in a 4K core environment.

METHOD

The Assembler Program resides only on logical drive zero and occupies approximately 7 cylinders. The source program is read and processed, one statement at a time, twice during each assembly. During the second pass, the statements are read either from the disk or the card reader. If the second pass is read from disk, the assembly is said to be in a one pass mode which is the normal mode for TSX assemblies. If the second pass is read from the card reader, the assembly is said to be in a two pass mode. The Assembler consists of 15 phases which use core based on an overlay method. The phases are loaded and processed automatically and require no manual intervention.

The Assembler uses an 80 word I/O buffer (SAREA) for all I/O functions. The contents are right justified one character per word and the card image format is converted to EBCDIC code one column per character and replaced in the buffer. The time required to perform an assembly seems to be dependent on the speed of the I/O devices used.

RESTRICTIONS

1. The Assembler Program can reside only on logical drive zero.

2. Input to Assembler must be from cards. Paper tape and magnetic tape input is not supported.

3. Only one 16 bit word is generated for each DC constant. This increases the size of the source deck thereby increasing read time.

4. There is no automatic relationship provided with INSKEL COMMON. The only way in which this relationship can be established is through a reference to 2 words in the fixed area which contain the high core address and size of INSKEL COMMON.

5. If an END card is not present in the input stream, pass 2 of the assembly is not executed and results are not obtained.

6. The END card remains inside the 1442 until the first card is punched at the end of pass 2. A separate card feed should be issued at the end of pass 1 so that the entire source deck is available to the operator insuring against the misplacement of the END card.

7. Blank cards encountered in the input stream cause an illegal op code diagnostic during pass 2. It should be possible to ignore and stacker select blank cards prior to the END card.

8. Use of a period to indicate EBCDIC coding in DC statements causes confusion since in other languages this indicates a constant with a decimal point.

9. All transfer vector LIBF subroutine calls generate a short BSI instruction tagged by index register 3(XR3). If the assembly language programmer utilizes XR3 within his program, it becomes his responsibility to restore XR3 to point to the transfer vector (TV) prior to issuing a LIBF. This is accomplished by an (LDX I3 103).

10. The EPR statement does not change library subroutine calls to extended prevision format nor does it change floating point constants to extended precision.

11. The operation of MDX and BSC instructions perform a multitude of functions. No mnemonics are provided to help identify each function.

12. There is no cross reference listing of symbols indicating the various places in the program where the symbols are referenced.

CONCLUSION

In general, the 1800 TSX assembly processor appears to be quick and reasonably efficient. The language itself could use more mnemonics to clarify statements; e.g. MDX and BSC, but also important is the need for MACRO instructions. IBM supplied MACRO's e.g. I/O routines and internal record movement are essential for an effective assembly language. Perhaps even more critical however, is the need to provide the programmer the facility to construct his own MACRO's. Without MACRO

capabilities, the Assembler is reduced to the simple role of mnemonic translator.

RECOMMENDATIONS

Solutions to the restrictions presented in this report should be reviewed for possible early implementation into the Assembler. This is especially urgent in the case of MACRO's and providing additional mnemonics.

1800 TSX FORTRAN

I. INTRODUCTION

The 1800 FORTRAN compiler was evaluated in comparison with currently existing FORTRAN compilers, specifically on IBM 1620, 7090 and 360 equipment.

The sections following summarize the external and internal aspects of the 1800 TSX implementation.

II LANGUAGE

The language is substantially like the

ASA BASIC FORTRAN as described in reference (3).

General

The language, although still basically BASIC, has been enriched in several useful respects. Certain features of the implementation for process usage introduce incompatibilities.

The following items should be noted when using TSX FORTRAN to compile decks used on other FORTRAN systems:

1) //FOR control cards have to precede every routine in the deck to be compiled.

2) FORTRAN COMMENT cards may not appear before continuation cards nor after END cards.

3) Requirements on the placement of DIMENSION, COMMON, EQUIVALENCE statements. The requirements are not unreasonable, but have to be checked when converting existing programs. See also Section III(1).

4) It is possible to equivalence a one word integer array with two and three word integer arrays. Care should be exercised when doing this.

5) Additional disk input/output statements such as FIND are included and will prove to be very useful in reducing overhead. These instructions do not exist in the smaller 360 FORTRAN compilers.

6) Arrays may be 3-dimensional.

7) DATA statements are included.

8) Hollerith fields may be specified in quotes, thus avoiding many counting errors.

9) REREAD, that is: "The ability to read a card, analyze its type, and then reread it with an appropriate format" is absent here as it is from all of IBM's FORTRAN compilers. This feature, however, is very desirable. See reference 5.

10) Alternate FORTRAN I/O subroutines for free field input would be useful in some environments.

11) Integers, although assigned two or three words for reasons of proper equivalencing, use only one word and are therefore limited in size to ± 32767. This is sufficient for subscripting and counting use, but not generally for integer computations (payroll, integer linear programming, etc.). It would be desirable to include two-word integer arithmetic.

12) Due to the 2's complement machine logic there is no integer representation for minus 0. This feature has been frequently used in statistical programs on 7000 series machines to indicate missing data, rather than 0 value data. The same problem exists in the System/360.

13) One special COMMON block:/INSKEL/ is defined to communicate with skeleton data areas. This feature follows the syntax of full FORTRAN.

14) STOP, CALL EXIT and CALL LINK are only to be used in nonprocess programs, whereas process programs for similar functions have to issue CALL VIAQ and CALL CHAIN. This complicates the change of program status from non-process to process and vice versa. This is confusing and not necessary.

15) EXTERNAL statement is included.

III PRODUCED OBJECT CODE AND OTHER OUTPUT

The compiler produces code with many subroutine references for the basic operations which limits speed of execution somewhat, but is probably by far the best compromise in the core size versus speed argument.

1) When non-process compiling is done to process programs that are to be run as process the ONE WORD INTEGERS control card should be used to avoid incompatibilities between the two modes. Without this precaution the EQUIVALENCE mechanism is altered and DATA entries may be changed when recompiling.

Individual declarations by variable name (see FORTRAN 360-H level) would be desirable to enable the specification of one word FORTRAN integers used in process control and leaving all other integer sizes compatible with real variables of either regular or extended precision.

2) The code is unique in that in order to provide software - memory - protection all subscripted store and read statements are compared with the limits of variable core (not with the dimension declaration) to protect process-routines and the system. The error messages given by the System director (see chapter on Diagnostics) are unfortunately not helpful in locating the source of the error. An expanded error message would be a major help in debugging.

A side effect is that repetetive store operations of a subscripted variable may, in order to save some execution time, be better carried out using a temporary variable. The relative speed gain is not much though since all operations, even stores, are executed through subroutines; also more core storage may be required. Example: Summing the columns of a matrix S into an array A:

```
  DO 11 = 1, 10                    DO 10 1 = 1, 10
  TEMP = 0.                        A (I) = )
  DO 10 J = 1, 10  rather than     DO 10 J = 1, 10
10 TEMP = TEMP + S (I,J)         10 A (I) = A(I) + S (I,J)
11 A(I) = TEMP
```

3) An adequate listing of variables and their relative locations is produced. There is no means, however, of obtaining a symbolic or even hexadecimal listing of the compiled code. This will make the determination of clobber-type errors considerably more difficult.

4) The compile time diagnostic capabilities are non-mnemonic (See Diagnostic report).

IV IMPLEMENTATION

The method of compiling is described in the Program Logic Manual.

Essentially the entire deck is read into a core memory area in squeezed form and then the 28 sequentially phases of the compiler transform the text in a more and more coded representation while on the other side of this core area a symbol table is built up. Eventually the symbol table information can be written out as a MAP and then the transformed source program has become the object program

and is written out. A large amount of handling of the source information and its transformation takes place since the source string waxes and wanes throughout this process. However, since in this machine the processor is so much faster than input-output this approach leads to fairly fast compile times. The program size is limited by the size of the core memory area, namely to about 120 cards for an 8K system. An 8K system, however, will not normally be able to load useful programs of this size, so that this limitation is then valid only when execution on a larger machine is desired.

V. STORAGE REQUIREMENT AND TIMING

Core Storage: Minimum 3692 at the high end of core.
Disk Storage: 103 sectors.
Speed for 150 statement program:
    Without Punching: 47 statements per minute.
    With Punching: (50 cards) 38 statements per minute.

VI REFERENCES

1) IBM 1800 FORTRAN Language C26-5905-3
2) IBM 1800 TSX System, Program Logic Manual Y26-3702-0
3) Communications of the ACM Volume 7 #10 Page 591
   ASA Standards for FORTRAN
   ASA Standards for Basic FORTRAN
4) OS/360 FORTRAN-H Level Programmers Guide
5) SHARE Secretary Distribution 157 C-4462:
   ENCODE/DECODE facilities for memory to memory data conversion and transmission in 360 FORTRAN

1800 TSX CORE LOAD BUILDER

INTRODUCTION

The Core Load Builder (CLB) builds all core loads that are to be loaded and executed in variable core. These core loads may be under control of the System Director or the Non-process Monitor. The basic purpose of the CLB is to combine a relocatable program with all called relocatable subroutines into an executable core load. This includes establishing all subroutine linkages, hardware interrupt servicing linkages, and creates all communication areas that are required.

METHOD

CLB uses a 2 pass method each consisting of various phases. Both passes are processed under control of the Master Control Routine (MC). Pass 1 contains the initialization and scan phases. The initialization phase sets up loading tables for use in subsequent phases and also creates entries in the loading tables based on information contained in the Control Record Entry Table (CRENT). The scan phase determines subroutine entry points, load addresses, and core storage requirements. Pass 2 contains the adjustment phase and load phase which are necessary to finalize the communication area and complete the building of the core load.

RESTRICTIONS

Subroutine names that appear in a calling sequence, e.g. CALL TIMER (SUB,I,J) must be defined in an EXTERNAL statement. There is no check made by CLB to insure that the subroutine name was defined in an EXTERNAL statement for system subroutine of this type. Under these conditions, an improper core load will be built and executed with unpredictable results.

CONCLUSION

In general, the CLB appears to perform all of its required functions in a satisfactory manner. The CLB is essentially I/O bound and the elapsed time for building core loads varies from a few seconds to approximately 2 minutes depending upon the size of the core load, the number of disk drives utilized by the system, and the number of core loads in the system.

1800 TSX SUBROUTINE LIBRARY

I. INTRODUCTION

The arithmetic and input/output libraries are discussed in the following sections. No tests have been run to verify accuracy, timing, etc., by the TSX Review Committee.

II. ARITHMETIC LIBRARY

The Arithmetic Library contains both the routines visible to the FORTRAN programmer as well as the many routines that are used by the FORTRAN generated object code and that may be used by the ASM programmer.

The Function - Evaluating routines use polynominal approximations to avoid the slow divide procedure, and call many of the invisible routines. Square root is evaluated by iteration. Arithmetic is symmetrically truncated to follow the rules for the machine logic. An indication of time required is given in Section V. The times required are significantly increased over those of other 1800 routines due to the requirement for re-entrant coding which enables these routines to be used from many levels concurrently. All calls are of the standard format, but all have entry points also to allow argument communication via the accumulator. The checking for variable core limits is to be done outside of the routines.

An additional, and very useful feature exists that allows testing of error indicators set by the functional routines through a FORTRAN call. This feature is lacking in all other FORTRAN systems and would be even more useful if standardized and made available in the other systems.

Among the routines invisible to FORTRAN programmers is a double word fixed point-fraction multiply and divide routine which could be useful to ASM programmers that are willing to do their own scaling.

III. INPUT/OUTPUT LIBRARY

The library routines support the data processing I/O in a straight forward manner. There is no central IOCS so that all error checking is done per I/O call. This leads to the fact that there is no system overlap of computing and I/O during DPI/O operations with the exceptions of the 1053 and 1443; however, the assembler user can overlap I/O with computing. Certain features of the 1442 are not describable by FORTRAN statements. This led to the fact that its stacker select feature and its last card indicator is not usable from FORTRAN. The disk routines use the disk in such a way that no rotational delay occurs if there is no interference from higher priority level programs.

All the I/O routines follow standard linkage conventions. Most of them save in addition, the A register to facilitate the interrupt coding.

There is also a 420 eord routine FBTD/FDTB to convert from standard and extended floating point to EBC (1800 EBCDIC) which is not used by FORTRAN I/O but can be used by ASM programmers to save memory space by avoiding the FORTRAN I/O routines. If there are any FORTRAN I/O statements, however, it would be advisable for the ASM programmer to use FORTRAN type I/O calls.

IV. PROCESS INPUT/OUTPUT LIBRARY

The process input/output library allows FORTRAN control over all process input/output operation, a facility that has not been previously present within FORTRAN systems. A fair knowledge of the machine and of FORTRAN is required to make effective use of the routines.

It is yet too early to evaluate fully the usefulness of these routines. Experienced process engineers may prefer to use assembly language input/output because of the straight-forwardness from an engineer's standpoint, but may find that the interrupt structure of the 1800 is such that the much greater ease using these subroutines will become an over-riding consideration. The processing of an I/O interrupt takes approximately 100 microseconds on a 2 micro-second machine, which compares favorably with the approximately 60 microseconds achievable by hand coding.

Recommendations

1) Re-entrant Code

   It would be desirable to have a separate set of non-re-entrant subroutines for batch processing use and single-level on-line use for maximum throughput.

2) Input/Output Subroutines

   It would be desirable for IBM I/O subroutine to notify the user of operation complete. This can be established through the inclusion of a subroutine name in an I/O calling sequence. This would allow the user to overlap I/O operations, thus avoiding I/O busy testing.

V. TIMING (2 microsecond memory)

These are approximate indications only.

Function	Name	Time Standard Precision	Time Ext. Precision
sine and cosine	FSIN, FSINE, E... FCOS,FCOSIN,E..	3.2ms	5.2ms
arctangent	FATAN,FATN,E...	5.1	9.2
square root	FSQRT,FSQR,E...	5.2	11.8
logarithm	FALOG,FLN,E...	4.6	4.5
e^x	FEXP,FXPN,E...	1.9	4.2
x^i	FAXI, FAXIX,E...	2.5	3.7
x^x	FAXB,FSBXB,E..	7.9	14.9
hyperb. tang.	FTANH,FTNH,E...	4.2	8.0
+ X	FADD,FADDX,E...	.49	.51
- X	FSUB, FSUBX, E...	.49	.51
- (... -x)	FSBR, FSBRX, E	.80	.82
*	FMPY,FMPYX,E..	.46	1.06
/	FDIV,FPIVX,E...	.75	2.10
/ (.../x)	FDVR,FDVRX	.78	2.34
fetch	FLD,FLDX,E..	.27	.32
=	FSTO,FSTOX,E...	.27	.33
float	FLOAT	.70	
fix	FIXI,FIXIX	.49	
$\|x\|$	FABS,FAVL,E...	.28	.29
bin to dec	FBTD	12.0	
dec to bc	FDTB	23.0	

1800 TSX SIMULATOR

INTRODUCTION.

The simulator is a major advance in debugging tools for the on-line programmer and will be highly useful to most installations, whether on-line, or off-line. It operates under control of the TSX Nonprocess Monitor, and allows the programmer to check out or test a program without interfering with or endangering the regular operations of the on-line system.

Because so little information has been generally available about the simulator, the TSX Review Committee felt it advisable to go into somewhat more depth in this section than in most others.

METHOD

The simulator can be used to debug process and non-process programs alike. It checks all core references, especially stores and branches. Simulated COMMON can be dumped on cards so that a run can be executed in several different parts.

There are options for:

a) Branch trace
b) Snapshots
c) Dumps.

In addition, the branch and arithmetic trace provided by the compiler can be operative in the simulator mode, but there is no full trace in the simulator.

Process input values may be read from cards or obtained from a random number generator. However, it may not be read from the 1816, directly from the process inputs, or calculated by user defined function routines.

If a program were being simulated under control of an off-line system, it would take about 120 times as long to execute under simulation as under direct execution. However, when under the control of the system director, it could take much longer due to the demands of the on-line process. There is no output option to show how much time a program would have required in real time.

SUBROUTINE SIMULATION

If the simulator knows (by a set of internal tables) that an IBM subroutine does not use I/O, then it is executed at machine speed; however, the simulator uses its own copy of the subroutine. IBM routines which use I/O are functionally simulated rather than being simulated step by step. User written utility routines could be added to these lists by the following methods; there are three cases:

a) Skeleton routines to be executed (non-I/O). There are two tables in the Initialize 2 (INIT2) section which must be modified. SXT contains data about skeleton subroutines to be executed, and NAll contains the EBC names of all subroutines in SXT. These tables can be modified by ordering the simulator in source form through IBM branch office procedures (however, this is about 20,000 cards and requires special assembly procedures). The user can add the entries to these two tables and then assemble the simulator.

b) I/O routines to be functionally simulated. Again, there are two tables which must be modified, NAll and either FST or TRT which are similar to SXT in that they contain data about the routines. The difference between FST and TRT is explained in the discussion of calls which terminate simulation. I/O routines which are functionally simulated must have consistent linkage format with those produced by the TSX compiler.

c) Any other programs - IBM recommends very strongly against executing or functionally simulating any programs other than those discussed in (a) and (b).

CALLS WHICH TERMINATE SIMULATION

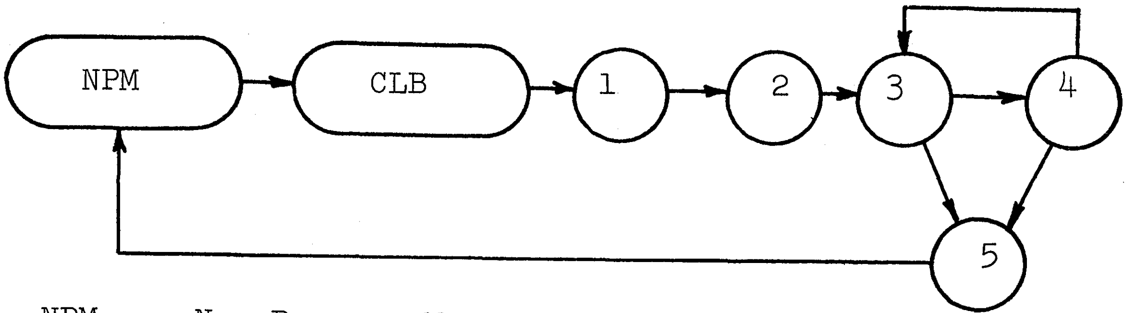
Simulation terminates with the execution of calls to:

BACK
CHAIN
VIAQ
SPECL
INTEX
DPART
STOP
PAUSE

The simulator tests one program at a time. There is a way for the user to modify the simulator so that anyone or more of these calls will never be terminated. The TRT table contains entries for all the calls which terminate simulation. By moving the entry for a given call from the TRT table to the FST table and adjusting the NAll table to correspond, that call will no longer cause termination. However, if termination is removed for other than CALL PAUSE, the user must evaluate the consequences very carefully. A rewrite of the whole simulator could be involved.

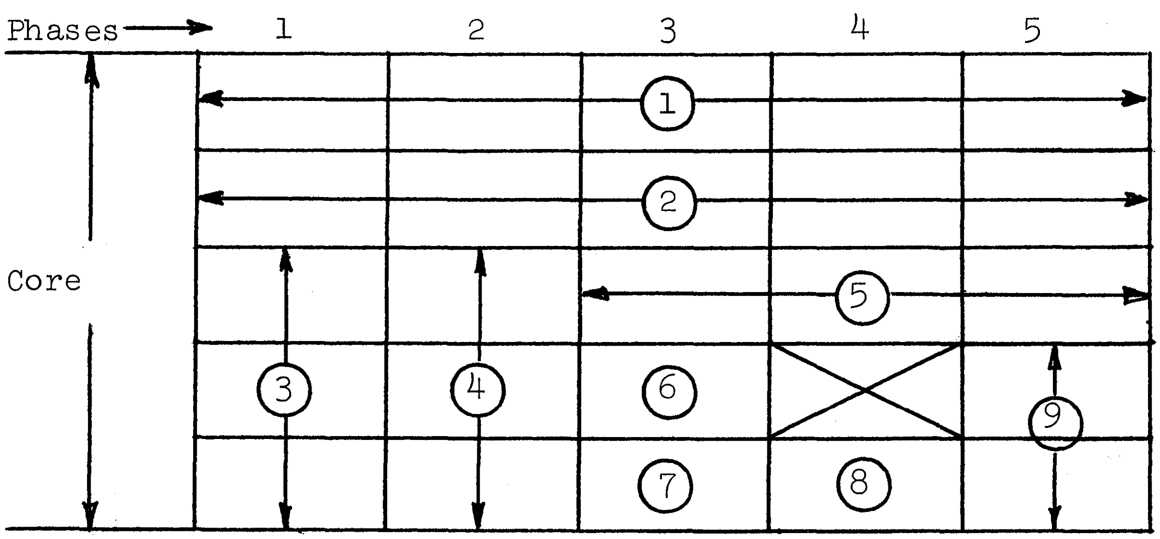
SIMULATOR SECTIONS AND EXECUTION PHASES

There are nine sections of the simulator executed in five phases as in the following phase transition diagram:



NPM = Non-Process Monitor

CLB = Core Load Builder



Sections

1) Executive Monitor
2) Resident 1
3) Initialize 1
4) Initialize 2
5) Resident 2
6) Subroutine Monitor
7) Instruction Interpreter
8) Subroutine Processor
9) Termination

CONCEPTIONS AND MISCONCEPTIONS

The simulator is not meant to simulate a process or any other random, parallel, asynchronous event. Since it is not aware of time, it is not aware of parallelism as such. Thus, it cannot be used as a system scheduler and resource allocation for an operating system as some users have suggested. Nor can the simulator be used to simulate the user's process as in a general system simulator or digital differential analyzer. Once the user accepts these facts, he will find that it is a rather good diagnostic and debugging tool. The program might be renamed to eliminate this confusion. It perhaps could be called the Diagnostic Interpreter since it intreprets 1800 instructions on the 1800 for diagnostic purposes.

RESTRICTIONS & RECOMMENDATIONS

1) Options to allow reading of data from the 1816, or directly from process inputs should be considered. Reading directly from the process inputs would require a special system parameter to denote whether reading analog input directly from the simulator is legal or not since in certain instances this could interfere with the normal on-line functions of the system. Simulations of the I/O test feature would have to be changed if access to actual process inputs were allowed.

2) There is no output to show how much execution time a program would have required, assuming no on-line interference. An instruction count in the snapshot information could be added. The user could multiply this by an average instruction time for the instruction mix being executed to tell how much time would have been required.

3) The requirement for punching leading zero's in control cards is a minor inconvenience. It was done to make the coding shorter in a program for which space was limited.

4) The DISKN simulation routine will write only into the non-process work area. If a process program being simulated tries to write in the process work area, it will print the first and last 8 words on the list printer and does not write anything into the process work area. This would present problems if the program is attempting to write into the process work area and then read it back, but is necessary to protect the process.

5) The following restrictions and recommendations concern the data input facilities of the simulation:

   a. Simulator does not provide specification for multiplexer address on card input. The problem of reading random multiplexer addresses from a sequential card hopper could be solved by reading all data cards to disk and setting up a separate file for each multiplexer address. The simulator could call each input by referring to the proper file based on the multiplexer address of the point to be read.

   b. A multiplier option for data cards that would act across repeats for AIP and act across addresses for AIR would be useful.

   c. There should be an option to specify random number generation on individual data cards when CARD has been specified on the control card.

6) The ability to take simulated core and disk off-line as in a checkpoint and restart from the same simulated core or just get further diagnostics should be considered for multiple drive systems.

7) The XIO and WAIT control cards should be changed to NO XIO and NO WAIT for functional clarification.

8) CALL PAUSE should be simulated rather than causing termination.

9) It should be pointed out that there is one discrepancy between the actual operation of the comparator feature and the way it is simulated.

In actual operation, the ADC reads one analog point at a time and the comparator checks for a limit violation in parallel. The reading and comparing proceed independently of each other. If a violation occurs, the I/O routine exits to a user subroutine. This subroutine returns control to the I/O routine. When the next point is read, the system may either take the check routine again or continue reading points as the data dictates. In any case, the end-of-table interrupt routine is always executed whether a limit violation occurred or not.

The comparator is simulated by first reading all the data for one analog read and then starting a programmed compare sequence. If no compare violation exists, control is transferred to the end-of-table interrupt routine. However, if there are one or more limit violations, the first limit violation causes an exit to the check routine. The check routine returns control to the program mainline at the first instruction following the analog read instruction instead of back to the simulator compare sequence. The end-of-table interrupt routine will not be processed, and if there are more than one limit violations, only the first one will be detected and checked.

1800 TSX DIAGNOSTICS

I. INTRODUCTION

The diagnostics from the various segments of the system are summarized below. A separate section is devoted to the implementation of the memory protection facility of the 1800. The utility of diagnostic messages is increased if the time to take corrective action is minimized. In order to achieve this, diagnostics should be:

1) Addressed to the person capable of taking corrective action
2) Clear
3) Complete.

In addition, errors affecting on-line use of 1800 system should require special care.

II. METHODS

The following list covers the TSX system areas with some remarks on their diagnostic methods:

1) TASK :codes for user lookup
2) System Loader :codes for user lookup
3) Cold Start :codes for user lookup
4) Skeleton Builder :codes and very abbreviated messages.

The above four areas are such that most errors are caused by incorrect deck set up by the programmer. During the above tasks one can assume that a responsible and knowledgeable person is either present or available.

5) System Director :codes with additional information.
There are both messages that are routinely operator-oriented, and others that can be caused by the programmer, and some of these are of the type that may occur in checked out production type problems.
The 23 codes prefixed by 9 or I are generally operator-oriented. The remaining 24 codes prefixed F, M, P, Q, or X are program oriented. With these messages time, device, program and location are given. The last two, if available, are not necessarily linked directly to the source of the error. Certain of these errors are machine errors and have therefore the priority level 0. More than one of these in succession forces a reload action. Some of these 23 codes may require operator action to continue the process program operation.

6) Supervisor :codes and abbreviated messages
7) DUP :codes and small messages

In the two systems above, many of the messages are programmer directed. Some, however, indicate operative problems. It seems doubtful that an operator will be able to sift through these quickly.

8) The Assembler :codes with messages, or flags on error lines.
The error flags follow established practice and specify sufficiently the error cause to the programmer.

9) FORTRAN :codes only of form Cxx.

General

The first four parts of the system, used during system generation, have different format, conventions, content and location of their error indications; however, all on-line messages have a basic error format.

Where the system is operated by non-programmers, difficulty may be experienced in training operators to sift the system output for information relevant to them. A lack of messages, or excessive abbreviations, may be felt as an annoyance in many practical situations.

III. RECOMMENDATIONS

1) Special indications for errors that may be operator correctable
2) Mnemonical diagnostic codes for FORTRAN, with an indication of column or text item causing the diagnostic.

Currently a message: "C27" which becomes after lookup in the operations manual (they are not listed in the FORTRAN manual) "SYNTAX ERROR IN FORMAT STATEMENT" is not as helpful as a diagnostic should be.

3) The system director messages for internal errors should include the contents of the Instruction and Index Registers, Accumulator and Extension to facilitate trouble shooting.

IV. HARDWARE DIAGNOSTICS

Some CE hardware diagnostics can be run in the 1800 system through special hardware facilities without disturbing other programs and operations, during the running of the TSX system. These routines can exercise minimally the I/O devices. The full CE diagnostics monitor cannot run under TSX.

V. STORAGE PROTECTION

The storage protection feature is used by TSX as follows:

The system-director code is protected from destruction. This is accomplished by the cold-start routine which assigns a storage protect bit to every non-zero word. Zero words are assumed to be non-changeable programs and to be storage protected except where they appear in a cold-start exception table.

Code generated by FORTRAN follows the above rule, and of course, ASM generated code can be written by the programmer to satisfy this rule too, so that this storage protection scheme could be expanded by the user in a very useful way. Input/Output routines, however, have tables containing non-zero words which are modified by execution of input-output instructions and therefore cannot have storage protection. An expansion of this scheme would require specifications to except these tables.

If extended use of storage protection is desired, the above scheme might be a basis for a COMMON standard. It is very probable that such an extension will become desirable in the future.

Assembly programs can alter protection bits only if a hardware key is set appropriately. FORTRAN programs do not have access directly to storage protection bits.

The current use of this feature by TSX is limited but probably sufficient in conjunction with the FORTRAN software mechanism to protect the system during careful operation. It does not do much to protect the users' programs themselves. The mechanism is definitely not foolproof.

1800 TSX LANGUAGES

INTRODUCTION

There has been a considerable amount of discussion and development work on languages for process control over the past few years. However, the purpose of this report is not to discuss language criteria but to list those languages supported by TSX.

LANGUAGES CONTAINED IN TSX SYSTEM

1. TSX Assembler Language
   See separate section on Assembler Language for the Committee's evaluation. NOTE: Machine language is not a subset of the TSX Assembler Language. It is only possible for the programmer to write in machine language by special manipulation. The Committee feels that this is a good decision.

2. TSX FORTRAN
   See separate section on FORTRAN for the Committee's evaluation.

The Committee has discussed the desirability of separating both FORTRAN and TSX assembler language from the TSX system and treating all languages uniformly as user programs; but it has reached no definite conclusion on this matter.

LANGUAGES WHICH ARE NOT CONTAINED IN THE TSX SYSTEM

There are several additional languages that will run as application programs under TSX

1. PROSPRO/1800

   PROSPRO stands for 1800 Process Supervisory Programming System and is a generator language for writing process programs. It will be distributed as a Type II program. The announcement letter states that PROSPRO is fully compatible with TSX.

2. COP/1800

   COP stands for 1800 Control Optimization Programming System. As with PROSPRO, this is an applications development rather than a systems development language.

3. DDC (Direct Digital Control)

   There is no information available about TSX support for DDC, although the Committee knows of several installations which are involved in this application. The number of questions would indicate wide interest in this topic and if this is so, a special discussion session will be arranged at a COMMON meeting.

CONCLUSIONS

The Committee received inquiries regarding various languages which have not been developed for the 1800. Those which received considerable interest were:

1) PL/1
2) List Processors
3) Decision Tables Languages
4) Report Generators and Data Handlers
5) Sort/Merge
6) Process Control Language

The Committee did not evaluate which of these are desirable and which are not, in an on-line operating system for data acquisition, teleprocessing and control. We would like to hear more discussion on this point by all the installations using 1800 TSX.

We urge that a project committee work to define a process control language.

1800 Process Systems Committee Report

An organizational meeting was held Monday, November 28, 1966. After lengthy discussion, two subcommittees were agreed upon, realizing that they could not be mutually exclusive and requiring considerable interaction.

A. Hardware Subcommittee responsibilities:

1. 1800 system hardware
2. Real time and process interface equipment
3. RPQ devices relative to process systems

Chairman: David Kraatz, 1001 Bedford Ave.,
North Kansas City, Missouri

B. Software Subcommittee responsibilities:

1. IBM supplied software
2. General interest user written routines
3. Cooperation with Process Application Committee

Chairman: Jay Ganatra, 660 South Blvd.,
Pontiac, Michigan

Initially, this committee plans to act as a collection or clearing area for recommended major revisions or additions to IBM 1800 systems. It is anticipated that suggestions will be collected and reviewed between COMMON meetings, that these and other ideas will be discussed by the 1800 Process Systems Committee in open sessions, and appropriate recommendations made through the proper channels to IBM.

For smaller problems, we will begin assemblying a directory of Process Control 1800 configurations so users, through this office, can contact similar installations.

As more 1800's are installed mutual problem areas will become apparent. The time spent at this COMMON meeting will give us the structure to handle these problems.

Post Office Box 3621

December 6, 1966

To: Current or future users of 1800 systems

At the COMMON meeting in New Orleans, an 1800 Process Systems Committee within the Systems Division 1800 project was formed. We want to get the following information from you in order to make future recommendations to IBM for future modifications to hardware and software systems:

A. Installation date

B. Description of the installation

1. Configuration
2. Primary applications

C. Comments on the TSX Preliminary Report

D. Any other comments

Please return your comments to me by January 31 so they may be reviewed and organized before the March meeting.

Lloyd W. Jones

Lloyd Jones
Chairman, Process Systems Committee
COMMON Users Group

January 4, 1967
San Jose
Process Control & Small Scientific Systems
464/062
4062

COMMON Meeting in New Orleans

Mr. G. W. Lohr

Since my presentation was a tutorial on subjects covered in the Reference Manual and Program Logic Manual for the 1130 Disk Monitor System, those publications can serve not only as an abstract of my talk, but also as detailed information.

Gene Lester

GL/sn

cc: Mr. W. P. Champagne

SYSTEMS DIVISION

SYMTRAN: The Addition of Algebraic Manipulative Capabilities to FORTRAN with Format

Mary Clo Carey

University of Southwestern Louisiana

Current address:
IBM Corporation
District 21 Test Center
2640 Canal Street
New Orleans, Louisiana 70122

529-5256 (area code 504)

Monday, November 28, 1966
1:30 - 3:00 P.M.
Session M-3.5

7 pages of text
3 pages of graphics

In the history of computers, numerous advances have been made in the field of numeric calculations. Problems which once required weeks, or even months, of human effort can now be solved in a matter of milliseconds. However, not all problems require a strictly numerical result. Many calculations cannot be performed until tedious algebraic operations have been completed; in some cases, the algebraic result is the end in itself. Production of this formal algebraic entity may require numerous invaluable, or even unavailable, man-years.

Since the computer can perform any conceivable numeric operation which can be rigorously delineated, why could the computer not perform purely algebraic operations? As early as 1954, ventures into the realm of algebraic manipulation and its related field, symbol manipulation, were begun. Numerous routines and systems were developed. These routines were specialized programs designed to perform one particular operation, or were subroutines to be called by a mainline program. All of the routines and systems developed were for large-scale digital computers.

Thus the decision was made to develop an algebraic manipulative system for a medium-size computer, the IBM 1620. The ability to formally manipulate algebraic expressions was to be included as an integral part of the new system. In the development of the system, five factors were taken into consideration:

1. the system should provide a tool to ease the burden of cumbersome algebraic manipulation;
2. the system should provide for execution of a variety of operations rather than being limited to the execution of a single operation;
3. the system should be adaptable to numeric as well as to algebraic problems;
4. the system should be easily learned; and
5. the system should be capable of extension.

Closer examination of the objectives revealed that four of the five objectives were already embodied in FORTRAN with Format. Because of the number of persons who have had some knowledge of FORTRAN programming, the decision was made to extend the FORTRAN with Format programming system.

SYMTRAN is thus a proper extension of the FORTRAN with Format programming system. All of the capabilities of FORTRAN with Format have been retained. The ability to formally manipulate algebraic expressions has been added. The algebraic expression may be any combination of variables, constants, and operation symbols which

adheres to certain rules. The SYMTRAN system provides for:

1. addition of algebraic expressions,
2. subtraction of one algebraic expression from another,
3. multiplication of one algebraic expression by another,
4. division of one algebraic expression by another, and
5. exponentiation of an expression to a positive fixed point power.

To accomplish the definition of the algebraic expressions and their formal manipulation, the SYMTRAN system consists of three main elements: (1) the SYMTRAN language, (2) the SYMTRAN compiler, and (3) the SYMTRAN subroutines. The system is strictly disk-oriented; disk storage is utilized for permanent storage of the compiler and the subroutines and for temporary storage of the compiled object program and the algebraic expressions.

The SYMTRAN language includes all valid FORTRAN with Format statements. In addition, formal algebraic expressions may be defined within the SYMTRAN language. Each formal algebraic expression is defined by being equated to a variable name, or more correctly an expression name, by which it can later be referenced. A definition statement takes the form,

e.g. @ONE = TWO + THREE

The "@" signifies that the statement is a formal definition. The entire expression to the right of the equal sign is the formal expression; the variable to the left of the equal sign (excluding "@") is the name of the expression.

Certain rules must be adhered to in the formation of formal algebraic expressions. Constants may be any valid fixed point or floating point numbers. Variables may be any combination of letters and numbers not to exceed five characters in length and must begin with a letter. Operation symbols allowable are +, -, *, /, and ** . All exponents must be constants. No parentheses may appear in an expression which is being formally defined. The restriction against parentheses eliminates the use of functions, subscripting, and divisors which are expressions. Mixed mode is allowed within a formal definition. The algebraic expression being defined must not exceed the permissible length of a FORTRAN with Format source statement, i.e., 72 characters.

All variables appearing within a formal expression definition are assumed to be basic, or atomic, variables. This restriction was motivated by four factors.

1. The expression being defined is to be stored in disk storage at compile time.
2. Variables which are to assume a numerical value do not have this value assigned until execution time.
3. Variables which are the names of expressions may in turn contain variables which are the names of expressions... . The chaining could continue indefinitely.
4. A variable which is the name of an expression may refer to an expression which contains the expression name, such as X = X + 5. Recursion would result and a basic variable could never be obtained.

Manipulation of the algebraic expressions defined is accomplished by the use of the usual FORTRAN arithmetic statements. Expressions may be combined by the operations of +, -, *, or /. The arithmetic statement may consist of any combination of variables, constants, and operation symbols subject to the FORTRAN regulations for formation of such statements. The variables may refer to a previously defined numeric value, may refer to an algebraic expression, or may be basic variables. If a variable is basic, i.e., it has not been assigned a value, it is treated as an expression consisting of a single element.

It is possible to test the result of operations on algebraic expressions for equality to zero, i.e., the resultant expression consisting of only the constant zero. To obtain such a test, the variable name assighed to the resultant expression must be a fixed point variable.

There are some restrictions on the use of expression names. No variable may appear more than once as the name of an algebraic expression that is being defined. In general it may be stated that a variable which names an expression may not be substituted for a variable whose numeric value is required. In particular, a name of an expression may not appear as (1) the argument of a functional subroutine, (2) the parameter of a DO loop, (3) the variable of a Computed GO TO, or (4) the power of any constant, variable, or expression. The results which would take place will vary in the different cases.

The SYMTRAN compiler, in addition to recognizing and compiling all valid FORTRAN with Format statements, must recognize and operate upon the formal expression definitions. All program statements are analyzed by the compiler program. If the statement is not determined to be an

arithmetic statement, the appropriate object coding is generated and compilation continues with the next source statement. A further discussion of the object coding will appear later.

Once an arithmetic statement is located, a check is made for "@" as the first character of the statement. If the "@" is not present, compilation proceeds as usual. If the "@" is present, it is deleted from the statement, the variable on the left side of the equal sign is collected and placed in the symbol table if necessary; the expression on the right side of the equal sign is compressed to remove any internal blanks. The expression is then ready to be stored onto disk storage.

A check is made of the Available Space List (AVSPLS) to determine where to store the expression. The AVSPLS is active from the beginning of compilation through the execution of the object program. At compile time, the AVSPLS contains the address of the next available sector within the five cylinders reserved for storage of expressions. The first check determines whether any sectors are available; if not, an error message is typed, the expression is not stored, and compilation continues with the next source statement. If there are available sectors, the expression is stored on the disk occupying one or two sectors. If the expression requires two sectors, a tag field is placed in the last six digits of the first sector giving the address of the second sector. Following the storage of the expression, the AVSPLS is updated to point to the next available sector for expression storage.

As each expression is recognized and stored, another important table is generated. This is the Table of Expression Names and Disk Storage Addresses(TEXNM). Each expression name and the sector address of its first sector are placed into the table. At the end of compilation the disk addresses are placed into the symbol table for use by the object program.

During the compilation of most source statements object coding is generated. This object code uses the same format as FORTRAN with Format. The object program card images are stored in consecutive sectors on the disk. While this is not the most efficient utilization of disk storage, it was felt that a modification of the object program format should be postponed until a later date. Once the object program has been compiled and stored on the disk, the program loader is called to bring the program from disk and load it into core along with the symbol table and the subroutines. At load time, the symbol table is completed; constants are stored, branches to numbered statements are completed, and sector

addresses are substituted for expression names.

During the execution of a FORTRAN object program, the majority of the operations are performed through subroutines. The same is true of the SYMTRAN system. The majority of the modifications and additions required to implement SYMTRAN are in the SYMTRAN subroutines. The SYMTRAN subroutines include the entirety of the FORTRAN with Format subroutines plus eleven additional subroutines to perform algebraic manipulations. The subroutines unique to the SYMTRAN system are:

1. SYMADD to add algebraic expressions,
2. SYMSUB to subtract one algebraic expression from another,
3. SIMPFY to simplify algebraic expressions,
4. SYMULT to multiply algebraic expressions,
5. SYMDIV to divide one algebraic expression by another,
6. SSTOR to store an expression on the disk,
7. SFIND to retrieve an expression from disk storage,
8. SYMIN to input an expression during execution of the object program,
9. SYMOUT to output an expression,
10. EXPAND to convert input form to internal representation, and
11. SYMSUP to interrogate operands and branch to the appropriate subroutine.

The five subroutines which perform the actual algebraic operations on the expressions adhere to the rules of algebra.

e.g. A + 2*A will yield 3*A as a result
X*X**3 will yield X**4 as a result

For these subroutines to operate upon the expressions, it was deemed necessary to have all expressions adhere to a fixed format. Before any algebraic operations are performed a check is made to determine if the expression is in the internal representation. If not, the expression is converted by the EXPAND routine. The basic unit of the expression is taken to be the term. The expression is divided into terms. Each term is then converted to the form: coefficient, variable-1, exponent-1, variable-2, exponent-2,.... All Coefficients and exponents are represented as floating point numbers, coefficients and exponents of one being inserted where necessary; variable names are represented as five character(10 digit) alphameric fields, left-justified; following the term is a record mark. Following the last term of the expression is an additional record mark.

e.g. ORIGINAL EXPRESSION
5*X + 18*Y**3/X - Y

INPUT FORM

$$\bar{7}5\bar{1}4\bar{6}7\bar{1}0\bar{7}1\bar{7}8\bar{1}4\bar{6}8\bar{1}4\bar{1}4\bar{7}3\bar{2}1\bar{6}7\bar{2}0\bar{6}8\bar{\neq}$$

INTERNAL REPRESENTATION

$$\bar{5}0000000\bar{0}1\bar{6}700000000\bar{1}0000000\bar{0}1\neq\bar{1}8000000\bar{0}2$$

$$\bar{6}800000000\bar{3}0000000\bar{0}1\bar{6}70000000\bar{0}\bar{1}000000\bar{0}\bar{0}1\neq$$

$$\bar{1}000000\bar{0}\bar{0}1\bar{6}800000000\bar{1}0000000\bar{0}1\neq\neq$$

Note: Variables which are divisors are represented with negative ~~coefficients.~~ exponents.

The results of all algebraic subroutines are stored onto the disk in internal representation. The SSTOR subroutine utilizes the symbol table to locate the name assigned to the resultant expression. If the name already designates an algebraic expression, the previous expression will be deleted from the disk and the addresses of the sectors occupied by the expression restored to the AVSPLS. SSTOR then determines the first available sector from AVSPLS, stores the first 94 digits of the expression into that sector, and inserts the sector address into the location in the symbol table designated for the variable name. In the event that an expression exceeds 94 digits in length, the last six digits of the sector are used as a tag to point to the next sector occupied by the expression. After deletions and additions in the storage area on the disk, adjacent sectors may not be available, thus the need for the tag field. The tag field for the last sector of an expression consists of six zeros.

Input and output of algebraic expressions present unique difficulties in comparison with input and output of numeric values. Because of the innumerable variations in length and complexity of expressions, it was determined that input and, more particularly, output should follow a free format. A FORMAT statement should not be essential in this case. However to preserve consistency and avoid confusion, every input-output statement referencing an algebraic expression must have an associated FORMAT statement. The FORMAT statement may be a dummy statement. Algebraic expression names and numeric variable names may not be mixed in an input-output list. Expression input follows the same regulations as the formation of formal expressions in the source language. Expression output will follow an output form determined by the SYMTRAN system. The SYMOUT routine will convert the expression, which will be in internal representation to output form. All coefficients and exponents having a value of 1 (except a constant 1) are eliminated

from the output; exponents of zero cause the associated variables to be eliminated; coefficients of zero cause the associated terms to be eliminated. All remaining coefficients and exponents are output in F format if the exponent is in the range -8 to +8, else the output is in E format. Operation symbols are inserted where required. If an expression exceeds one output record in length, successive records are utilized until output is complete. There is no limit to the length of an output expression.

The SYMTRAN system as it exists today is by no means complete. Further extension of the SYMTRAN system is always possible. Some of the possible extensions are:

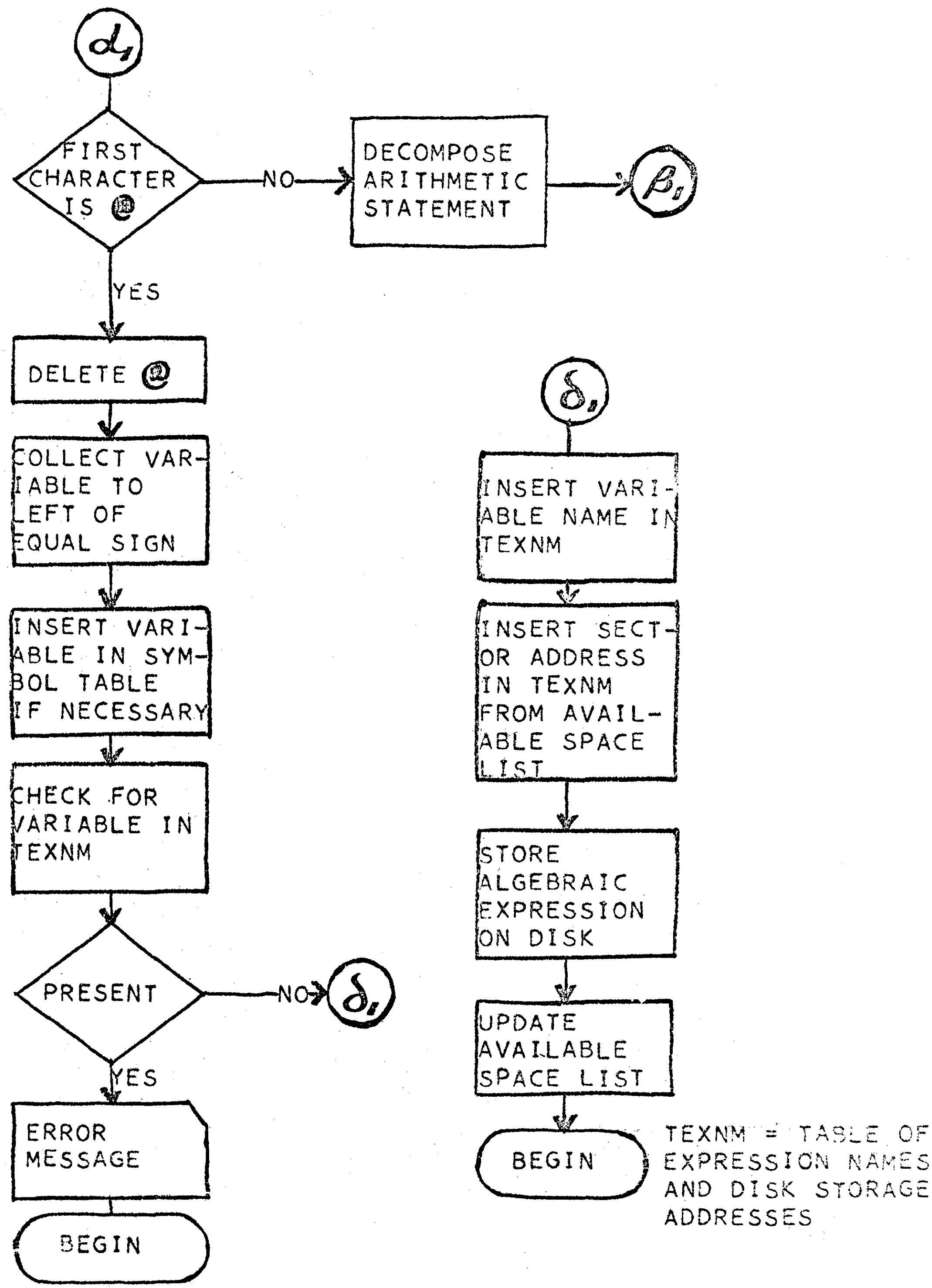
1. utilization of more than one disk drive to allow for more expression storage;
2. modification of the system to support more input-output devices, such as a 1443 printer;
3. modification of the arithmetic subroutines, particularly EXPAND, to allow for greater complexity of formal expressions;
4. addition of functional subroutines to operate on algebraic expressions; and
5. addition of routines to perform such functions as differentiation and integration of formal expressions.

In reconsidering the original factors taken into consideration in the development of the system, it will be realized that the primary objectives in the development of the SYMTRAN system have been attained.

Although the SYMTRAN system is not complete, it does represent one venture into the realm of algebraic manipulation on a medium-size computer. But more significant is the integrity of SYMTRAN. SYMTRAN is not a collection or processors or various routines each modifying the original input for future operations; rather SYMTRAN is a complete <u>system</u>.

SYMTRAN COMPILER

ARITHMETIC STATEMENT
DECOMPOSITION



```
1620 SYMTRAN SYSTEM, MAY 1966

ENTER SOURCE PROGRAM, PUSH START
27000 C     SYMTRAN TEST PROGRAM
27000       Z = A*B+C
27048       @A = X+Y
27048       @B =X-Y
27048       @C= 2*X**2+3*Y-2
27048       TYPE 999,Z
27072    10 STOP
27120   999 FORMAT(E15.3)
27142       END

PROG SW1 ON FOR SYMBOL TABLE, PUSH START
39879 Z
39869 A
39859 B
39849 000
39839 C
39829 0999
39819 0999
39809 0010


PROCESSING COMPLETE
3.00 X**2.00 -Y**2.00 +3.00 Y-2.00
STOP
```

```
1620 SYMTRAN SYSTEM, MAY 1966

ENTER SOURCE PROGRAM, PUSH START
27000 C      SYMTRAN TEST PROGRAM
27000        Z=A*B*C+A*B-C
27108        @A = X+Y
27108        @B =X-Y
27108        @C= 2*X**2+3*Y-2
27108  7777  TYPE 999,Z
27132   999  FORMAT(//31HSUCCESS, NOW TRY SOMETHING ELSE)
27228    10  STOP
27276        END

PROG SW1 ON FOR SYMBOL TABLE, PUSH START
39879 Z
39869 A
39859 B
39849 000
39839 C
39829 001
39819 7777
39809 0999
39799 0999
39789 0010


PROCESSING COMPLETE


SUCCESS, NOW TRY SOMETHING ELSE
3.00 X**2.00 +Y**2.00 +2.00 X**4.00 -2.00 Y**2.00 X**2.00
 +3.00 Y*X**2.00
3.00 Y**3.00 +3.00 Y-2.00
STOP
```

COMMON

New Orleans, Louisiana

1620 Systems Papers

A Processor For Both SPS And FORTRAN

Guy George Jr.
University of Southwestern Louisiana

Box 382 U.S.L.
Lafayette, Louisiana 70501

M3.5
Monday
November 28,1966
1:30-3:00 P.M.

Text 5
Graphics 5

AUSYM
An Automatic Programming Language With Symbolic Capabilities
For The IBM 1620

Introduction A professional programmer encounters problems which vary in complexity and subject matter. Each type of artifical programming language has a set of attributes which make it desirable or undesirable for a given problem. Often the programmer is required to sacrafice some of the favorable attributes of one language in order to utilize another language in the solution of the problem. It would be advantageous to have languages which possess the favorable characteristics of several of these artifical languages.

Two major forms of artificial programming languages are automatic languages and symbolic languages. Automatic programming languages such as FORTRAN with FORMAT for the IBM 1620 remove the necessity for the programmer to be aware of the basic operations being performed by a particular computer. The programmer does not have to manually assign and account for variable storage or instruction addresses. Not only is he not required to perform many tasks necessary in a symbolic programming language such as SPS for the IBM 1620, he cannot perform the tasks if he wishes to. This is of little consequence to the nonprofessional programmer, but it does remove the flexibility of SPS available to a professional programmer.

Although there exists a multitude of languages and language subsets, few possess the abilities of an automatic language and yet retain access to a flexible machine oriented symbolic language. There seems to be no such flexible system available for the IBM 1620. Therefore, the problem stated in simple terms is that a sufficient number of applications are in need of such a flexible system.

Available Capabilities The only integration of FORTRAN and SPS for the 1620 now in use is the ability of a FORTRAN language to accept a subroutine written in symbolic coding. This subroutine can then be called by the mainline FORTRAN program in various manners. Assembly of the SPS statements is completely external to the FORTRAN statement translation. The machine language coding generated by the SPS statements will not be mixed in with the coding generated by the FORTRAN compiler. The FORTRAN symbol table is not available to the SPS written routine; and to share any data, extensive use of the COMMON statement must be employed.

The Proposed System It would be convenient to have a fully integrated system of FORTRAN and SPS. The integrated system could have a single symbol table. With one symbol table, it would not be necessary to declare a FORTRAN variable to be the same as a variable in SPS. Transfers of control between differently coded segments could be done without restriction to the type of transfer. The system should not have interruptions in the coding between the different segments of the program, but should

flow continuously from one section to another. The system should be fast and economical enough in the use of core to allow a symbol table and object program of reasonable size.

AUSYM for an AUtomatic programming language with SYMbolic capabilities is such a system. AUSYM is FORTRAN oriented and on the level of FORTRAN with FORMAT for the IBM 1620 with additional capabilities. Since the system is FORTRAN oriented, variables are declared in the FORTRAN manner by mentioning them in the source program. Without a need to declare available storage locations, the SPS capabilities consist of imperative type statements only. The coding forms are unchanged to preserve familiarity, and the system is written to be included on the Monitor I system for ease of operation.

Machine Requirements Once a language has been defined it is necessary to be concerned with a machine in any considerations of implementation. The machine requirements for AUSYM are as follows: an IBM 1620 CPU with at least one 1311 disc drive, a 1622 card read and punch unit, a Monitor I system, indirect addressing, and the divide command. The system can also make use of the 1443 printer, 1621 paper tape read and punch; and since the programmer has direct control in symbolic portions, the 1627 plotter can be employed through programming. The machine must have at least 40K memory and may have 60K memory without modifications.

Source Program Division As in most variations of FORTRAN, AUSYM requires a specific order in the order in the appearance of certain statements. The order requirements divide an AUSYM source program into two distinct segments. The first segment of an AUSYM source program must contain, in any order, all FORMAT and DIMENSION statements. Comment cards may be contained in this segment of a source program. This segment is ended with the first card which contains an S or an F in column one.

The second segment of an AUSYM source program, signaled by the use of an S or an F in column one, is the mainline program. In this segment it is possible to change from a FORTRAN type coding to an SPS type coding and vice versa.

Changing Program Languages In the second part of an AUSYM program it is possible to change at will between FORTRAN coding and SPS coding. When an S or an F appears in column one it indicates that it and any following cards with a blank or a C in column one will require the same type translation procedure. An S or an F controls the type of processing until another F or S appears in column one.

Construction of Variable Names and Labels Variable names and labels have identical restrictions for their construction. They may be no longer than five characters and must start with an element of the alphabet. The other four digits may come from the alphabet or the ten decimal digits. After the first letter the characters can be arranged in any order desired. Imbedded blanks are not permitted in labels or statement numbers, but will be allowed at any other time without effecting the compilation.

Symbolic Capabilities A symbolic language source statement is either a macro or a regular source statement depending on the number of object instructions generated by the source statement. The regular symbolic statement is the reason for the flexibility of SPS. One SPS statement generates a single machine language instruction.

The entire command repetoire of the 1620 is available in SPS statements of this type, which are generally referred to as imperative statements. The imperative statement structure for AUSYM is identical to that of an SPS statement. Each imperative statement may contain a label, must contain an operation mnemonic, and may or may not contain a P operand, Q operand, flag operand, or comment operand. An entire list of the imperatives available in AUSYM is given below. Also included, is a list of the macros available in AUSYM. They utilize the capabilities of the FORTRAN subroutines.

An operand may contain a subscripted variable of either one or two dimensions. The only restriction is that the subscripts are constants. If variable subscripts were allowed, it would be necessary to generate more than one machine language instruction per AUSYM symbolic statement.

Automatic Capabilities The FORTRAN capabilities for the AUSYM system are given below. The range for fixed and floating point values is the same as the range for FORTRAN with FORMAT for the 1620. Variable names are limited as described above. The major differences can be noted in the listing of capabilities. The I/O commands are similar to those available in FORTRAN IV. There is a PAUSE V and EXIT command. An important variation from FORTRAN with FORMAT is the use of the address variable. This is set up with an ASSIGN statement. Except for the statement number used with a DO statement, any statement number in a FORTRAN type statement may be an address variable. The exception is because AUSYM requires a matching CONTINUE statement for each DO statement. The rules governing statement construction in FORTRAN with FORMAT are applicable to AUSYM.

Branching Between Codings If the system is to realize any significant flexibility it must be capable of transferring control from a section coded in AUSYM automatic to a section coded in AUSYM symbolic and vice versa. At first this appears to be a simple matter. However, it is important that the programmer realize the manner in which branches are effected between codings.

When branching from automatic into symbolic coding the programmer must make use of an SPS label used as an address variable. When the symbol table is arranged for object time, labels in SPS will be initialized to contain the object time address of the statement the label refers to. This is what is set up when an ASSIGN statement is used to define an address variable. There is no extra action on the part of the programmer. It is as though the label was a statement number.

In considering branches in the symbolic language, the position

branched to may be an actual, symbolic or asterisk address. When a branch is needed into an automatic language section, no label is available since the statements are distinguised with statement numbers. If a statement number is used as the label to branch to, the compiler has no way of distinguishing the statement number from an absolute address. For this reason, to branch from symbolic to automatic AUSYM coding requires some action on the programmer's part. The branch should be to an address variable used as an indirect address.

System Constituents Viewed as a whole, there are three parts to the AUSYM system which must be placed on Monitor I. First, there are the in-core subroutines, which are an integral part of any object program produced by the compiler. Secondly, there are the relocatable subroutines, any one of which is added to the object program only if it is called upon. Finally, there is the AUSYM compiler whose job is to generate an object code from the various input source statements.

Subroutines--In Core and Relocatable The in-core subroutines used by an AUSYM object program are the same as the subroutines provided with IBM's FORTRAN with FORMAT for the 1620. Four very short routines have been added which provide AUSYM with capabilities not available in FORTRAN with FORMAT. These are for the operation of the PAUSE to type up to a five digit word, for the EXIT to return control to the Monitor, and for the reread (D=0 for READ) to rescan an input record.

The relocatable subroutines differ from those provided by FORTRAN with FORMAT in two ways. First, the arguments address is sent to the subroutine instead of sending the address to the symbol table. Secondly, to allow for relocatability by AUSYM all statements were made to generate object instructions of uniform length. The reason for this is the method for relocating and loading the relocatable subroutines.

The Compiler With the source program divided as previously mentioned, it is not necessary to retain the entire compiler in memory. Once the FORMAT information has been processed there is no need for this portion. In a similar fashion, once the mainline program has been translated, its translator is no longer needed in memory. Continuing in this fashion, the compiler is written in definite segements which are overlayed once their functions are completed. The various segments of the compiler are referred to as phases and each phase has a number of tasks to fulfill.

The design of the phases was such to insure the use of as little core as possible during any one phase. Using this approach, there is a maximum amount of space allotted to the symbol table. With the system on disc the time for overlaying is at a minimum.

Brief Description of Phases. Phase one defines constants to be used throughout the compilation process as well as small routines to be used repeatedly. Some of the routines included are: PRINT, to take care of listings; ERROR, to give error messages; SMOT, to bread a symbol from a certain location; etc. Once the size of memory has been determined, local control cards are processed. The symbol table is initialized and processing begins. This phase continues until the first

S or F is sensed in column one of a card. All FORMAT and DIMENSION statements must be translated in phase one. Prior to exit, the initial order of the symbol table is established with a sort.

Phase two is the major phase of the entire compiler. The symbol table search and subscript determination routines are entered with phase two. If an SPS statement is encountered, pass one is conducted to reserve space in the object program and to enter labels into the symbol table. The statements are recorded for complete translation in phase three. FORTRAN statements are analyzed for category and are either translated as arithmetic statements or as category two statements.

Phase three is designed to conclude the production of an object program. This is pass two SPS processing. The records are read in blocks and special routines are used to locate the storage location on the disc for the object code generated by a statement. If any new variables are encountered, they are entered into the symbol table. Prior to exit, the compile time symbol table is written onto disc and error indicators are checked to determine whether to proceed or not. A vector is constructed to indicate which relocatable subroutines, if any, are required.

Phase four actually consists of several overlaying program segments. If relocatable subroutines are required, they are all placed in memory; and as a subroutine's relocation is completed, the subroutine is added to the object code on disc. Information to the operator, such as object starting location and symbol table starting location, is typed. The FORMAT information and the object program are read into memory. Then, the compile time symbol table is expanded into memory. Symbol table listings are produced. The only remaining item needed for execution is the in-core subroutines. These are loaded, and control is transferred to the object program.

Conclusion Included below is an example program which demonstrates most of the capabilities of the AUSYM system. In a thesis by the author, a description of the internal workings of the program phases is given. This is done in a fashion to aid a reader in following the source listing. Topics included are: Storing Format Information, The Symbol Table, The Symbol Table Search, Subscripted Arrays, Pass I Symbolic Processing, etc. The thesis also includes appendices for source listings, program compilation, and sample runs.

ARITHMETICS

A	AM	S	SM	M	MM	LD
LDM	D	DM	C	CM		

INTERNAL DATA TRANSMISSION

TD	TDM	TF	TFM	TR	MF	TNS
TNF						

BRANCHING INSTRUCTIONS

B	BNF	BNR	BD	BT	BTM	BB
BI	BC1	BC2	BC3	BC4	BL	BH
BP	BE	BZ	BNL	BNN	BV	BXV
BNI	BNC1	BNC2	BNC3	BNC4	BNH	BNP
BNE	BNZ	BL	BN	BNV	BNXV	BA
BNA						

INPUT-OUTPUT

RN	RNTY	RNPT	RNCD	WN	WNTY	WNPT
WNCD	DN	DNTY	DNPT	DNCD	RATY	RA
RAPT	RACD	WA	WATY	WAPT	WACD	PRN
PRNS	PRA	PRAS	PRD	PRDS	SK	RDN
WDN	CDN	RDGN	WDGN	CDGN	RTN	WTN
CTN	RTGN	WTGN	CTGN			

MISCELLANEOUS

K	RCTY	TBTY	SPTY	BKTY	SKIP	SF
CF	H	NOP				

MACRO INSTRUCTIONS

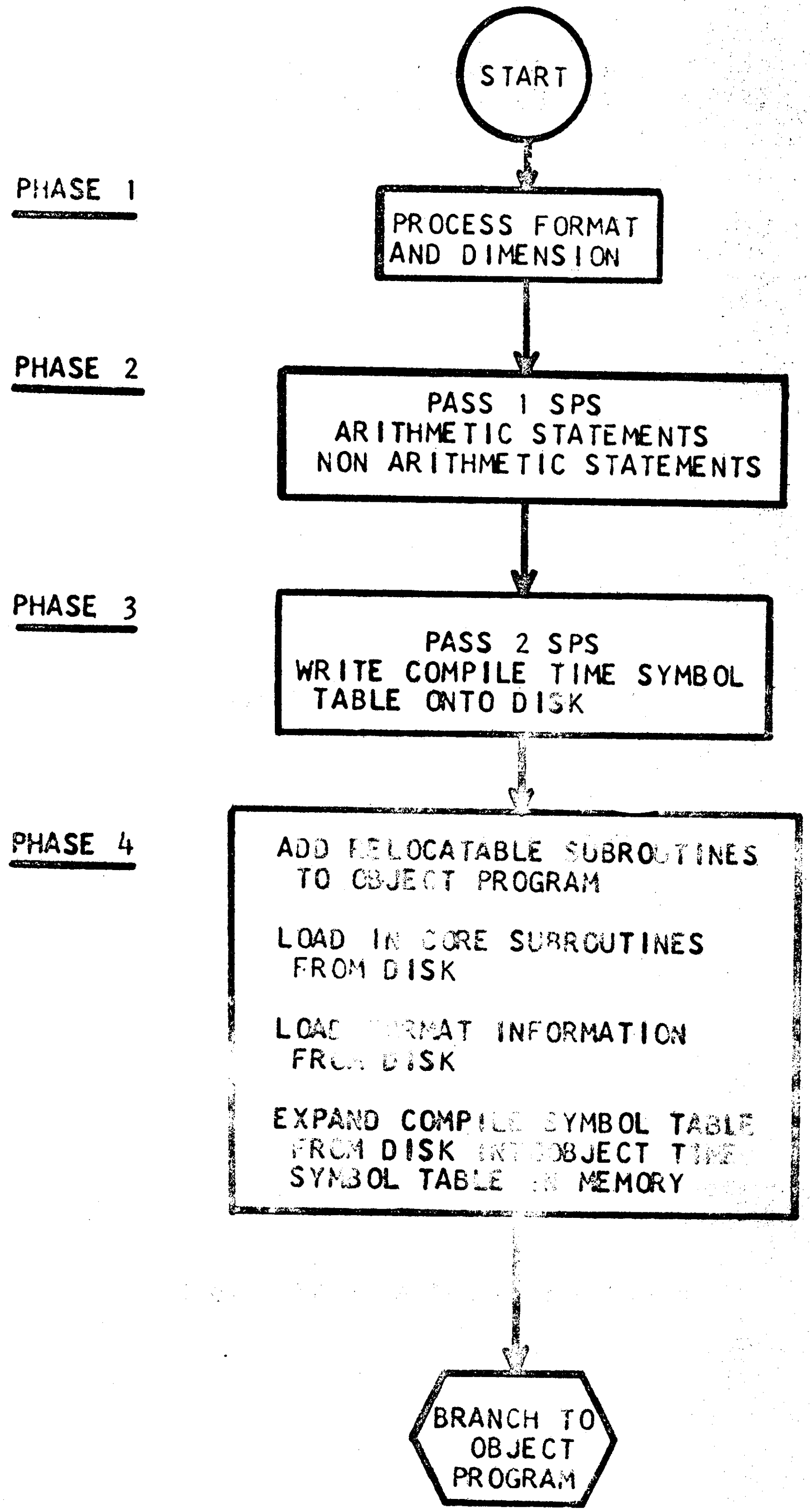
FA	FS	FM	FD	FLT	FIX	FSQR
FSIN	FCOS	FATN	FEX	FLN		

Symbolic Capabilities

Automatic Capabilities

```
DIMENSION V1(N1),V2(N2),V3(N3),....

ASSIGN (N) TO (V)

GO TO N

GO TO (N1,N2,N3,...),V

IF (SENSE SWITCH N) N1,N2

IF (EXP) N1,N2,N3

DO N V1=V2,V3,V4

N CONTINUE

PAUSE V

EXIT

END

      READ (D,N),V1,V2,V3,...
**  D=0 REREAD
    D=1 READ FROM TYPEWRITER
    D=2 READ A CARD
    D=3 READ FROM PAPER TAPE

      WRITE (D,N),V1,V2,V3,...
**  D=0 PRINT
    D=1 PUNCH CARD
    D=2 TYPE
    D=3 PUNCH PAPER TAPE

    N FORMAT (SPECIFICATION)
```

Compiler Phases



```
C  AUSYM SAMPLE PROGRAM TO SORT 500 VALUES USING
C  A BUBBLE SORT.  THE INPUT VALUES MAY BE IN ONE
C  OF TWO LOCATIONS ON A CARD
       DIMENSION X(500)
   100 FORMAT(I2)
   101 FORMAT(10X,F10.3)
   102 FORMAT(20X,F10.3)
   103 FORMAT(16H ORDERED NUMBERS)
   104 FORMAT (1X,F10.3)
   105 FORMAT(I3)
F      ASSIGN (1)TO (LOOP)
       ASSIGN (8) TO (BOTEM)
       READ (2,105) MAX
       MAX=MAX-1
       DO 1 I=0,MAX
     5 READ (2,100) IND
       IF (IND-2) 2,3,4
     2 READ (0,101) Y
       GO TO FVAL
     3 READ (0,102) Y
       GO TO FVAL
C  IF THE INDICATOR IS GREATER THAN 2, IT IS ILLEGAL
     4 PAUSE ILGAL
       GO TO 5
S     FVAL  NOP *+60
            TDM *-11,9
C  THE FIRST READ IS MOVED INTO POSITION
            TF   X(500),Y
            TF   X(500)-2,Y-2
            B    LOOP,,6
F      M=500-I
C  THE BUBBLE SORT IS STARTED
       DO 7 J=M,500
       IF(X(J)-Y)6,6,7
     7 CONTINUE
     6 J=J-1
S           SM   M,1,8
            C    J,M
            BE   BOTEM,,6
            MM   J,10
            AM   99,X(1)-10
C  SAVE CONTAINS THE DIGIT REPLACED WITH A RECORD MARK
            TD   SAVE,99,11
            TD   99,400,6
            SM   99,10
            SF   95
```

```
C   SAVE-1 CONTAINS THE ADDRESS OF THE LOCATION THAT THE
C   RECORD MARK WILL BE IN AFTER THE TRANSMIT RECORD
             TF   SAVE-1,99
             MM   M,10
             AM   99,X(1)-19
             SF   95
             TF   *+30,99
             AM   99,10
C   THE MOVE DOWN TO MAKE ROOM FOR THE IN COMING ELEMENT IS
C   ACCOMPLISHED WITH A TRANSMIT RECORD
             TR   ,99,11
             TD   SAVE-1,SAVE,6
F     8 X(J)=Y
      1 CONTINUE
        WRITE (0,103)
        M=500-MAX
C   THE VALUES ARE WRITTEN IN AN ORDERED FASHION
        DO 12 J=M,500
        WRITE (0,104) X(J)
     12 CONTINUE
        PAUSE END
        EXIT
        END
```

PROGRAM ABSTRACT

TITLE: University of Mississippi Test Scoring Program Revised (UMTS-R)

SUBJECT CLASSIFICATION: 13.0.031

AUTHOR: Richard D. Ross

DIRECT INQUIRIES TO: Richard D. Ross, Director
Computer Center, Carrier 103
University of Mississippi
University, Mississippi
Phone: area code 601-232-8368

DESCRIPTION: UMTS-R is a flexible means of scoring objective exams taken on mark sense cards. It features a card input and card output or 1443 printer output if printer is available. A numerical grade for each student is published along with a grade distribution (with mean and standard deviation) and an exam analysis--indicating how many choices per question. UMTS-R has the following additional features:

(1)Allows multiplicity of correct answers.
(2)Allows each answer to be weighted with a weight value from 1 to 5.
**(3)Allows omitted question numbers to be punched following the student's grade card.
**(4)Allows question numbers incorrectly answered to be punched following the student's grade card.
**(5)Allows weight factor to be punched beside each question number on exam analysis output.
**(6)Allows identification to be punched in columns 76-80 of student's grade card.
(7)Allows additional identification to be punched in columns 01-30 of first header card.
(8)Allows blank cards to be read at any time.
(9)Allows all key cards, alternate key cards, and weight cards to be read in random order.
(10)Allows student answer cards to be read in random order.
(11)Allows batch processing.
**(12)Allows blank card to be punched following student's grade card.
**(13)Permits I. D. number to be replaced by sequence number.
*(14)Permits grading of a test without a card number.
*(15)Permits grading of a test without a name.
*(16)Permits grading of a test without a section number.
*(17)Permits grading of a test without a course number.

*(18)Allows variable length I. D. numbers (02 to 11 columns).
*(19)Allows variable length name (01 to 20 columns).
*(20)Allows variable length section numbers (01 to 03 columns).
*(21)Allows variable length course numbers (01 to 06 columns).
(22)Replaces invalid characters with blanks, thus eliminating check stops.
*(23)All double punched columns may be considered as omitted questions.

*Specified by "$DEFINE" card
**Specified by "$" Control Card

Another important feature of UMTS-R is the speed of grading each student's exam. Given below is the speed of grading:

Number of Questions	Time in Seconds
50	.93
100	1.20
150	1.55
200	1.90
300	2.60
400	3.30
500	4.00

RESTRICTIONS/RANGE: No special instructions are required although TNF and/or Direct Divide can be used on computers that have these capabilities.

EQUIPMENT SPECIFICATIONS: Memory 20K, 1622. No special instructions are required but indirect addressing is required. Memory 40K can be used if available and also the 1443 printer can be used if available.

ADDITIONAL REMARKS: SPS language processed by AFIT SPS (Program Number 01.1.023). Fixed point and non-relocatable. Running time is 1 1/2 seconds for 150 questions and 4 seconds for 500 questions. This program is written to handle 25 mark sense columns and split them in half putting questions 1-25 in the 9 to 5 positions and questions 26-50 in the 4 to 0 positions. Each answer must be punched in a separate column for the computer so that a half-after-four time pick up was added to the mark sense punch to pick up coselectors and punch each question in a separate column. However, this is variable and may be defined by the $DEFINE card.

DESCRIPTION OF PROGRAM

RESULTS AND METHODS: UMTS-R is designed to grade objective examinations for a maximum of 150 5-choice questions for 20K machines and 500 5-choice questions for 40K machines for 999 to 99999 students per exam depending upon the specification of the $DEFINE card. The program sets up a 10-digit constant for each question to be graded. This 10-digit constant is initialized to flag zeros ($\bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}$) to represent the answers (EDCBAEDCBA). The only possible answer for any question is a 0,1,2,...8 or 9. Hence, this constant can be set up to grade any question on any test.

To illustrate how this constant can be used, we will assume that we are on question 1 and there is a possibility of two answers that will be correct (A and C) and this question has a weight of 4 on the test. The 10-digit constant for question number 1 will then have the form $\bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ \bar{0}\ 4\ \bar{0}\ 4$. Initially the weight of each question is assumed to be 1 and at any time a weight card is read in the card number determines which questions are to be weighted and if question number 1 was given a weight of 4, the computer would check all positions of the 10-digit constant for question 1 and change all non-flagged digits to the proper weight. If a question is to be left blank, the computer will fill in for the 10-digit constant 10 flagged record marks.

The address of the first position of each 10-digit constant has the address of XXXX0. Hence, by transmitting a digit to the zero position of the address, the computer can readily determine if the answer is correct, incorrect, or to be omitted. This check is determined by a non-flagged digit, flagged digit, or a record mark respectively.

All cards are read alphabetically and the first position of each alphabetic answer is checked for a digit to determine if the student has omitted the question. The program will accept double-punched columns for answers only if they are specified by $DEFINE card.

As each student's questions are graded, an exam analysis table is up-dated, and as the student's score card is punched, a grade distribution table is up-dated. The output that is obtained from this program is the student's score card, the grade distribution with accumulative frequencies and percentiles along with the number of tests given, mean, and standard deviation. This is followed by the table of choices made for each question which includes an asterisk beside the correct choice or choices for each question. For multiple section exams or multiple-course exams, the above information may be punched for each section, each course, and all courses totalled together.

The answer cards have appropriate header cards included so that they may be printed with an 80-80 407 board. A 9-punch is placed in column 1 of a single card for each group to permit skipping to a new page for each group (wiring first reading column 1 through correct coselectors to carriage skip on a 9, and also to non-print). If the 1443 printer is specified then each line is printed accordingly.

$DEFINE CARD: The $DEFINE card is used to completely define a 1620 computer system and a test card layout form. The $DEFINE card is the first card read following the object deck and given below is the format of the card.

COLUMNS	DATA
01-10	$DEFINEbbb
11-15	Memory Size (20000, 40000, or 60000)
16	"0" or blank for card output, "1" for 1443 printer output
17-19	Maximum number of questions to be graded (SIZE)
20-21	Maximum number of cards needed for "SIZE" number of questions (NCARD)
22-23	Number of digits for all totals (03, 04, or 05). If 03, then only 999 students may be graded at one time, if 04 then 9999, and if 05 then 99999 students may be graded. (LT)
25-34	These ten columns are used to define what punches represent the answers A, B, C, D, and E. In some cases an "A" can be represented by a "9" punch or a "4" punch, a "B" by an "8" and "3" punch,---, "E" by a "5" and "0" punch. Then the ten columns would contain 9876543210 to represent ABCDEABCDE. If "A" is represented only by a "5" punch, "B" by a "4" punch,---, and "E" by a "1" punch, then the constant would be 5432154321 to represent ABCDEABCDE
35	"1" to change weight factor output on exam analysis from B, C, D, and E to 2, 3, 4, and 5. Otherwise a "0" or blank.

36-37	Beginning column for I. D. number
38-39	Ending column for I. D. number
(a)40-41	Beginning column for name
(a)42-43	Ending column for name
(b)44-45	Beginning column for section number
(b)46-47	Ending column for section number
(c)48-49	Beginning columns for course number
(c)50-51	Ending column for course number
(d)52-53	Beginning column for card number
(d)54-55	Ending column for card number
56-57	Beginning column for answers to test
58-59	Ending column for answers to test. Total number of questions (NQ) is determined by the beginning and ending columns of answers to test.
61-70	These ten columns are used to define legal double punched columns (0,1,2,...9). The first digit of each alphanumeric test answer is checked and normally only answers of the form 7X are used, otherwise the answer is considered as being omitted. In this case columns 61-70 would contain 0000000100 to represent 0123456789 Suppose that forms 5X, 6X, and 7X were to be considered valid, then columns 61-70 would contain 0000011100 to represent 0123456789 where 0123456789 represents the first digit of each alphanumeric answer. If columns 61-70 are blank then form 7X will be assumed.
73	"0" or blank if Direct Divide is available, otherwise, a "1"
74	"0" or blank if TNF instruction is available, otherwise, a "1"

75	"0" or blank if name is available, otherwise a "1"
76	"0" or blank if section number is available, otherwise a "1"
77	"0" or blank if course number is available, otherwise a "1"
78	"0" if card numbers are on all test cards or "1" if test cards have no card number. If no card number is available then alternate keys and weight cards will not be accepted.
80	"1" to skip typing of message XXXXX UNUSED CODE Otherwise a "0" or blank

(a) May be left blank if "1" specified in column 75
(b) May be left blank if "1" specified in column 76
(c) May be left blank if "1" specified in column 77
(d) May be left blank if "1" specified in column 78

No error messages will be typed if there is not enough memory, but the computer will type the number of unused core positions in the form $\bar{X}$XXXX UNUSED CORE. A negative number indicates overlap of memory.

There is a limit when specifying card columns for the following:

ITEM	Minimum	and	Maximum	Lengths
I. D. Number	02	to	11	columns
Name	01	to	20	columns
Section	01	to	03	columns
Course	01	to	06	columns
Card Number	01	to	01	columns
Test Answers	01	to	75	columns

The I. D. number and/or course number and/or section number and/or name may occupy the same columns on the card, but they must not overlap. The card columns for the I. D. number, name, etc., may be anywhere on a card, for example

I. D.	could be in columns 07-14
Name	could be in columns 02-06
Section	could be in columns 20-21
Course	could be in columns 15-19
Card Number	could be in columns 01-01
and Test Answers	could be in columns 25-80

depending on a particular test card format.

To determine the number of core positions used by UMTS-R, the following formula may be used

CORE = 13059 + SIZE(10+5*LT*2)+3 + 105*LT*2+2+NCARD*NQ*2

where	SIZE	= Maximum number of questions
	LT	= Number of digits in all total constants
	NCARD	= Maximum number of cards
	NQ	= Number of questions per card

INPUT

1. Program Deck
2. $DEFINE Card
3. Control Card

Card Columns	Data
01	All control cards contain a $ in column "1".
02	Number of cards per student. If there are ten (10) cards per student then column 2 will contain a "0".
03-05	Number of questions on the exam.
06-08	Number of questions not to be graded (this includes only those questions properly left blank).
09	"0" or blank if the grade distribution and exam analysis by section is desired. Otherwise, a "1".
10	"1" if the grade distribution by section is to be omitted. Otherwise, a "0" or blank.
11	"1" if the exam analysis by section is to be deleted. Otherwise, a "0" or blank.
12	"1" if the grade distribution and exam analysis by course is desired. Otherwise a zero or blank.
13	"1" if the grade distribution by course is to be deleted. Otherwise, a "0" or blank.
14	"1" if the exam analysis by course is to be deleted. Otherwise, a "0" or blank.
15	"1" if the grade distribution and exam analysis is desired on last card indicator. Otherwise, a "0" or blank.
16	"1" of grade distribution on last card indicator is to be deleted. Otherwise, a "0" or blank.

17	"1" if exam analysis on last card indicator is to be deleted. Otherwise, a "0" or blank.
18	"1" if name is to be omitted from output. Otherwise, a "0" or blank.
19	"1" if sequence number is to replace I. D. number. Otherwise, a "0" or blank.
20-24	Any data in columns 20-24 of header card will be punched in columns 76-80 of each student's output card. This could be used to give the percent of the final grade that this test will be and the test number or any other identification that is needed. Another possible use for this output is to put the instructor's initials, or in some cases, their last name. If left blank, nothing will be punched.
25	"1" if blank card is desired between student answer cards. Otherwise, a "0" or blank.
26	"1" if omitted question numbers are not to be punched following the student's answer card. Otherwise, a "0" or blank.
27	"1" if weight value is not to be punched following the question number on exam analysis cards. Otherwise, a "0" or blank.
28	"0" or blank if questions answered incorrectly are to be omitted. Otherwise, a "1".
29	"1" to omit student's grade card output. Otherwise, a "0" or blank.
30	"1" to punch student's grade card. Otherwise, a "0" or blank. This is for 1443 printer version where information is printed, not punched. If used in card version, two grade cards will be punched.
31-60	Any information punched in columns 31-60 of the control card will be punched in columns 1-30 of the first header card. This can be used for course identification.

3. Keys For The Exam

The key cards for the exam are the same as the student answer cards. They are of three types: major keys, secondary keys, and weight cards.

A. MAJOR KEYS - Required
Contain the instructor's first choice of correct answers. It must contain an answer for each question to be graded. Questions not to be graded must be left blank.
I. D. columns specified by $DEFINE card have a 99---99.

B. SECONDARY KEYS - Optional
Contain alternate answers to those given on the major keys. If a question on a secondary key card is left blank, no alternate answer is assumed. There can be 4 or less secondary key cards for each major key. I. D. columns specified by $DEFINE card have a 99---98 for first alternate key, 99---97 for second, and 99---96 for third, and 99---95 for the fourth alternate key.

C. WEIGHT KEYS - Optional
If used, the weight key will have a weight for each question answered on the major key. An answer A on the weight key assigns that question a weight of 1; a B, a weight of 2; a C, a weight of 3; D, a weight of 4; and E, a weight of 5. If a question is left blank, the weight is assumed to be 1.
I. D. columns specified by $DEFINE card are numbered 99---94.

Card Column specified by $DEFINE card of ALL the key cards contain:
1, if the card pertains to the first set of questions
2, if the card pertains to the second set of questions
3, if the card pertains to the third set of questions
and so on, until
9, if the card pertains to the ninth set of questions
0, if the card pertains to the tenth set of questions

Only one answer per question is allowed, but by using the alternate key cards, if the student answers any one of the correct answers he will get credit for that question. Let it be stressed that <u>one and only one answer is to be marked per question</u>.

If any of the alternate key cards or weight cards are not marked, they do not have to be read in, but if they are read in they are ignored.

The order by which the key cards are read in after the control card is of no consequence.

4. Student's Answer Card

The student answer cards are completely defined by the $DEFINE card. The student answer cards do not have to be in any particular order and the only requirement is that all cards for one student be read in together.

OUTPUT

1. Student's Grade Card

Card Column	Data
Right justified to column 03	Section number (length specified by $DEFINE card)
Right justified to column 09	Course number (length specified by $DEFINE card)
Left justified to column 15	Student's name (length specified by $DEFINE card)
Right justified to column 43	Student I. D. number or sequence number. (length specified by $DEFINE card)
49-51	Number of correct answers
57-59	Number of incorrect answers
65-67	Number of questions omitted
72-74	Score
76-80	Any data in columns 20-24 of the control card.

2. Student's question numbers "ANSWERED INCORRECTLY" card.

Card Column	Data
18-36	Comment "ANSWERED INCORRECTLY"
45-47	First question number answered incorrectly
49-51	Second question number answered incorrectly
.	.
.	.
.	.
77-79	Ninth question number answered incorrectly

If more than nine questions were answered incorrectly, they will be punched on the following card, etc.

3. Student's "QUESTIONS OMITTED" card has the same format as the "ANSWERED INCORRECTLY" card with the exception of the comment in columns 18-36 which will be

"QUESTIONS OMITTED"

4. Grade distribution cards.

Card Columns	Data
Right justified to column 03	Section number (length specified by $DEFINE card)
Right justified to column 09	Course number (length specified by $DEFINE card)
14-16	Question number
18	Weight of questions
23-26	Number of A answers
32-35	Number of B answers
41-44	Number of C answers
50-53	Number of D answers
59-62	Number of E answers
68-71	Number of omissions
78-80	Percent of correct answers to this question

SAMPLE INPUT

```
$1050003   1. 1   1ROSS 1  1  PSYCOLOGY 201 SEC 01 + 02
99999MASTER KEY CARD   02 20117877888788887877877 778772233323 3 32323223233232332
99994WEIGHT CAPDEY     01 2011999988888777778888899999      33333      1111100000
99997ALTERNATE KEY     01 20118788
99998ALTERNATE KEY     01 2011999979   77777                          2222222
56223WILKINSON CAMILLE 01 2011
58377HABES JOHN D JR   01 20117877888888887877888778888223322333332323323232333222
60243SINGLETON WILLIAM 01 20117778887778778878877777879223332323325233323333322223
57829BAILEY PHILLIP JAS01 20117877888888878788777778773233323222333232223333332
59089SHADDINGER MARYE J01 20117877888888887877777777878233332333232322222333332
54373EMBRY JANICE PHILL01 20117877887887887877877877777822333233323222323323332332
56120TOWNSFND JOHN HERN01 201178788 778788788 877777878223 32332 33 2322 23 3232
56201WELLS STEVE C     01 20117778888787887877877777877223233332232223233333332222
55541LAFOLLETTE LOIS B 01 20117877887788777887878887888223332323231323222333332
54136CRAWFORD BUREN R  01 20117777788787787777877777877322333332333232223232332222
58988RAY SUSAN GILBERT 01 201178   8788  878777 787787723333 333 332 32 3  33332
58285FRAZIER BRENDA MAR01 20117778888788888878777877877333332123233323322333232332
55344GRIMES F ANITA    01 201178788887888888787778778772333233332 233322333333332
55147CURTIS MARY M     01 20117877888787787877877777877233332333233223332233332232
54691PASH JOHN         01 20117878788888877878777877877233332333233223233332222232
56136UZZLE ROBERT H    01 2011787788888887787887777788723332223222322333233332232
47551CHAMPION JOHN POWE01 20117777787787787877777777877722323223222222322223222322
59123SLACK CHERYL GAY  01 20117878888887887877788787888833333223332333233233332233
52850FYFE CHARLES WATKI01 20117778888788788877877877878222332233232323233333332232
57780ALEXANDER STACY M 01 20117878888888887877877777877233332333233223323233333332
55497JONES SANDRA L    01 20117878888778787877877777877223333233322333223223233323232
60082BARNES CLAIR CORTL01 2011787 88878 8878778778878873233332122232233323232332332
59232TAYLOR DIANNA TEEN01 20117778888788887878777777877223332333232223222333333322


55225EDWARDS HAROLD M J01 20117778787888787878 78777877233232333233223332233332
55257FINCH JO CAROL    01 2011787888878788787787778888223333332333333232333332
57767ABRAHAM GLADYS K  01 201177787887778878888778877872333223332223223323333332232
55659MAGEE ROBERT M    01 2011777778788887887787777787722333332223322322332322232
57140LITTON PEGGY SCOTT01 2011777888877888787788877787823333233333233322333332332
54505WHITTEN MARY ANN  01 2011777888878888887887877788723233323332322332232332232
47513CALVIN JOHN HENRY 01 201178788887888878787788878772333323332333233233333333
54650GUNN VIRGINIA C   01 2011787888888877787887777788722332233223232322223333332
54383REA WILLIAM L     02 201178788787888878778778878772233323332322233332333232
57047HARRIS CAROLYN ANN02 20117877888788887877877777877223332333 3223322323332232
55584LOGAN FREDERICK M 02 201178788887888888877877 878782332333 3 33323 233332 32
50037BEAVER SONDRA KATH02 201178778887887878777778 778772223322 3 22223 233332232
55262FLAGG CAROLYN B   02 201178787887888887877877 778782233323 3 3232322232332232
56116TIBBETTS ROBERT N 02 2011777788878878787878787 778772223332 3 32223223333232
58867NECAISE JEANNE KAY02 201178778887888887877877 778772233323 3 32323222333322232
55064CARVER CANDACE ANN02 201177777887878878777778 778782333323 2 32223222333332232
55983SHARPE SUSAN MANN 02 201177777887787777877778 778782233323 2 22323223232323232
53308RICE LEO EDWARD   02 201178778887888887878877 778772233323 3 33323233233332232
56959DOUGHERTY PATRICIA02 201178778888888887888887 778782333233 2 33323222333332232
55311GEORGE FRED ANTHON02 201178778878887777877887 788872233323 3 33323223232222232
57123LAUGHLIN JAMES RIC02 201178778887788887877877 778772333323 2 33223223333322232
55128COX ALVIN C JR    02 201178878877878777887878 777882233333 2 332232323233233
53526WEBB THOMAS LYTLE 02 201177787888878887877787 778773333323 2 3233322333333232
58576JUE TRUMAN        02 201178777877887888877777 778783222333 3 3323322223323332
58248FELLOWS DONALD K J02 20117877888787887877877777877222 3323 3 32223223 332222
58265FOOSE JONATHAN EDW02 20117878888787777878877777878233332323232233323333333332
50208CHURCH CLARENCE H 02 201177788888788788877877 878872233333 2 322233232333222
```

SAMPLE [illegible]

PSYCOLOGY 201 SEC	COURSE	SEC 01 + 02 NAME	I. D. NUMBER		NUMBER RIGHT		NUMBER WRONG		NUMBER OMITTED		SCORE	IDEN
01	201	WILKINSON CAMILLE	00001		35		9		3		70	ROSS
		ANSWERED INCORRECTLY		5	8	17	37	42	43	45	47	48
		QUESTIONS OMITTED		1	2	3						
01	201	HABES JOHN D JR	00002		36		11				72	ROSS
		ANSWERED INCORRECTLY		8	18	19	22	24	25	30	41	47
				48	49							
01	201	SINGLETON WILLIAM	00003		35		12				66	ROSS
		ANSWERED INCORRECTLY		7	9	13	16	25	37	39	41	44
				48	49	50						
01	201	BAILEY PHILLIP JAS	00004		40		7				84	ROSS
		ANSWERED INCORRECTLY		8	16	26	34	37	44	47		
01	201	SHADDINGER MARYE J	00005		41		6				86	ROSS
		ANSWERED INCORRECTLY		8	17	25	27	43	47			
01	201	EMBRY JANICE PHILL	00006		41		6				88	ROSS
		ANSWERED INCORRECTLY		7	8	10	23	25	42			
01	201	TOWNSEND JOHN HERN	00007		33		8		6		65	ROSS
		ANSWERED INCORRECTLY		7	10	15	25	34	37	47	48	
		QUESTIONS OMITTED		6	16	29	38	43	46			
01	201	WELLS STEVE C	00008		38		9				75	ROSS
		ANSWERED INCORRECTLY		10	29	31	34	39	42	44	48	49
01	201	LAFOLLETTE LOIS B	00009		37		10				77	ROSS
		ANSWERED INCORRECTLY		7	15	19	21	24	25	37	43	44
				47								
01	201	CRAWFORD BUREN R	00010		39		8				83	ROSS
		ANSWERED INCORRECTLY		5	10	26	28	31	37	48	49	
01	201	RAY SUSAN GILBERT	00011		30		6		11		63	ROSS
		ANSWERED INCORRECTLY		7	8	17	27	37	47			
		QUESTIONS OMITTED		3	4	5	10	11	18	31	39	42
				44	45							
01	201	FRAZIER BRENDA MAR	00012		38		9				79	ROSS
		ANSWERED INCORRECTLY		13	16	17	26	27	37	41	43	44
01	201	GRIMES F ANITA	00013		37		9		1		79	ROSS
		ANSWERED INCORRECTLY		13	16	17	27	30	31	39	44	47
		QUESTIONS OMITTED		36								
01	201	CURTIS MARY M	00014		39		8				75	ROSS
		ANSWERED INCORRECTLY		10	27	37	41	42	43	47	48	
01	201	PASH JOHN	00015		37		10				74	ROSS
		ANSWERED INCORRECTLY		5	8	16	17	27	37	42	44	46
				48								
01	201	UZZLE ROBERT H	00016		33		14				63	ROSS
		ANSWERED INCORRECTLY		8	16	24	27	30	32	34	37	41
				42	43	44	47	48				

01	201	CHAMPION JOHN POWE	00017		35		12				73	ROSS
		ANSWERED INCORRECTLY		5	7	10	17	22	23	29	32	34
				43	46	49						
01	201	SLACK CHERYL GAY	00018		30		17				65	ROSS
		ANSWERED INCORRECTLY		8	10	17	18	19	21	22	24	25
				26	27	30	37	41	44	48	50	
01	201	FYFE CHARLES WATKI	00019		39		8				76	ROSS
		ANSWERED INCORRECTLY		13	25	28	32	42	44	47	48	
01	201	ALEXANDER STACY M	00020		42		5				88	ROSS
		ANSWERED INCORRECTLY		8	27	37	41	47				
01	201	JONES SANDRA L	00021		42		5				86	ROSS
		ANSWERED INCORRECTLY		9	34	37	47	48				
01	201	BARNES CLAIR CORTL	00022		39		6		2		88	ROSS
		ANSWERED INCORRECTLY		21	24	26	34	39	41			
		QUESTIONS OMITTED		4	10							
01	201	TAYLOR DIANNA TEEN	00023		42		5				83	ROSS
		ANSWERED INCORRECTLY		16	17	44	47	49				
01	201	EDWARDS HAROLD M J	00024		34		12		1		70	ROSS
		ANSWERED INCORRECTLY		5	7	8	16	19	27	29	37	41
				42	43	47						
		QUESTIONS OMITTED		17								
01	201	FINCH JO CAROL	00025		36		11				74	ROSS
		ANSWERED INCORRECTLY		10	24	25	31	34	37	39	42	43
				44	47							
01	201	ABRAHAM GLADYS K	00026		32		15				67	ROSS
		ANSWERED INCORRECTLY		5	9	10	15	16	21	23	24	27
				30	37	41	44	47	48			
01	201	MAGEE ROBERT M	00027		37		10				75	ROSS
		ANSWERED INCORRECTLY		5	7	8	13	31	34	37	44	45
				48								
01	201	LITTON PEGGY SCOTT	00028		40		7				88	ROSS
		ANSWERED INCORRECTLY		9	18	19	25	27	39	44		
01	201	WHITTEN MARY ANN	00029		39		8				85	ROSS
		ANSWERED INCORRECTLY		13	16	19	24	27	28	39	48	
01	201	CALVIN JOHN HENRY	00030		37		10				74	ROSS
		ANSWERED INCORRECTLY		16	17	19	21	27	37	41	44	47
				50								
01	201	GUNN VIRGINIA C	00031		40		7				84	ROSS
		ANSWERED INCORRECTLY		8	16	24	30	34	43	47		

SEC	COURSE	SCORE	FREQUENCY	CUM. FREQ.	PERCENTILE
01	201	63	2	2	6
01	201	65	2	4	13
01	201	66	1	5	16
01	201	67	1	6	19
01	201	70	2	8	26
01	201	72	1	9	29
01	201	73	1	10	32
01	201	74	3	13	42
01	201	75	3	16	52
01	201	76	1	17	55
01	201	77	1	18	58
01	201	79	2	20	65
01	201	83	2	22	71
01	201	84	2	24	77
01	201	85	1	25	81
01	201	86	2	27	87
01	201	88	4	31	100

NUMBER OF TEST = 31 MEAN = 77 STANDARD DEVIATION = 7

SEC	COURSE	QUESTION NUMBER	A	B	C	D	E	OMIT	PERCENT CORRECT
					EXAM ANALYSIS				
01	201	1	*	*	30*			1	97
01	201	2	*	18*	12*			1	97
01	201	3	*	*	29*			2	94
01	201	4	*	19*	10*			2	94
01	201	5	*	23*	7			1	74
01	201	6 B		30*	*			1	97
01	201	7 B	*	23*	8				74
01	201	8 B		13	18*				58
01	201	9 B		27*	4				87
01	201	10 B		20*	9			2	65
01	201	11 C		21*	9*			1	97
01	201	12 C		25*	6*				100
01	201	13 C		6	25*				81
01	201	14 C		30*	1*				100
01	201	15 C		3	28*				90
01	201	16 B		12	18*			1	58
01	201	17 B		20*	10			1	65
01	201	18 B		3	27*			1	87
01	201	19 B		7	24*				77
01	201	20	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
01	201	21		5	26*				84
01	201	22		3	28*				90
01	201	23		28*	3				90
01	201	24		9	22*				71
01	201	25	1	9	21*				68
01	201	26		5	26*				84
01	201	27		14	17*				55
01	201	28		28*	3				90
01	201	29		27*	3			1	87
01	201	30		25*	6				81
01	201	31 B		5	25*			1	81
01	201	32 B		28*	3				90
01	201	33	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
01	201	34 B		21*	10				68
01	201	35	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
01	201	36		26*	4*			1	97
01	201	37		18	12*	1			39
01	201	38		13*	17*			1	97
01	201	39		7	23*			1	74
01	201	40		31*	*				100
01	201	41 D		11	20*				65
01	201	42 D		9	21*			1	68
01	201	43 D		20*	10			1	65
01	201	44 D		16	14*			1	45
01	201	45 D		28*	2			1	90
01	201	46 E		28*	2			1	90
01	201	47 E		19	12*				39
01	201	48 E		16*	15				52
01	201	49 E		25*	6				81
01	201	50 E		3	28*				90

ASTERISK (*) INDICATES CORRECT ANSWER

		QUESTIONS OMITTED		44										
02	201	FOOSE JONATHAN EDW	00049		39		8				81	ROSS		
		ANSWERED INCORRECTLY		10	16	25	27	39	41	44	47			
02	201	CHURCH CLARENCE H	00050		36		11				69	ROSS		
		ANSWERED INCORRECTLY		8	9	13	21	24	31	34	41	47		
				48	49									

SEC	COURSE	SCORE	FREQUENCY	CUM. FREQ.	PERCENTILE
02	201	54	1	1	5
02	201	66	1	2	11
02	201	69	1	3	16
02	201	72	1	4	21
02	201	73	1	5	26
02	201	76	2	7	37
02	201	78	1	8	42
02	201	81	3	11	58
02	201	82	2	13	68
02	201	84	2	15	79
02	201	86	1	16	84
02	201	88	1	17	89
02	201	93	1	18	95
02	201	94	1	19	100

NUMBER OF TEST = 19 MEAN = 79 STANDARD DEVIATION = 9

SEC	COURSE	QUESTION NUMBER	A	B	C	D	E	OMIT	PERCENT CORRECT
					EXAM ANALYSIS				
02	201	1	*	*	19*				100
02	201	2	*	14*	5*				100
02	201	3	*	1*	18*				100
02	201	4	*	6*	13*				100
02	201	5	*	14*	5				74
02	201	6 B		18*	1*				100
02	201	7 B	*	16*	3				84
02	201	8 B		4	15*				79
02	201	9 B		16*	3				84
02	201	10 B		14*	5				74
02	201	11 C		13*	6*				100
02	201	12 C		14*	5*				100
02	201	13 C		3	16*				84
02	201	14 C		19*	*				100
02	201	15 C		2	17*				89
02	201	16 B		4	15*				79
02	201	17 B		13*	6				68
02	201	18 B		4	15*				79
02	201	19 B		4	15*				79
02	201	20	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
02	201	21		3	16*				84
02	201	22		1	18*				95
02	201	23		18*	1				95
02	201	24		3	16*				84
02	201	25		8	11*				58
02	201	26		2	17*				89
02	201	27		6	13*				68
02	201	28		15*	4				79
02	201	29		17*	2				89
02	201	30		18*	1				95
02	201	31 B		6	13*				68
02	201	32 B		17*	2				89
02	201	33	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
02	201	34 B		12*	7				63
02	201	35	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
02	201	36		17*	2*				100
02	201	37		7	12*				63
02	201	38		8*	11*				100
02	201	39		4	15*				79
02	201	40		19*	*				100
02	201	41 D		3	14*			2	74
02	201	42 D		3	16*				84
02	201	43 D		14*	5				74
02	201	44 D		10	8*			1	42
02	201	45 D		18*	1				95
02	201	46 E		17*	2				89
02	201	47 E		10	9*				47
02	201	48 E		2*	16			1	11
02	201	49 E		17*	2				89
02	201	50 E		1	18*				95

ASTERISK (*) INDICATES CORRECT ANSWER

TOTAL FOR ALL SECTIONS

COURSE	SCORE	FREQUENCY	CUM. FREQ.	PERCENTILE
201	54	1	1	2
201	63	2	3	6
201	65	2	5	10
201	66	2	7	14
201	67	1	8	16
201	69	1	9	18
201	70	2	11	22
201	72	2	13	26
201	73	2	15	30
201	74	3	18	36
201	75	3	21	42
201	76	3	24	48
201	77	1	25	50
201	78	1	26	52
201	79	2	28	56
201	81	3	31	62
201	82	2	33	66
201	83	2	35	70
201	84	4	39	78
201	85	1	40	80
201	86	3	43	86
201	88	5	48	96
201	93	1	49	98
201	94	1	50	100

NUMBER OF TEST = 50 MEAN = 78 STANDARD DEVIATION = 8

TOTAL FOR ALL SECTIONS

EXAM ANALYSIS

COURSE	QUESTION NUMBER	A	B	C	D	E	OMIT	PERCENT CORRECT
201	1	*	*	49*			1	98
201	2	*	32*	17*			1	98
201	3	*	1*	47*			2	96
201	4	*	25*	23*			2	96
201	5	*	37*	12			1	74
201	6 B		48*	1*			1	98
201	7 B	*	39*	11				78
201	8 B		17	33*				66
201	9 B		43*	7				86
201	10 B		34*	14			2	68
201	11 C		34*	15*			1	98
201	12 C		39*	11*				100
201	13 C		9	41*				82
201	14 C		49*	1*				100
201	15 C		5	45*				90
201	16 B		16	33*			1	66
201	17 B		33*	16			1	66
201	18 B		7	42*			1	84
201	19 B		11	39*				78
201	20	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
201	21		8	42*				84
201	22		4	46*				92
201	23		46*	4				92
201	24		12	38*				76
201	25	1	17	32*				64
201	26		7	43*				86
201	27		20	30*				60
201	28		43*	7				86
201	29		44*	5			1	88
201	30		43*	7				86
201	31 B		11	38*			1	76
201	32 B		45*	5				90
201	33	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
201	34 B		33*	17				66
201	35	THIS QUESTION OMITTED AT THE REQUEST OF THE INSTRUCTOR						
201	36		43*	6*			1	98
201	37		25	24*	1			48
201	38		21*	28*			1	98
201	39		11	38*			1	76
201	40		50*	*				100
201	41 D		14	34*			2	68
201	42 D		12	37*			1	74
201	43 D		34*	15			1	68
201	44 D		26	22*			2	44
201	45 D		46*	3			1	92
201	46 E		45*	4			1	90
201	47 E		29	21*				42
201	48 E		18*	31			1	36
201	49 E		42*	8				84
201	50 E		4	46*				92

ASTERISK (*) INDICATES CORRECT ANSWER

UNIV OF MISS TEST SCORING REVISED (UMTS-R) NOVEMBER 18, 1966 PAGE 1

SYMBOL TABLE

ZZZZZZ	13559	THRUUA	04292	TAB2L1	01035	TAB1L1	00975	STARTX	09120	SQRSUB	10724	RIGHTT	05008
READCK	00408	READBB	00444	PUNCH1	11706	PUNCHH	11656	FREQTL	01811	FREQMT	02235	FREQL1	01227
DECSRR	10540	CHECKK	13420	CHECKA	04220	ABEGIN	00401	AAA	10723	ACN	12128	ACOUR	12118
ADN	09403	ADR	09400	AID	12088	ALRL	01096	ANAME	12098	ANS	13569	ANSZ	13351
APA	08172	APB	08184	APC	08208	APCC	07916	APD	08240	APDD	08288	APNX	01744
AQUES	12138	ARN1	02140	ARN11	02164	ARN3	01904	ARN4	01916	ARN6	02344	ARN7	02784
ASEC	12108	AST	09413	ATDE	09541	A14	10358	A71	11889	A8	10171	BA	03472
BFF	02716	BL	12307	BLANK	11923	BLC	03809	BNF	02660	BNFF	02696	BN1	04594
BN2	03076	BN3	03112	BN6	03376	BN66	03460	BRAN	09144	CARD	12301	CFLAG	00614
CHECK	04944	CID	10208	CIDC	10230	CKA	13439	CKB	13459	CN	07255	CNA1	00729
CNA2	00735	CONA	13389	CONB	13411	COUR	10320	COURC	10332	COURR	10344	CON	13461
CT	06487	DAD	09348	DADD	09361	DBN	11368	DDA	09353	DDACK	09347	DECA	10711
DECAX	04044	DECL	10669	DECS	10364	DECSR	10396	DECX	04605	DEC1	10660	DISC	00001
DIV	11046	DIVA	11507	DIVB	11517	DIVC	04042	DIVD	07230	DIVJ	11108	DIVK	11060
DIVV	11324	DOUT	10696	D1X	11475	D2X	11480	D3N	11440	D3X	11487	D4X	11492
ERRA	03892	ERRAA	03928	ERRF	09368	ERR1	09693	ERR2	09761	ERR3	09835	ERR4	09929
ERR5	09985	ERR6	06077	ERR6H	06119	ERR7	04663	EXTRA	12357	FREQ	01203	FREQL	01215
FREQT	01695	GRA	12277	GRB	12287	HA	12369	HB	12531	HC	12693	HCA	03976
HCAA	03988	HCT	04056	HD	12855	HE	13017	HG2	12355	HOLD	02283	HOLD1	02319
HOLD2	02451	HP20	04700	HP24	03648	HP36	01788	HX	10875	H22	10141	H24	09068
H322	05960	H44	13179	H488	06252	H4888	06160	H522	05984	H588	05808	H722	06056
H822	05912	J	12207	JA	04435	JID	10252	JIDC	10274	JIDCC	10296	K	12217
L	12227	LA1	05880	LA2	05892	LA3	05936	LA4	05972	LA5	05868	LCARD	09048
LOOP	04832	LOOPA	04164	LT	00721	LT5	00719	LT5T1	00747	M	12237	MAXS	03787
MEAN	08419	MEN	09527	MM	12257	N	12247	NCARD	00682	NCOPY	00939	NG	09583
NINE	11921	NMM	12309	NN	12267	NOCK	05104	NOMIT	13569	NOMZ	04900	NO	03724
NQDT	03797	NQM1	04136	NQTT	03505	NQT2	03761	NUMT	09493	OMIT	04139	PCOUR	07688
PLLA	05620	PLLAA	06672	PLLB	06684	POUT	07340	PPA1	11680	PPA2	11692	PSA	06900
PSB	07040	PSEA	06432	PSEC	06464	PSFCS	06388	PSEE	03856	PSSS	06696	PSX	06584
PUNCH	11524	QC	06485	QQ	12359	QUES	12304	RAD	03776	RAD24	03776	RCNN	09032
RCON	11029	RCONX	09395	RDEC	10432	READ	00760	READA	00446	READL	00470	READ1	00514
READ2	00570	RED	03692	RIGHT	13367	RM	12174	ROSP	08672	ROUT	08716	ROUTT	08696
RPUN	08928	RR	13510	RRC	13554	RRCC	10186	RRCP	13530	RTR	13559	RTTT	08740
SC	03727	SDC	12297	SFC	10302	SECC	10308	SETUP	04328	SF	00674	SFF	00662
SFLAG	00638	SID	12345	SIZE	00685	SNOM	12365	SQRS	10760	SQRT	10816	START	00736
STAT	00784	STATX	09100	STRT	09204	STT	02560	STUD	03028	TABZ	12153	TABZL	03039
TAB1	00807	TAB1L	00963	TAB2	00999	TAB2L	01023	TAB21	01095	TAS	10095	THC	12209
TCNT	07159	TESTA	03327	TESTB	03471	THR	05044	THRA	04256	THRU	05032	THRUA	04244
THRUF	04976	THRUU	05080	TNF	07231	TOUT	08564	TTN	07880	WET	02272	WN	12363
WRON	02271	WRONG	04047	WT	13566	WTT	13440	WW	12361	X	11727	XA	05115
XCOUR	12331	XNAME	12343	XP82	11919	XSAVE	12323	XSFC	12325	XSECA	12327	XSECB	12329
XTOT	12337	Y	13189	ZERO	12081	ZRA	10918	ZRB	01552	ZRC	10990	ZRD	01624
ZRE	01636	ZRF	01988	ZRG	02036	ZRJ	02332	ZRM	03184	ZRN	03172	ZRO	10882
ZRP	02096	ZRR	02960	ZRRX	02996	ZRZ	01480	ZSS	02856	Z10	11931	Z13	11934
Z18	11939	Z7	11928	Z8	11929	Z A	13810	Z AA	13575	Z AB	13657	Z ABC	13829
Z AL	17370	Z ALL	14083	Z AN	17406	Z AT	13655	Z A1	16974	Z A14	13839	Z A2	16986
Z BB	13719	Z CC	13741	Z CHECK	13641	Z CONA	14066	Z CORF	14071	Z DCK	14715	Z DIVD	13644
Z ERR	15061	Z HDHD	14351	Z H1	14379	Z H2	14443	Z H22	14533	Z H4	14205	Z H44	14523
Z H5	14223	Z H6	14239	Z J	14042	Z LT	14079	Z NCARD	14077	Z PRINC	14072	Z PRINT	13766
Z PRTT	17766	Z PR1	13768	Z PR3	13780	Z PR4	13792	Z RCON	18175	Z RDR	14080	Z RM	14044
Z ROSS	14109	Z RRC	13877	Z RRCC	13854	Z RT	17910	Z R1	15138	Z R11	17994	Z R2	15294
Z R22	18030	Z R3	15378	Z R33	18066	Z R4	15402	Z SIZE	14075	Z START	14736	Z STP	14940
Z TM	17130	Z TNF	13645	Z X	14555	Z XA	15102	Z ZERO	14037				

```
LOCTN  OP   P/I.    Q          PG/LN  LABEL  MNEM OPERANDS AND REMARKS

                               00010  *      UNIV OF MISS TEST SCORING REVISED    (UMTS-R)
                               00020  *      RICHARD D ROSS
                               00030  *      OCTOBER 6, 1966
00402                          00040         DORG  402
00001       00000                     DISC   DS    ,1,,                    DISC=0 FOR NO DISC  1 FOR DISC
00401       00000              00050  ABEGIN DS    ,*
                               00060  *      SUBROUTINE TO READ A CARD AND CHECK FOR
                               00070  *      MBR-E  AND  MBR-O  CHECK STOPS
00406       00005              00080         DS    5
00408  37   11727   00500      00090  READCK RACD  X
00420  46   00446   01200      00100         BE    READA
00431       00005              00110         DC    5,01600,*
00432  46   00446   01200      00120         BE    READA
00443       00005              00130         DC    5,01700,*
00444  42   00000   00000      00140  READBB BB
00446                          00150         DORG  *-9
00446  16   00481   ‡1726      00160  READA  TFM   READL+11,X-1
00458  46   00470   01200      00170         BE    *+12
00469       00005              00180         DC    5,01700,*
00470  43   00482   12138      00190  READL  BD    *+12,AQUES
00482  46   00514   01200      00200         BE    READ1
00493       00005              00210         DC    5,01600,*
00494  46   00514   01200      00220         BE    READ1
00505       00005              00230         DC    5,01700,*
00506  49   00570   00000      00240         B     READ2
00513                          00250         DORG  *-4
00514  15   0048‡   00000      00260  READ1  TDM   READL+11,0,6
00526  11   00481   000θ1      00270         AM    READL+11,1,10
00538  15   0048‡   00000      00280         TDM   READL+11,0,6
00550  12   00481   000θ1      00290         SM    READL+11,1,10
00562  49   00482   00000      00300         B     READL+12
00569                          00310         DORG  *-4
00570  11   00481   000θ2      00320  READ2  AM    READL+11,2,10
00582  14   00481   ‡1885      00330         CM    READL+11,X+80*2-2
00594  47   00470   01100      00340         BNP   READL
00606  42   00000   00000      00350         BB
00608                          00360         DORG  *-9
                               00370  *      END OF SUBROUTINE
                               00380  *      SUBROUTINE TO SET FLAGS AND CLEAR FLAGS
00612       00005              00390         DS    5
00614  15   00675   00003      00400  CFLAG  TDM   SF+1,3
00626  49   00650   00000      00410         B     SFLAG+12
00633                          00420         DORG  *-4
00637       00005              00430         DS    5
00638  15   00675   00002      00440  SFLAG  TDM   SF+1,2
00650  16   00673   ‡2088      00450         TFM   SFF+11,AID
00662  26   00680   12088      00460  SFF    TF    SF+6,AID
00674  32   12088   00000      00470  SF     SF    AID
00685       00003              00480  SIZE   DS    3,*,,                   MAXIMUM NUMBER OF QUESTIONS
00682       00002              00490  NCARD  DS    2,*-3,,                 MAXIMUM NUMBER OF CARDS
00686  11   00673   000‡0      00500         AM    SFF+11,10,10
00698  45   00662   00673      00510         BNR   SFF,SFF+11,11
00710  32   11726   00000      00520         SF    X-1
00721       00002              00530  LT     DS    2,*,,                   LENGTH OF CONSTANTS ( 03, 04 OR 05
00719       00000              00540  LT5    DS    ,*-2,,                  LT= LT * 5
00722  42   00000   00000      00550         BB
```

```
     UNIV OF MISS TEST SCORING REVISED     (UMTS-R)                                        NOVEMBER 18, 1966     PAGE    2

LOCTN  OP   P/L     Q           PG/LN  LABEL  MNEM OPERANDS AND REMARKS

00724                           00560         DORG   *-9
00729       00006               00570  CNA1   DC     6,100002
00735       00006               00580  CNA2   DC     6,900018
                                00590  *      END OF SUBROUTINE
00736  32   00760   00000       00600  START  SF     READ
00747       00005               00610  LT5T1  DS     5,*,,                       LT5T1= LT*5*1000+1
00748  16   11889   00000       00620         TFM    A71,00,10
00760  26   11885   12081       00630  READ   TF     X+80*2-2,ZERO
00772  17   00408   00000       00640         BTM    READCK
00784  17   00614   00000       00650  STAT   BTM    CFLAG
00796  32   12088   00000       00660         SF     AID,,6
00807       00005               00670  TAB1   DS     5,*,,                       TAB1= TAB1
00808  14   12093   00000       00680         CM     AID+5,0,610
00820  46   00760   01200       00690         BE     READ,,,                   IF BLANK CARD IS READ,
                                00700  *                                      SKIP THE CARD
00832  14   11727   000+3       00710         CM     X,13,10
00844  47   02140   01200       00712         BNE    ARN1
                                00714  *
                                00716  *      DISC VERSION    READ IN NUMBER OF COPIES NEEDED
                                00718  *      ON        $$020
                                00720  *      WHICH SAYS YOU NEED TWO EXTRA COPIES AND USE DRIVE 0
                                00721  *
                    00000       00722         IF     DISC-1,8,0
00856  14   11729   0+313       00723         CM     X+2,1313,8
00868  47   00940   01200       00724         BNE    *+72
00880  25   00939   11733       00725         TD     NCOPY,X+4*2-2
00892  25   00938   11731       00726         TD     NCOPY-1,X+3*2-2
00904  25   09348   11735       00727         TD     DAD,X+5*2-2,,              PICK UP DISC DRIVE CODE
00916  16   09353   00000       00728         TFM    DDA,0
00928  49   00760   00000       00729         B      READ
00939       00003               00730  NCOPY  DC     3,0,*
                                00735  *      HEADER CARD
00940  16   11889   00070       00740         TFM    A71,70,10
00952  33   00760   00000       00750         CF     READ
00963       00005               00760  TAB1L  DS     5,*,,                       TAB1L= TAB1 - LT + 1
00964  33   08740   00000       00770         CF     RTTT
00975       00005               00780  TAB1L1 DS     5,*,,                       TAB1L1= TAB1 - LT + 1 + 1
00976  17   00614   00000       00790         BTM    CFLAG
00988  32   11742   00000       00800         SF     X+9*2-3
00999       00005               00810  TAB2   DS     5,*,,                       TAB2= TAB2
01000  26   12427   11845       00820         TF     HA+30*2-2,X+60*2-2
01012  32   12330   00000       00830         SF     XCOUR-1
01023       00005               00840  TAB2L  DS     5,*,,                       TAB2L= TAB2 - LT + 1
01024  32   12336   00000       00850         SF     XTOT-1
01035       00005               00860  TAB2L1 DS     5,*,,                       TAB2L1= TAB2 - LT + 1 + 1
01036  11   12331   00009       00870         AM     XCOUR,9,10
01048  11   12337   00009       00880         AM     XTOT,9,10
01060  11   12363   00009       00890         AM     WN,9,10
01072  11   12357   00009       00900         AM     EXTRA,9,10
01084  32   12346   00000       00910         SF     HG2-9
01095       00005               00920  TAB21  DS     5,*,,                       TAB21= TAB2+1
01096  25   +2300   11727       00930  ALRL   TD     CARD-1,X,2
01108  21   01107   00729       00940         A      ALRL+11,CNA1
01120  14   01102   +2309       00950         CM     ALRL+6,CARD-1+9
01132  47   01096   01200       00960         BNE    ALRL
```

```
LOCTN  OP  P/L    Q        PG/LN  LABEL   MNEM OPERANDS AND REMARKS

01144  22  01107  00735    00970          S    ALRL+11,CNA2
01156  15  12308  00005    00980          TDM  NMM-1,5,11
01168  15  12300  00000    00990          TDM  CARD-1,0,11
01180  43  01204  12301    01000          BD   *+24,CARD
01192  15  12300  00001    01010          TDM  CARD-1,1,11
01203      00005           01020  FREQ    DS   5,*,,                    FREQ= FREQ
01204  32  12302  00000    01030          SF   QUES-2
01215      00005           01040  FREQL   DS   5,*,,                    FREQL= FREQ - LT + 1
01216  32  12305  00000    01050          SF   BL-2
01227      00005           01060  FREQL1  DS   5,*,,                    FREQL1= FREQ - LT + 1 + 1
01228  23  12301  03724    01070          M    CARD,NQ
01240  46  01468  01200    01080          BZ   ZRZ-12
01252  22  00099  03724    01090          S    99,NQ
01264  24  00099  12304    01100          C    99,QUES
01276  46  01468  01300    01110          BNN  ZRZ-12,,,              NUMBER OF CARDS AND QUESTIONS
                           01120  *                                DO NO AGREE
01288  23  12301  03724    01130          M    CARD,NQ
01300  24  00099  12304    01140          C    99,QUES
01312  47  01468  01300    01150          BN   ZRZ-12
01324  26  12323  12329    01160          TF   XSAVE,XSECB
01336  26  12565  11931    01170          TF   HB+18*2-2,Z10
01348  43  01372  12343    01180          BD   *+24,XNAME
01360  26  12565  12315    01190          TF   HB+18*2-2,NMM+4*2-2
01372  26  12689  13185    01200          TF   HB+80*2-2,H44+4*2-2
01384  14  12355  00000    01210          CM   HG2,0,10
01396  47  01420  01200    01220          BNE  *+24
01408  26  12689  11929    01230          TF   HB+80*2-2,Z10-2
01420  24  12301  00682    01240          C    CARD,NCARD
01432  46  01468  01100    01250          BP   ZRZ-12
01444  24  12304  00685    01260          C    QUES,SIZE
01456  47  01480  01100    01270          BNP  ZRZ
01468  17  09368  09985    01280          BTM  ERRF,ERR5
01480  17  10882  00000    01290  ZRZ     BTM  ZRO,,,                 ZERO TAB1, TAB2, FREQ, AND FREQT
01492  31  01023  00965    01300          TR   TAB2L,TAB1L,611
01504  31  01811  01215    01310          TR   FREQTL,FREQL,611
                           01320  *       ZERO ANSWER TABLE
01516  22  09403  09403    01330          S    ADN,ADN
01528  11  09400  13561    01340          AM   ADR,ANS-9+1
01540  22  09403  12304    01350          S    ADN,QUES
01552  31  09400  13350    01360  ZRB     TR   ADR,ANSZ-1,6
01564  11  09403  10001    01370          AM   ADN,10001
01576  43  01552  09401    01380          BD   ZRB,ADN-2
                           01390  *       SET UP CHECK CONSTANTS
01588  26  13439  11931    01400          TF   CKA,Z10
01600  26  13459  11931    01410          TF   CKB,Z10
01612  16  12207  00001    01420          TFM  J,1,10
01624  25  01642  12207    01430  ZRD     TD   ZRE+6,J
01636  15  13439  00001    01440  ZRE     TDM  CKA,1
01648  11  12207  00001    01450          AM   J,1,10
01660  24  12207  12301    01460          C    J,CARD
01672  47  01624  01100    01470          BNP  ZRD
01684  32  13430  00000    01480          SF   CKA-9
01695      00005           01490  FREQT   DS   5,*,,                    FREQT= FREQT
                           01500  *       ZERO CARD/QUESTION CONSTANT AND INITILIZE
01696  31  13460  13350    01510          TR   CON-1,ANSZ-1
```

COMMON

New Orleans, Louisiana

"A Computer Plotting Language"

Dr. James R. Oliver
and
John McMahon

University of Southwestern Louisiana
Box 133, U.S.L.
Lafayette, Louisiana 70501

M-3.5
Monday
November 28, 1966
1:30 - 3:00 P.M.

Text 16

A COMPUTER PLOTTING LANGUAGE

James R. Oliver University of Southwestern Louisiana, Lafayette, Louisiana
and
John G. McMahon, General Motors Proving Grounds, Flint, Michigan

Results described in graphical form are often much easier to interpret and provide a more meaningful description than information which is merely aligned in the rows and columns of a printed report. The architect would like to use the plotter in formulating his design of a building. On the drawing board, he makes a sketch of his ideas and if any alterations are to be made, he begins again to redraw the portions he wishes to retain in his design. Using a plotter language, one or two of the parameter cards can be changed and the complete drawing with the changes can be made with a minimum of time and effort. By means of the plotter, the civil engineer is able to view the geometry of the construction of a new street or expressway and how it will affect surrounding property. The dimensions of the parts of an engine can be calculated by a computer and a drawing graphically constructed with the plotter making it unnecessary for the mechanical engineer to draw these parts by hand from generated data. The geographer is capable of constructing a contour map from elevation data points by means of the plotter. There are also many other disciplines which could use the plotter as a graphical device to great advantage.

At the present time, there are very few automatic programming languages available which make use of an x-y plotter on line with a small computer. In the available literature there were only two programs designed to implement a language to make use of the plotter: "IBM 1620-1627 Fortran Plotter Subroutines" and "1620 Numerical Surface Techniques and Contour Map Plotting". The Fortran plotter subroutines exist in several forms but all are referenced by a Fortran CALL statement followed by a list of parameters. Experience has shown that this method is inadequate for many purposes and is somewhat difficult to learn, particularly for beginning programmers.

The Numerical Surface and Contour Map Program consists of several programs which calculate approximations to values defining a surface and prepare a display of the surface in the form of a map. The input to this program must follow a rigid format. An understanding of the program somewhat above that expected of a beginner is necessary to utilize this system of programs effectively.

It is for these reasons that it would be desirable to construct a plotter language which requires a minimum knowledge of computers and computer languages. The EZPLOT language was written to allow the use of the plotter as graphical output device with a minimum of restrictions and programming effort.

There are forty-eight characters in the EZPLOT language. These

include 26 letters of the alphabet, the ten decimal digits, and twelve special characters: blank [ ], period [.], right parenthesis [ )], plus sign [+], dollar sign [$], asterisk [*], minus sign [-], slash or solidus [/], comma [,], left parenthesis [( ], equal sign [=], and 'comercial at' sign [@].

Combinations of these characters are used to construct statements in the EZPLOT language. There are three types of statements: source statements, comment statements and data statements. Source statements define operations to be performed and are decomposed by the interpretive processor and then executed. These statements are composed of from one to five input records. Each input record consists of eighty columns on a card or 80 type positions on the console typewriter. Blanks have no meaning except in H format specification, to be discussed in a later chapter, and hence they are ignored. Every input record which is a part of the source statement must have a 'commercial at' sign [@] in the first position of the input record. The source statements themselves must be located in columns 7-72, inclusive, of the source input record. Positions 73-80 of the source input record are completely ignored by the processor and may be used for card sequencing, program identification or any other purpose the programmer wishes. It is necessary that the dollar sign character [$] be the last character in the statement. The dollar sign indicates to the interpreter that the end of a statement has been reached. At this point, source statement decomposition is completed and corresponding machine language commands are executed.

Comment statements may be used in documenting a program and are defined by an asterisk [*] in the first position of the input record. The comment itself may be located in one or more positions 2-80 of the input record. As a result of the identifying asterisk, the comment will not be processed as part of the program and will instead serve as supplementary information to persons reading the program. Comments may also be placed after the dollar sign which denotes the termination of a source statement since processing does not continue beyond that point on that input record.

Data statements consist of information on an input record which is to be read by an input statement and utilized by the program to produce the desired results. The data input record must not have a 'commercial at' sign [@], an asterisk [*], or a slash or solidus [/] in the first position.

DEFINE STATEMENTS

DEFINE statements are necessary for indicating which options are to be selected for orienting the plotting surface, or what physical size of the plotting surface is to be used, and the minimum and maximum values to be plotted. DEFINE statements also accomplish certain special assigning and converting.

DEFINE ORIENTATION STATEMENTS

The DEFINE ORIENTATION statement indicates to the interpretive processor, in one of three systems, how a pair of coordinates is to be oriented physically with respect to the plotter. This statement should precede any plotting which is to take place in the program. It allows the programmer the freedom of specifying whether the first or the second coordinate in a coordinate pair is to be plotted lengthwise in addition to designating a positive and negative directions of each coordinate. The plotting surface is rotated internally about the coordinates of the current pen position should any plotting have taken place.

The argument, enclosed in parentheses, indicates that the top of the plotter drum is to move toward the front when it is used in further plotter operations. The three systems or conventions mentioned are Cartesian coordinates (+X, -X, +Y, -Y), geographical direction (NORTH, SOUTH, WEST, EAST), and position in the coordinate pair (+1, -1, +2, -2). This statement must have a single argument from the set of 12 possible arguments mentioned above. Otherwise, an error message will be typed at which time the error may be corrected. A positive number in the first position of a pair of coordinates in subsequent statements is considered to be +X in the Cartesian system, NORTH in the geographical system, and +1 in the coordinate position system. The others correspond to the directions in their respective systems. Thus:

+X is the same as NORTH is the same as +1;

-X is the same as SOUTH is the same as -1;

+Y is the same as WEST is the same as +2;

-Y is the same as EAST is the same as -2.

For example:

```
DEFINE ORIENTATION (+Y)
```

This statement indicates that a positive value in the second coordinate position (+Y) will be plotted by moving the top of the drum forward and that a positive value in the first position (+X) will be plotted in a direction which is to the right when one faces the plotter.

At the beginning of every program a DEFINE ORIENTATION (+X) is automaticallv executed internally. The first statement of this type in the program will perform re-orientation of coordinates. The orientation is changed by either another DEFINE ORIENTATION statement or by initiating another program.

DEFINE PLOT AREA STATEMENT

The DEFINE PLOT AREA statement indicates to the interpretive processor the actual physical plot size, and the minimum and maximum values to be plotted in each direction. This statement should also precede any statements which perform plotting operations. The primary purpose of this statement is to permit the programmer to scale the drawing to the desired size.

In a DEFINE PLOT AREA statement, there are six arguments enclosed in parentheses and separated by commas. This statement is of the following form:

DEFINE PLOT AREA (XMIN, XMAX, XSIZE, YMIN, YMAX, YSIZE).

XMIN, XMAX, XSIZE, YMIN, YMAX, and YSIZE are arithmetic expressions. XMIN indicates the minimum value and XMAX indicates the maximum value which will appear as the first coordinate of a pair of coordinates in subsequent plotting operations. XSIZE specifies the physical distance between XMIN and XMAX on the plotting surface in inches. YMIN indicates the minimum value and YMAX indicates the maximum value which will appear as the second of a pair of coordinates. YSIZE specifies the physical distance between YMIN and YMAX in inches.

The value of XSIZE and YSIZE must be positive and less than or equal to 999.99. The current physical pen position will be defined as XMIN and YMIN when this statement is encountered. For example:

DEFINE PLOT AREA (-3.0, 31.0, 6.4, 73.5, 76.5, 9.0).

The current pen position is defined as (-3.0, 73.5). There will be six and four-tenths inches between the minimum and maximum values of the first coordinate and nine inches between the minimum and maximum values of the second coordinate. Errors such as the wrong number of arguments, a 'maximum value' less than its corresponding 'minimum value' or a 'size value' outside the allowable range, will be typed on the console typewriter and the statement may be corrected at this time.

A DEFINE PLOT AREA (0.0, 99999.0, 999.99, 0.0, 99999.0, 999.99) is automatically executed internally at the beginning of every program. The plot area and the minimum and maximum plot values are changed by either another DEFINE PLOT AREA statement or by initiating another program.

DEFINE POINT STATEMENT

The purpose of the DEFINE POINT statement is to assign a pair of coordinates to one name. In this way, a single point may be referred to by one name, called a point variable name, rather than by its two coordinates. This statement has three arguments enclosed in parentheses

and separated by commas. It takes the following form:

DEFINE POINT (NAME, XCOR, YCOR).

XCOR and YCOR are arithmetic expressions which define the location of the point called NAME. NAME is a point variable name synonymous with position (XCOR, YCOR) and may be used in place of a pair of coordinates where coordinates are required or may appear in an arithmetic expression used to indicate one or both coordinates where coordinates are required. NAME must not be used in any other manner. For example:

DEFINE POINT (PLACE, 3.0, 7.0).

This statement has defined PLACE as a point variable synonymous with the coordinates (3.0, 7.0) and thus may be used instead of this pair. As a further example, consider the following statement to follow the DEFINE POINT statement expressed above:

DEFINE POINT (NEXT1, PLACE+4.3, PLACE+6.9).

This statement is synonymous with:

DEFINE POINT (NEXT1, 7.3, 13.9).

The following statement is invalid unless PLACE has been previously specified as a simple variable:

X = PLACE+1.0

An appropriate error message will be typed on the console typewriter if any syntactical error occurs at which time the error may be corrected. If a DEFINE ORIENTATION or DEFINE PLOT AREA statement should occur later in the course of the program, the point variable will be defined as if the same DEFINE POINT statement followed after these statements.

DEFINE PEN POSITION STATEMENT

The purpose of the DEFINE PEN POSITION statement is to reassign pen coordinates. This statement defines the present physical position of the pen in terms of two coordinates. Thus this particular physical point has reassigned values for its coordinates. This statement may be in either of two forms:

DEFINE PEN POSITION (NAME)

or DEFINE PEN POSITION (XVL, YVL).

In the former statement, NAME is a point variable and has appeared as the first argument in a DEFINE POINT statement. In the letter statement, XVL and YVL are arithmetic expressions. The argument(s) are enclosed in parentheses and are separated by a comma if two arguments are used.

DEFINE RADIANS STATEMENT

The DEFINE RADIANS statement converts degrees, minutes, and seconds to radians which are units used in angle measurement. This statement may have from two to four arguments enclosed in parentheses and separated by commas. It is of the following form:

```
DEFINE RADIANS (RADN, DEGR, MIN, SEC).
```

DEGR, MIN, and SEC are arithmetic expressions denoting degrees, minutes, and seconds, respectively, which are to be converted to radians. It is not necessary for seconds or minutes to appear in which case it will be assumed that they are zero. If minutes do not appear, seconds also must not appear in the list. RADN is a variable which will contain the converted value. For example:

```
DEFINE RADIANS (ANGLE, 37.0, 23.0, 15.0).
```

Thirty-seven degrees, twenty-three minutes and fifteen seconds are equal to .65253497 radians. As a result of this statement the value .65253497 is stored in the variable location called ANGLE. Any error in syntax will be typed on the console typewriter and at this time the error can be corrected.

PLOT-DRAW STATEMENTS

It is this series of statements which cause plotter pen movement to produce plotted lines. There are six such statements defined by the EZPLOT language. Additional commands can be added to the system. These statements may be written with the word PLOT or with the word DRAW. In this and each case the result is the same. In this chapter, these statements will be typed on the console typewriter should any syntactical errors be encountered at which time they may be corrected.

PLOT POINT STATEMENT

This statement causes the pen to be lowered and moved in a straight line to the position specified in the argument list. The PLOT POINT statement is of the following form:

```
PLOT POINT (XC, YC).
```

XC and YC are arithmetic expressions separated by commas specifying the terminal point of the line to be drawn. A point variable may replace the coordinate pair. The purpose of this statement is to allow the programmer to draw a straight line from the current position of the pen to a specified point without specifying the initial point.

PLOT LINE STATEMENT

This statement causes the pen to be raised and moved to the first position specified in the argument list. The pen then moves in a straight line to the second point and from the second point to the third point until the argument list has been exhausted. If there is a comma between two pairs of coordinates, the pen will be in the down position when drawing the line between those two points. If, however, there is a period between two pairs of coordinates, the pen will be in the up position when drawing the line between the two points. A pair of coordinates may be replaced by a previously defined point variable. A pair of coordinates must be enclosed in parentheses and separated by a comma within the parentheses. Pairs of coordinates must be separated by commas indicating a line is to be drawn or by periods indicating a line is not to be drawn between the two points. For example:

```
PLOT LINE (X1,Y1), (X2,Y2), (X3,Y3). (X4,Y4), (X5,Y5).
```

This statement indicates that a line is to be drawn between point (X1,Y1) and (X2,Y2) and between (X2,Y2) and (X3,Y3) and between (X4,Y4) and (X5,Y5). Notice there will be no line drawn between point (X3,Y3) and (X4,Y4).

The programmer is allowed to draw several connected and/or disconnected lines by means of a single statement. By constructing a compound statement such as this, it becomes unnecessary to form many short, simple statements. It is allowable to use PLOT LINES in place of PLOT LINE. With the same argument list, they yield the same result.

PLOT VECTOR STATEMENT

This statement causes the pen to be raised and moved to a point. The pen is then moved at some angle from the plus X axis a specified distance. The pair of coordinates must be enclosed in parentheses and separated by a comma. The coordinate pair may be replaced by a point variable. The PLOT VECTOR statement may appear in either of the following forms:

```
PLOT VECTOR (XVAL,YVAL), (RADN), (DIST)

or PLOT VECTOR (XVAL,YVAL), (RADN), (DIST).
```

XVAL and YVAL are arithmetic expressions indicating the coordinates of the

initial point of the line. A comma after the coordinates indicates the pen is to move with the pen in the down position. A period after the coordinates indicates the pen is to move with the pen in the up position and that no line is to appear on the plotting surface as a result. RADN is an arithmetic expression indicating the angle countercloskwise from the plus X axis if RADN is positive and clockwise from the plus X axis if RADN is negative. RADN is enclosed in parentheses and is followed by a comma. DIST is an arithmetic expression indicating the length of the line measured in the units which are being plotted. If the distance is negative, the line will be drawn at an angle of RADN-3.14159265 with a length of -DIST. DIST is enclosed in parentheses.

If the scaling constants for X and Y are unequal, that is if XSIZE/(XMAX-XMIN) is not equal to YSIZE/(YMAX-YMIN). As specified in the last DEFINE PLOT AREA statement, then the number of units per inch in the Y direction and X direction are different. By convention, the scaling constant which yields the shortest line will be used in determining the length of the line. For example, consider the following two statements as part of a program:

```
DEFINE PLOT AREA (0.0, 10.0, 30.0, 0.0, 5.0, 10.0)

PLOT VECTOR (1.0, 3.5), (.78539816), (4.5).
```

The scaling constants are as follows:

XSCALE = XSIZE/(XMAX-XMIN) = 30.0/(10.0-0.0) = 3 inches per X unit,

YSCALE = YSIZE/(YMAX-YMIN) = 10.0/(5.0-0.0) = 2 inches per Y unit.

Since the scaling constants are unequal and the Y scaling constant yields a shorter line than the X scaling constant, the former will be used in determining the length of the resulting line. The two statements mentioned produce a nine inch line drawn at an angle of .78539816 radians or fourty-five degrees with positive X axis from the point (1.0, 3.5).

PLOT CIRCLE STATEMENT

This statement causes a circle to be drawn. The PLOT CIRCLE statement takes the following form:

```
PLOT CIRCLE (XCNT,YCNT), (RADUS)
```

XCNT and YCNT are arithmetic expressions which define the center of the circle and RADUS is an arithmetic expression defining the radius of the circle. The pair of coordinates must be enclosed in parentheses and separated by a comma. The coordinate pair may have a point variable substituted in their place. If the radius is negative, it will be made positive before plotting the circle.

If the X and Y scaling constants are unequal, the same convention applies to RADUS as to DIST in the PLOT VECTOR statement previously discussed. The actual radius of the resulting circle will be determined by the smaller of the scaling constants. The following is an example of the PLOT CIRCLE statement where the scaling constants are equal:

```
PLOT CIRCLE (7.5,5.5), (4.0)
```

As a result of this statement, a circle with a radius of four units is drawn with center at the point (7.5,5.5).

PLOT ARC STATEMENT

This statement causes an arc of a circle to be drawn. The PLOT ARC statement takes the following form:

```
PLOT ARC (XCNT,XCNT), (XARC,YARC), (RADN)
```

XCNT and YCNT are arithmetic expressions defining the coordinates of the center of the circle of which the arc is a part. XARC and YARC are arithmetic expressions indicating the coordinates of the point on the arc where plotting is to begin. The two pairs of coordinates must be enclosed in parentheses and separated within the parentheses by commas. A point variable may be substituted for either of the coordinate pairs. A comma follows each pair of coordinates. RADN is an arithmetic expression which specifies the size of the arc in radians and must be enclosed in parentheses. If the value of RADN is positive, the arc will be drawn counterclockwise from the point indicated by the second pair of coordinates. If the value of RADN is negative, the arc will be drawn clockwise from that point. If the value of RADN is greater than 6.2831853 or less than -6.28319531, which indicates an arc greater than a circle, only one circle will be drawn.

The following is an example of the PLOT ARC statement:

```
PLOT ARC (7.0, 5.0), (9.5, 5.0), (3.1415927).
```

This statement produces a semicircle concave in the negative Y direction with center at (7.0, 5.0) and radius of two and one-half units. This statement allows the programmer to draw any segment of a circle desired.

PLOT CHARACTER STATEMENT

The PLOT CHARACTER statement causes the plotting of characters from the set defined by the language in Chapter I. This statement provides a means of labeling the drawing with alphameric information. The statement appears in the following form:

```
PLOT CHARACTER (XC,YC), (SIZE), (ORNT), List.
```

XC and YC are arithmetic expressions indicating where the lower left-hand corner of the first character is to be plotted. SIZE is an arithmetic expression indicating the height of the characters to be plotted. The value of SIZE must be greater than or equal to 0.1 and less than or equal to 9.9. If SIZE is less than 0.1 or greater than 9.9, the characters will be drawn with a vertical size of 0.1 inch and 9.9 inches, respectively. ORNT must be an integer constant of +1, -1, +2, or -2. The characters will be drawn parallel to the X axis in the positive X direction if the constant is +1 and in the negative X direction if the constant is -1. If the constant is +2, the characters will be drawn parallel to the Y axis in the positive Y direction and if the constant is -2, the characters will be drawn in the negative Y direction.

The list which follows the orientation constant specifies format information and variables to be alphamerically plotted. This list follows the same rules as the format/variable list discussed in Chapter IV with the added restriction that the solidus is not allowed as a format specification. The following is a part of a program:

```
DEFINE RADIANS (TRAJ, 41.0, 25.0)

PLOT CHARACTER (150.0, -120.0), (.4), (+1),
     (18HTRAJECTORY ANGLE = ,F8.5), TRAJ.
```

This last statement causes the following information to be plotted alphamerically in the positive X direction beginning at the point (150.0, -120.0) in characters four-tenths of an inch in height:

```
TRAJECTORY ANGLE = .72286.
```

PROCEDURE, CALL AND REPEAT STATEMENTS

PROCEDURE STATEMENT

The PROCEDURE and corresponding END OF PROCEDURE statements define a subprogram which will be stored on disk and will be available for calling by future programs. The PROCEDURE statement is of the following form:

```
PROCEDURE NAME (VAR1, VAR2, VAR3, ... , VARN).
```

NAME is the unique name by which the main calling program refers to this subprogram. This name must consist of from one to five alphabetic and/or numeric characters of which the first must be alphabetic. VAR1, VAR2, VAR3, ... , VARN are variable names which will have values assigned to them at the time the procedure is called by the main program and may be used accordingly in the subprogram itself.

The END OF PROCEDURE defines the terminal point of the procedure and generates the necessary instructions to return control to the calling program. This statement may appear with or without the procedure name as follows:

END OF PROCEDURE

or END OF PROCEDURE NAME

If the name is included it must be the same as in the PROCEDURE statement.

The following is an example of a subprogram which calculates the volume and surface area of a cylinder when the radius and length of the cylinder are given:

```
PROCEDURE CYLIN (RADUS, LONG, VOL, SURAR)

VOL = LONG*3.1415927*RADUS**2

SURAR = LONG*3.1415927*2.*RADUS + 2.*3.1415927*RADUS**2

END OF PROCEDURE CYLIN
```

RADUS and LONG correspond to the radius and length of the cylinder, respectively, and are supplied to the subprogram by a CALL statement in the main program. VOL and SURAR correspond to the volume and surface area of the cylinder, respectively, which are calculated by the subprogram and returned to the main program through the CALL statement. In this way, frequently used computations may be formulated once by a PROCEDURE and then used by any future programs which require such formulas through the use of a CALL statement in the main program.

CALL STATEMENT

The CALL statement is the method in which the programmer calls a subprogram in the main program, provided that this subprogram has already been processed as a procedure. The CALL statement is of the following form:

```
CALL NAME (EXPR1, EXPR2, EXPR3, ... , EXPRN).
```

NAME is the name of the desired PROCEDURE and must therefore conform to the same roles as the NAME in the PROCEDURE statement. EXPR1, EXPR2, EXPR3, ... , EXPRN are arithmetic expressions the value of which will be assigned to the corresponding variables in the PROCEDURE list. If variables appear in the CALL list, they must agree in mode with the corresponding variable in the PROCEDURE list and these values will be returned to the main calling program. A subprogram must not contain

a CALL statement itself.

A variable in the main program and a variable in the subprogram which have the same variable name are treated as different entities unless they are defined to be the same by the CALL and PROCEDURE statements.

The following segment of a program calls the subprogram which calculates the volume and surface area of a cylinder defined in the preceding section:

```
READ, (2F10.4), R, L

CALL CYLIN (R, L, V, S)

PRINT, (4F10.4), R, L, V, S.
```

These statements read the value of the radius and length of a cylinder, call the subprogram to perform the volume and surface area calculations, and print the value of the radius, length, volume, and surface area of the cylinder.

REPEAT STATEMENT

The REPEAT and corresponding END OF REPEAT statements define the limits on a series of statements which are to be executed repetitively, called a REPEAT sequence. A REPEAT sequence may be completely contained within another REPEAT sequence up to five levels. This arrangement of REPEAT sequences is called nesting. The innermost REPEAT and END OF REPEAT statements define the beginning and end of the same REPEAT sequence, and so on working outward to the outermost REPEAT sequence. This may be thought of in terms of a last-in-first-out push-down list in which succeeding REPEAT statements cause the list to be pushed down with the current REPEAT statement placed at the top of the list. An END OF REPEAT effects a match or correspondence between itself and the REPEAT which is at the top of the push-down list which, in turn, defines a REPEAT sequence.

Within each REPEAT sequence there must be an IF statement defining the conditions on which control of the program will be transferred to a statement outside of the loop. Otherwise, there is no way to exit from the loop to execute statements which follow and thus the same REPEAT sequence will be performed indefinitely to the exclusion of the remainder of the program. The following is an example of the proper use of REPEAT sequences:

```
SUM = 0.0
@A = 0
REPEAT
@A = @A + 1
@B = 0
REPEAT
@B = @B + 1
SUM = SUM + VALUE (@A,@B)
```

```
    IF (@B .LE. 10), THEN CONTINUE, ELSE GO TO 15
    END OF REPEAT
 15 IF (@A .LE. 10), THEN CONTINUE, ELSE GO TO 25
    END OF REPEAT
 25 PRINT, (E14.8), TOTAL.
```

VALUE (@A, @B) defines a member of an array which is to be discussed in the following chapter. This portion of a program sums the values in a ten by ten array called VALUE and stores the results in SUM which in turn is printed out. In this example, the second subscript varies more rapidly than the first subscript.

ARRAY, DELETE, AND END OF PROGRAM STATEMENTS

ARRAY STATEMENT

The ARRAY statement defines many variables by the same name. The different variables are distinguished from each other by a number which follows the variable in parentheses. The array may be integer or real and may be of one, two or three dimensions. One or more arrays separated by commas may be defined in each ARRAY statement. The ARRAY statement may take any of the following forms:

```
    ARRAY NAM1 (C1, C2, C3)

 or ARRAY NAM2 (C4, C5)

 or ARRAY NAM3 (C6)
```

NAM1, NAM2, and NAM3 are arrays which have three, two, and one dimensions, respectively. C1, C2, C3, C4, C5, and C6 are integer constants indicating the size of each dimension. When these array variables or subscripted variables are used in other than an ARRAY statement, C1, C2, C3, C4, C5, and C6 may be either integer or variables. Consider the following examples:

```
    ARRAY ONE (7), TWO (5, 5), THREE (2, 2, 2)
```

ONE is an array of one dimension with seven elements. TWO is a five by five array of two dimensions. THREE is a two by two by two array of three dimensions.

DELETE STATEMENT

The DELETE statement causes the variable names which follow in a list, whether they be simple, subscripted, or point variables, to be eliminated from the program list of variable names and makes this storage area

available for new variables to be defined by the program. An array is deleted by using the array name without dimensioning information in the list of variables to be deleted. Single elements of an array can not be deleted. For example:

DELETE YES, NO, MAYBE.

This statement causes the simple variables, arrays, and/or point variables with the names YES, NO, and MAYBE to be removed from the program list of variable names.

END OF PROGRAM STATEMENT

This statement indicates to the interpretive processor that this is the end of this program. The processor then initialized itself in preparation for another program which may follow.

SUMMARY

With the ability to orient the plotting surface and to reference a pair of coordinates by one name, EZPLOT, allows a greater number of options that are not presently available in Fortran subroutines. Some of the other advantages of EZPLOT are its capability of drawing a line given a point, the angle, and the length of the line and plotting a circle or an arc of a circle in an efficient manner by use of specilized routines. Drawing a circle in Fortran involves using one of the library subroutines which themselves require a relatively long time to execute. The Fortran programmer must sacrifice speed to obtain accuracy or must be satisfied with a 'circle' with straight sides which are quite evident to insure speed. The circle and arc drawing routines in EZPLOT make use of certain relationships and approximating procedures which improve the appearance and minimize the time required in drawing a circle or arc. Also the form of EZPLOT statements is more meaningful to the programmer with its English-like structure as opposed to the cumbersome CALL PLOT statement in Fortran.

At the University of Southwestern Louisiana, the Computing Center Staff are interested in making the plotter available to as many areas of study as possible. A subset of EZPLOT is being implemented to help fill this interest. This subset is being written for an IBM 1620 computer, Model I, with indirect addressing, hardware divide, and having the move flag, transfer numeric strip and fill instructions. The additional hardware requirements are 40,000 core positions, a 1311 disk drive, a 1622 card reader/punch unit or equivalent and a 1626/1627 plotter.

The implemented language consists of READ and ACCEPT statements explained in this chapter, DEFINE and PLOT-DRAW statements as explained earlier with a variation of PLOT CHARACTER which is explained in this chapter. No arithmetic is defined in the implemented language except where coordinates

are required. In this case, the only arithmetic allowed is a point variable plus or minus a constant or variable, where the point variable has appeared as the first argument of a DEFINE POINT statement. If the letter 'T' is found in column six subsequent source statements are read from the typewriter or read from the card reader if the letter 'C' is found. All source statements must have a 'commercial at' sign [@] in the first position of each input record and may occupy up to five cards with a dollar sign as the terminal character in each statement.

The READ and ACCEPT statements cause data to be read by the card reader and typewriter, respectively. These statements are of the following form:

READ, VAR1, VAR2, VAR3, ... ,VARN

ACCEPT, VAR1, VAR2, VAR3, ... ,VARN.

These statements cause the variables VAR1, VAR2, VAR3, ... ,VARN to have assigned to them the numeric values present on the input record. The variable names in this list may consist of from one to five alphabetic and/or numeric characters of which the first must be alphabetic. A variable name may be split between two source input records. A single READ or ACCEPT statement reads a single data input record.

The data input record which is a maximum of eighty positions in length, consists of numeric data with each pair of items separated by one or more blanks. This numeric data may have a preceding plus or minus sign and a decimal point. If there is no sign explicitly indicated, the number is assumed to be positive. The decimal point is assumed to be immediately to the right of the last digit if it is not included.

The PLOT CHARACTER statement causes a string of characters to be plotted and is of the following form:

PLOT CHARACTER (XC,YC), (ORNT) $.

XC, YC, SIZE, and ORNT have the same function as in the PLOT CHARACTER discussed earlier. The dollar sign [$] indicates the end of source statement. On succeeding source input records, the contents of positions seven through seventy-two inclusive are plotted until a record mark [≠] (0, 2, 8 punch on card) is encountered. These are defined as source/data records and must have a 'commercial at' sign in position one. The PLOT CHARACTER statement and its associated source/data records must not occupy more than five source input records. For example:

PLOT CHARACTER (2.0, -7.0), (.4), (+1) $

VELOCITY IN FEET/SECOND. ≠

This causes the second input record to be plotted excluding the record mark with the lower left-hand corner of the letter V to be plotted at position (2.0, -7.0) on the graph. The characters will be four-tenths of an inch in height and will be plotted parallel to the X axis in the positive X direction.

Finally the END statement with the word 'END' beginning in column seven signifies the end of the program and deletes all of the variable names in the symbol table. As a result, subsequent programs are unable to use information which has been stored by a previous program.

<u>Name of Prime Committee</u>:

<u>Subject</u>: Relocatable Data-Conversion Subroutines for 1620 SPS II

<u>Speaker's Name</u>: W. N. Tuttle

<u>Representing</u>: General Radio Company

<u>Mailing Address and Phone Number</u>: 22 Baker Avenue, West Concord,
Massachusetts 01781 369-4400

<u>Day and Time of Speech and Session Number</u>: Monday, November 28, 1966,
3:30-5:00 p.m., Session M-4.8

<u>Number of Pages of Text</u>: 5. No graphics

RELOCATABLE DATA-CONVERSION SUBROUTINES FOR 1620 SPS II

W. N. Tuttle

General Radio Company, West Concord, Massachusetts

ABSTRACT

Much of the programming time with SPS is usually taken up with handling input and output data-conversion problems. A set of three general-purpose subroutines is described to reduce this labor, and make possible shorter programs. These are IFL, to convert from integer to floating-point, AFL, to convert from an alphameric input field to floating-point, and FLA, to convert from floating-point to an alphameric output field. FLA gives unbiased rounding and can, as an option, change the number of decimal places, as required, to fit a wide range of output values into a field of given width. Only a single macro instruction in the mainline program is required for each job. Each subroutine is loaded automatically when its macro is used in a program. IFL uses 859 core positions, AFL 1371, and FLA 1977. Indirect addressing is the only required special feature. The subroutines can be used only with the fixed-length subroutine set and will not work without modification in Monitor SPS II-D.

Introduction

Data conversion and the handling of input and output data are taken for granted in Fortran programming. This is an important reason for the popularity of Fortran and similar compilers. Programmers are frequently willing to put up with the serious limitations of these compilers rather than go through the chores of data conversion which are required when SPS is used.

Examples of the limitations are the restriction of integer computations in non-monitor Fortran to numbers of four digits or less, the unavailability of proper rounding procedures, and the inflexibility of the output formats.

In SPS even quite complicated mathematics is not difficult to write, and the programs are usually better and run faster than those in Fortran. The input, output, and data-conversion problems, however, take a disproportionate amount of effort and keep many programmers from profiting from the very great flexibility and efficiency of SPS.

The present paper describes three relocatable subroutines which have been written to handle some of the most frequent and most bothersome data-conversion problems. The first, AFL, is for alphameric to floating-point conversion, the second, IFL, is for integer to floating-point conversion, and the third, FLA, is for floating-point to alphameric conversion and for automatically shifting the decimal point in any of several ways to accommodate as wide a range of values as possible in an output space of fixed width. The latter subroutine was described a year ago at the New York meeting.[1]

All three of these subroutines also solve the problem of rounding to the number of significant figures retained at any stage of the process. Rounding is not difficult when a fixed number of decimal places is retained, as in many output specifications, but it requires careful programming if it is to work in all the situations that can arise in general data-conversion problems. Proper rounding can significantly reduce truncation errors. The procedures give unbiased rounding, which is rounding in such a way that there is no tendency for the rounding process to increase or decrease the average of a series of values. Rounding is discussed in detail in the earlier paper and will not be considered further here.

AFL Subroutine, Alphameric to Floating-Point Conversion

The first problem in a program is usually to get data into the computer, and AFL, alphameric to float, is for alphameric read-in. If a number is punched in a card and read into the computer, what appears in core will be the alphamerically coded version of the number. This must be converted to the standard floating-point form. Thus if -12.073 is punched in the card,

20717203707773

will be read into the input area of core. It is desired to store the floating-point equivalent

$$\bar{1}2073000\bar{0}\bar{0}2$$

at some other specified address. The conversion is a bothersome job even when the input format is made as simple as possible. But restricting the format complicates the problem of punching data cards and makes it hard to cover wide ranges of the input variables. It is hoped that AFL will fill the need for a subroutine that will take care of this job with a minimum of restrictions on the way the data cards are punched.

1 W.N. Tuttle, "A Relocatable SPS Subroutine for Editing and Rounding Output Data for Scientific Tables and Similar Applications", Proceedings of Common Joint Eastern Midwestern Region, October 6, 7, 8 1965, pp. 535-540.

The AFL subroutine has this "free format" and since it is relocatable, it doesn't clutter up the source program. The macro, AFL, causes the subroutine to be loaded automatically, and the necessary information is given by four operands. Three things must be specified because of the nature of the job. These are the address of the input area, its alphameric width, and the address where the floating-point equivalent is to be generated. A fourth operand, which may be omitted, is for the user's convenience. It makes it possible in the conversion process to multiply the number by any desired power of 10. This, for example, would permit data to be read in parts per million without complicating the card punching.

The form of the macro instruction is

AFL FLOT,FLD,W,PWR

This says to take the alphameric-coded number from FLD, which has width W, multiply by ten to the power PWR, convert to floating point and store at address FLOT. The PWR operand can be omitted if the third comma appears. Note that the order of the operands corresponds to that used in most of the regular instructions. The input data is at the address given by the second operand and the result of the operation is transmitted to the address given by the first operand.

The input format is free in that the width of the input area is specified but the program will accept anything that makes sense within this space. Obvious errors are rejected, and numbers in exponential form are not accepted. The number can start and end anywhere within the area, the decimal point can be used or not as desired, and leading or following zeros make no difficulty. The program rejects numbers with blank spaces between characters, except that the sign does not have to be adjacent to the first character. It rejects two decimal points or two signs.

AFL as well as IFL and FLA, the two other subroutines of the group, is handled just like any other relocatable subroutine of the regular set. A library card is inserted in the SPS II processor deck and the subroutine, with header and trailer cards, is added to the fixed-length subroutine set. AFL uses indirect addressing but requires no other special features. It uses 1371 core positions.

IFL Subroutine, Integer to Floating-Point Conversion

This subroutine forms at a specified address the floating point equivalent of an integer at another address. It leaves the integer intact at its original location so that it is available for further use if desired. IFL can also be used for handling data read in numerically, provided that the proper flags are either read in or added in the program to make the input numeric field a proper integer field. This subroutine, like AFL, can multiply the number by a desired power of ten, and it rounds to 8 digits if the integer field exceeds this length.

This subroutine requires only three operands, as it is not necessary to specify the width of the integer field. The subroutine checks for the location of the flag that defines the start of the field. The instruction, therefore, appears as

IFL FLO,FLD,PWR

Here the integer field at FLD is reproduced at FLO in its floating-point equivalent after multiplying by 10 to the power PWR. As with AFL, the PWR operand can be omitted if the last comma is retained.

IFL is particularly useful when part of a computation is done in integers and the rest in floating point. For example, an integer in the product area can be rounded, converted and transferred in a single operation.

IFL uses indirect addressing and requires 859 core storage positions.

FLA Subroutine, Floating-Point to Alphameric Conversion

This subroutine, described in detail in the earlier paper, is for the conversion and editing of output data so that mathematical tables, tables of scientific data, etc., can be printed in final form directly from the card output without additional editorial effort.

One function performed by FLA is to fit the number to the specified column width by decimal-point shifting so that the maximum range of a quantity can be accommodated without resorting to exponential format. Several options are available in this part of the process including that of omitting the decimal point and rounding to the nearest integer. The type of output is controlled by a code operand.

In all the output options unbiased rounding is used whenever the number of significant figures retained is less than eight. Biased rounding or "up-rounding" is available as an option for applications such as accounting, where it is traditional. This option is also controlled by the code operand.

The macro instruction for FLA has five operands and is similar to that for AFL in that the first three operands are the address to which the converted number is to be transmitted, the address of the number to be converted, and the width of the alphameric field. The fourth and fifth operands are the nominal or uncorrected number of decimal places and the code specifying the type of conversion. Thus

FLA A,B,C,D,E

means that the floating-point number at B is to be converted and transmitted to the alphameric area A. The alphameric width at A is C, and the type of conversion is controlled by the code operand E.

In normal use of the subroutine, the code operand E is omitted, giving automatic right shift of the decimal point to accommodate large numbers and automatic left shift so that as many significant figures as possible can be retained in an output area of given width.

Automatic right decimal-point shift is illustrated by the following table, where the width C is six and the normal number of decimal places D is two:

```
 12.35
123.46
1234.6
12346.
```

Note that two decimal places are kept until the width is exceeded.

For smaller numbers within the same width, the automatic left decimal-point shift would give

```
 12.35
  1.23
   .12
  .012
 .0012
.00012
```

Either or both shifts can be omitted if desired. A 1 in the tens position of the code operand causes omission of the left shift, a 1 in the hundreds position causes omission of the right shift. Omission of the left shift might be specified in the example above if figures beyond the second decimal place were not of interest.

The code operand can also, by a 1 in the thousands position, call for conventional up-rounding instead of unbiased rounding. Thus the operand 1010 calls for up-rounding with omission of the left shift.

Omission of the decimal point and rounding to the nearest integer is available as an additional option. This is called for by making the D operand, which specifies the number of decimal places, equal to -1.

A final feature of the subroutine, always in effect, is that an additional position in the output area is made available for positive numbers because no space is required for the sign.

AFL requires indirect addressing and uses 1977 core positions.

A SYSTEM OF REMOTE TIME SHARING THE IBM 1620/1710 COMPUTER SYSTEM BY USE OF THE IBM 1710 SYSTEM AND REMOTE TELETYPE CONSOLES.

by

Donald Box
Assistant Professor
Electrical Engineering
Tennessee Technological University
Cookeville, Tennessee

and

Hugh B. Kerr
Director
The D. W. Mattson Computer Center
Tennessee Technological University
Cookeville, Tennessee

Phone Area 615 526-9521 Ex. 325

Monday, November 28, 3:30 - 5:00 p. m.
M-4.8

Text - 12

Graphics - 2

SUBJECT ... A SYSTEM OF REMOTE TIME SHARING THE IBM 1620/1710 COMPUTER SYSTEM BY USE OF THE IBM 1710 SYSTEM AND REMOTE TELETYPE CONSOLES.

DESCRIPTION ...... IN JANUARY 1966, A REQUEST WAS MADE OF THE D W MATTSON COMPUTER CENTER AT TENNESSEE TECHNOLOGICAL UNIVERSITY BY THE ELECTRICAL ENGINEERING DEPARTMENT TO PROVIDE THE NECESSARY HARDWARE AND SOFT-WARE TO PERMIT THE IBM 1710-1620 SYSTEM TO BE USED ON A PRIORITY INTERRUPT BASIS, THE INTERRUPT AND INPUT BEING MADE FROM REMOTE TELETYPE TERMINALS. DUE TO ECONOMIC CONSIDERATIONS, IT WAS IMPERATIVE THAT THIS BE DONE WITHOUT THE OUTLAY OF MONEY FOR THE MORE ELABORATE INTERFACE HARDWARE AVAILABLE FROM IBM (AND OTHER MANUFACTURERS) TO HANDLE THE NECESSARY DATA TRANSMISSIONS.

THE EQUIPMENT ALREADY ON HAND WAS AS FOLLOWS ......

(A) IBM 1620 MOD I
- (1) INDIRECT ADDRESSING
- (2) FLOATING POINT HARDWARE
- (3) ADDITIONAL INSTRUCTIONS
- (4) BASIC INTERRUPT
- (5) 1710 INSTRUCTIONS

(B) IBM 1710 CONTROL SYSTEM WITH 1711 AD CONVERTER AND 1712 MULTIPLEXER AND TERMINAL UNIT
- (1) 10 CONTACT OPERATES (NON-LATCHING)
- (2) 4 GROUPS OF CONTACT SENSE
- (3) BASIC INTERRUPT
- (4) REAL TIME CLOCK
- (5) OTHER FEATURES IMMATERIAL TO THIS APPLICATION

(C) IBM 1311 DISK MODULE

(D) IBM 1623 WITH 20 K STORAGE
(GIVING A TOTAL OF 40 K COMPUTING SYSTEM)

(E) IBM 1622 CARD READ PUNCH

SINCE A GREAT DEAL OF DIFFICULTY HAD BEEN ENCOUNTERED IN IMPLEMENTING THE IBM 1710 FORTRAN EXECUTIVE SYSTEM, AND FURTHURMORE SINCE THE 1710 EXECUTIVE DID NOT PERMIT MONITOR COMPILATIONS, ETC., IT WAS DECIDED TO LOOK ELSEWHERE FOR AN INTERRUPT EXECUTIVE SYSTEM. INFORMATION FROM IBM WAS OBTAINED CONCERNING AN INTERRUPT EXECUTIVE SYSTEM WRITTEN BY MR. EARL SPRAKER (THEN OF THE IBM ATLANTA OFFICE) FOR THE GEORGIA STATE HIGHWAY DEPARTMENT TO FACILITATE THE AUTOMATIC POLLING OF TRAFFIC OVER THE STATE OF GEORGIA.

WITH THE COOPERATION OF THE GEORGIA STATE HIGHWAY DEPARTMENT, SOURCE DECKS WERE OBTAINED FOR THIS EXECUTIVE SYSTEM THE NECESSARY DELETIONS AND ADDITIONS MADE TO RECOGNIZE A REMOTE USER INTERRUPT(OR MANUAL ENTRY INTERRUPT FROM THE COMPUTER CENTER), SAVE THE TOP 20 K OF CORE, BRING IN A ... BOOT STRAP ... (OR ANALYZER) PROGRAM AND BRANCH TO THIS PROGRAM. THE...BOOT STRAP...PROGRAM SAVES THE BOTTOM 20 K OF CORE, ANA-

LYZES THE TYPE OF INTERRUPT, DETERMINES THE PROGRAM (ONE OF SEVERAL IN THE EXECUTIVE LIBRARY) BRINGS IN THE PROGRAM AND BRANCHES TO IT. UPON CONCLUSION OF THE USER CALLED INTERRUPT PROGRAM, THE BOTTOM 20 K IS RESTORED, A BRANCH BACK INTO THE EXECUTIVE PROGRAM IS EFFECTED, THE TOP 20 K IS RESTORED AND THE PROGRAM WHICH HAD BEEN INTERRUPTED IS RESUMED.

AS CAN BE SEEN BY THE ABOVE DESCRIPTION, ANY KIND OF OPERATION UNDER THE MONITOR 1 (VERSION 2) CAN BE OPERATING AS A ... BACKGROUND ... PROGRAM, PERMITTING INTERRUPT AT ANY TIME BY A REMOTE USER.

LIMITATIONS IMPOSED ON INTERRUPT PROGRAMS ....

(1) THE PROGRAM MUST OCCUPY NO HIGHER CORE ADDRESS THAN 34999. THIS IS NECESSARY BECAUSE OF THE BOOTSTRAP PROGRAM WHICH MUST RESIDE HERE UNTIL A BRANCH TO THE INTERRUPT PROGRAM IS EFFECTED. AFTER THE INTERRUPT PROGRAM IS LOADED, 35000 THROUGH 39999 IS AVAILABLE FOR USE BY THE INTERRUPT PROGRAM, OVERLAYING THE NOW USELESS BOOT STRAP PROGRAM.

(2) ALL INTERRUPT PROGRAMS MUST TERMINATE BY RESTORING THE BOTTOM 20 K OF CORE (NATURALLY, WITHOUT OVERLAYING THE STILL NEEDED INSTRUCTIONS IN THE INTERRUPT PROGRAM) AND THEN BRANCHING TO LOCATION 1044 IN THE EXECUTIVE PROGRAM, WHICH THEN RETURNS THINGS TO NORMAL BEFORE RETURNING TO THE BACKGROUND PROGRAM. THIS IS DONE, IN THE CASE OF A FORTRAN PROGRAM, BY UTILIZING THE ... BACK ... ROUTINE.

(3) THE USER MUST TAKE CARE NOT TO DESTROY THE WORK CYLINDER AREA OF DISK WHICH MAY CONTAIN INFORMATION IN USE BY THE BACKGROUND PROGRAM. THIS MEANS THAT LOCAL PROGRAMS MAY NOT BE USED (INASMUCH AS THEY ARE STORED IN CORE IMAGE AT LOAD TIME IN THE SCRATCH AREA OF DISK). IT ALSO MEANS THAT THE ... DEFINE DISK, SEEK, FETCH, ETC..... MAY NOT BE USED INASMUCH AS THEY INFER USE OF THE WORK CYLINDERS.

(4) THE PROGRAM CALLED BY THE USER MUST, OF COURSE, BE ONE STORED IN THE EXECUTIVE LIBRARY. THIS ELIMINATES THE POSSIBILITY OF FORTRAN OR SPS COMPILATIONS BEING DONE ON THE INTERRUPT BASIS FROM REMOTE TERMINALS.

MISCELLANEOUS PROBLEMS AND SOLUTIONS

(1) SLACK TAPE PROBLEM

INASMUCH AS THE REMOTE USER PUNCHES HIS INFORMATION DIRECTLY INTO PAPER TAPE (THE TAPE PUNCH BEING IN THE COMPUTER CENTER) AND THEN READS THE TAPE ON INTERRUPT BASIS, THE LENGTH OF SLACK TAPE CONTINUALLY BUILDS UP UNLESS ADEQUATE PROVISION IS MADE TO TAKE OUT THE SLACK. THIS IS ACCOMPLISHED IN THE ... BACK ... PROGRAM BY FIRST TURNING OFF THE TAPE PUNCH MOTOR, THEN READING TAPE UNTIL THE TAPE IS PULLED TIGHT, AT WHICH TIME THE PUNCH MOTOR IS RE-STARTED BEFORE RETURNING TO THE BACKGROUND PROGRAM THROUGH THE EXECUTIVE. OF COURSE, THIS CALLS FOR A

CONSIDERABLE AMOUNT OF VALUABLE COMPUTER TIME TO BE WASTED, BUT IS UNAVOIDABLE IN THE PRESENT PHYSICAL SETUP AT TTU. THE SEEMINGLY OBVIOUS SOLUTION WOULD BE TO PUNCH THE TAPE REMOTELY, INTERRUPT THE PROGRAM AND CALL FOR THE TAPE TO BE READ DIRECTLY FROM THE REMOTE STATION. HOWEVER, THIS WOULD CALL FOR MULTIPLE ... HARD LINES ... TO BE INSTALLED BETWEEN THE REMOTE STATION AND THE CPU WHICH IS, TO US, IMPRACTICAL.

(2) PROBLEM OF MORE THAN ONE REMOTE STATION ATTEMPTING USE THE INTERRUPT LINE AT ONE TIME.

WITH THE PRESENT TTU SETUP, IT IS QUITE POSSIBLE FOR SOMEONE TO ATTEMPT INPUT FROM HIS REMOTE STATION AT THE SAME TIME ANOTHER PERSON IS IN THE PROCESS OF PUNCHING TAPE. WE MINIMIZE THIS PROBABILITY BY INSTALLING A SWITCH BOX AT EACH REMOTE STATION WITH REMOTE USER STATUS LIGHTS. THIS ... BLACK BOX ... CONSISTS OF THREE SWITCHES AND TWO LIGHTS AS FOLLOWS .....

READY LIGHT ... A LIGHT THAT INDICATES TO THE USER WHEN THE INTERRUPT LINE IS NOT BUSY AND READY FOR USE.

BUSY LIGHT ... A LIGHT THAT IS ON WHEN ANY USERS ... ON ... SWITCH IS ON.

ACTIVATE SWITCH ... TURNS ON THE TELETYPE AND ALLOWS THE USER TO MONITOR ALL MESSAGES ON THE INTERRUPT LINE. THE ... READY ... IS TURNED ON.

ON SWITCH ... TURNS OFF ALL ... READY ... LIGHTS ON THE LINE AND TURNS ON ALL ... BUSY ... LIGHTS INFORMING ALL STATIONS ON THE LINE THAT THE INTERRUPT LINE IS IN USE.

INTERRUPT SWITCH ... THIS IS A MOMENTARY CONTACT SWITCH THAT CREATES THE PROCESS INTERRUPT TO PLACE THE CPU IN INTERRUPT MODE, CAUSING THE USERS TAPE TO BE READ AND HIS PROGRAM EXECUTED.

RULES FOR INPUT

AS IN ALL COMPUTER WORK, THE RESULTS ARE ONLY AS GOOD AS THE INFORMATION FED TO THE SYSTEM. IN THE EVENT MISTAKES ARE MADE IN INPUT, PROVISION IS MADE IN THE INTERRUPT SYSTEM TO OUTPUT AN ERROR MESSAGE, IN CODE FORM, TO THE USER. THE ESTABLISHED ERROR CODES ARE IDENTIFIED LATER IN THIS DOCUMENTATION. IN THE EVENT THAT AN ERROR IS ENCOUNTERED, THE ERROR CODE IS OUTPUTTED ON THE USER TELETYPE CONSOLE, HIS JOB IS ABANDONED, AND CONTROL IS TRANSFERRED BACK TO THE EXECUTIVE PROGRAM WHICH THEN RETURNS TO THE JOB BEING DONE IN THE COMPUTER CENTER.

THE USER INPUT TO THE INTERRUPT PROGRAM WILL, OF COURSE, BE VARIABLE, DEPENDENT UPON HIS DATA REQUIREMENTS. THE BEGINNING

OF HIS INPUT MUST, HOWEVER, STRICTLY CONFORM TO THE FOLLOWING FORMAT ....

1ST CHARACTER - UPPER CASE SHIFT
2ND CHARACTER - 5/8 SYMBOL
3,4,5,6 CHARACTERS - DIM NUMBER OF THE PROGRAM HE WISHES TO CALL
....
....
....
(INCLUDE ALL ... CALL ALPHA ... CHARACTERS THAT YOU CHOOSE TO CALL AS INPUT, AND ALL VARIABLES THAT YOU WISH TO INTRODUCE BY THE ... CALL READ .... ROUTINE. BE SURE TO INCLUDE AS MUCH INPUT AS YOU INTEND TO CALL FROM THE PROGRAM)

A NUMERIC INPUT

ALL NUMERIC INPUT MUST BE FLOATING POINT NUMBERS, SUCH AS THE FOLLOWING EXAMPLES ......

.000165$
125.32$
126$
3$
.1635$

THE DECIMAL POINT, IF NOT INCLUDED, WILL BE ASSUMED TO BE AT THE RIGHT OF THE LAST DIGIT OF THE NUMBER. EACH NUMBER MUST BE FOLLOWED BY A $ (DOLLAR SIGN). THE INPUT OF EACH NUMBER WILL BE CALLED BY THE FOLLOWING FORTRAN STATEMENT .....

```
CALL READ(X)
```

WHERE .... X .... MUST ALWAYS BE THE SYMBOLIC NAME OF A FLOATING POINT NUMBER. ONLY ONE VARIABLE MAY BE READ BY ONE CALL READ STATEMENT. THE NUMBERS, WHEN READ FROM TAPE, WILL BE STORED AS 8 DIGIT MANTISSA FLOATING POINT NUMBERS AND STORED IN THE APPROPRIATE MEMORY LOCATION.

B NUMERIC OUTPUT

ALL OUTPUT WILL BE STANDARD E14.8 OUTPUT AND, THEREFORE, MUST BE A FLOATING POINT NUMBER. THE OUTPUT OF EACH NUMBER WILL BE CALLED BY THE FOLLOWING FORTRAN STATEMENT ......

```
CALL WRITE(X)
```

WHERE ... X ... IS THE SYMBOLIC LOCATION OF THE FLOATING POINT NUMBER TO BE OUTPUTTED.

IF YOU NEED CARRIAGE RETURNS OR LINEFEEDS ON THE TELETYPE, YOU MUST CALL FOR THEM BY USE OF THE ... ALPHA ... ROUTINE OR THE ... FORM ... ROUTINE. MAKE SURE THAT ALL INPUT HAS BEEN MADE BEFORE ATTEMPTING ANY OUTPUT WHATSOEVER.

C ALPHAMERIC INPUT/OUTPUT

IN ORDER TO PERMIT A VARIATION (FROM ONE PROGRAM EXECUTION TO ANOTHER) OF PROBLEM IDENTIFICATION OR OUTPUT IDENTIFICATION,

AN INPUT/OUTPUT STATEMENT IS PROVIDED AS FOLLOWS ....

```
CALL ALPHA(J,K)

WHERE J = 1 IF THE STATEMENT IS TO BE USED TO
            READ ALPHAMERIC INPUT, OR
          2 IF THE STATEMENT IS TO BE USED TO
            MAKE AN ALPHAMERIC OUTPUT
      K = A LOCATOR NUMBER (OR IDENTIFICATION
          NUMBER FOR THIS ALPHAMERIC INFORMATION)
```

AS AN EXAMPLE OF THE USE OF THIS ROUTINE, LET US SUPPOSE THAT YOU WISHED TO READ IN FROM TELETYPE TAPE, SOME ALPHAMERIC INFORMATION. YOU WOULD USE THE FORTRAN INSTRUCTION ...

```
CALL ALPHA(1,100)
```

THIS WOULD CALL FOR ALPHAMERIC INFORMATION PUNCHED INTO TAPE AND TERMINATED BY AN UPPER CASE SYMBOL FOLLOWED BY A DOLLAR SIGN TO READ INTO MEMORY AND STORED UNDER THE IDENTIFICATION NUMBER 100.

LATER ON IN THE PROGRAM, YOU WOULD WISH TO OUTPUT THIS INFORMATION ON YOUR TELETYPE CONSOLE. YOU WOULD DO THIS BY USE OF THE FOLLOWING FORTRAN STATEMENT ....

```
CALL ALPHA(2,100)
```

THE PREVIOUSLY READ ALPHAMERIC INFORMATION WOULD THEN BE PRINTED OUT ON YOUR TELETYPE CONSOLE. ALL CARRIAGE RETURNS, LINEFEEDS, BELLS, ETC, THAT YOU NEED ON THIS OUTPUT WOULD, OF COURSE, HAVE BEEN ENTERED AS CHARACTERS IN THE STORED ALPHAMERIC INFORMATION. A MAXIMUM OF 100 ALPHAMERIC CHARACTERS ARE ALLOWED IN ANY FORTRAN PROGRAM. THE ...CALL ALPHA... CAN BE USED ANY NUMBER OF TIMES AS LONG AS THE TOTAL CHARACTER COUNT DOES NOT EXCEED 100.

THE NUMBERS J AND K MAY BE EITHER SYMBOLIC OR ACTUAL, BUT IN EITHER CASE, MUST BE FIXED POINT NUMBERS OR VARIABLES.

REMINDER - THERE ARE NO AUTOMATIC LINEFEED OR CARRIAGE RETURNS PROVIDED BY THE ... ALPHA ... ROUTINE. THEREFORE, IF THEY ARE NEEDED, THEY MUST BE PROVIDED IN THE CHARACTERS THEMSELVES.

REMINDER - DO NOT ATTEMPT ANY OUTPUT UNTIL ALL INPUT HAS BEEN FINISHED.

D ALPHAMERIC OUTPUT OF STORED STATEMENTS

MANY TIMES, IT WILL BE USEFUL TO IDENTIFY ALPHAMERICALLY CERTAIN OUTPUT WITHOUT THE TIME CONSUMING USE OF THE... ALPHA .... READ ROUTINE. TO SUPPLY THIS NEED, THE .... FORM .... SUBPROGRAM HAS BEEN DEVISED. THIS ROUTINE IS CALLED BY THE FOLLOWING FORTRAN STATEMENT ....

```
CALL FORM(N1,A,B,C,D)
```

WHERE N1 IS THE NUMBER OF FLOATING POINT ARGUMENTS TO BE CONVERTED TO ALPHAMERIC OUTPUT (MAXIMUM VALUE OF 4)

A,B,C,D (A MAXIMUM OF 4) ARE FLOATING POINT

ARGUMENTS CONSISTING OF ALPHAMERIC PAIRS OF DIGITS TO BE OUTPUTTED.

AS AN EXAMPLE, LET US SUPPOSE THE STATEMENT ... ROOTS ARE REAL ... IS TO BE OUTPUTTED ON TELETYPE TAPE. THIS MAY BE ACCOMPLISHED BY THE FOLLOWING FORTRAN STATEMENTS.....

```
A=.59565663  (ROOT)
B=.62004159  (S AR)
C=.45005945  (E RE)
D=.41530000  (AL  )
CALL FORM(4,A,B,C,D)
```

ANY CARRIAGE RETURNS, LINE FEED, BELLS, ETC MUST BE PROGRAMMED BY THE USER.

E CORRECTING TELETYPE ERRORS

IN THE EVENT THAT YOU MAKE A MISTAKE AND RECOGNIZE THE MISTAKE IMMEDIATELY, YOU MAY ERASE THE FAULTY CHARACTER BY FOLLOWING IT IMMEDIATELY BY THE FOLLOWING ...

UPPER CASE
1/4 SYMBOL
PROPER CHARACTER, ETC

THIS WILL CAUSE THE FAULTY CHARACTER TO BE ERASED AND REPLACED BY THE PROPER CHARACTER. IN THE CASE OF DATA ENTRY, THIS MUST, OF COURSE, BE DONE BEFORE THE INCLUSION OF A DOLLAR SIGN.

F GENERAL INSTRUCTIONS FOR WRITING INTERRUPT PROGRAMS

THE USER PROGRAMS MAY BE WRITTEN IN FORTRAN, SPS OR MACHINE LANGUAGE. ANY PROGRAM WRITTEN TO BE USED ON INTERRUPT BASIS SHOULD FOLLOW THE GUIDE LINES OUTLINED BELOW ....

1 SINCE I/O OPERATIONS ARE INHERENTLY SLOW ON TELETYPE, YOU SHOULD LIMIT YOUR INPUT AND OUTPUT TO THE MINIMUM. YOU MAY CALL THE ... READ ... AND ... WRITE ... ROUTINE AS MANY TIMES AS YOU DESIRE BUT YOU MAY STORE AND CALL WITH THE ALPHA ROUTINE A LIMIT 100 CHARACTERS

2 SINCE YOU ARE INTERRUPTING ANOTHER PROGRAM EXECUTING, YOUR INTERRUPT PROGRAM SHOULD BE REASONABLY SHORT IN EXECUTION. A REASONABLE MAXIMUM TIME FOR AN INTERRUPT PROGRAM TO REQUIRE MIGHT BE CONSIDERED TO BE 4 MINUTES, ALTHOUGH MOST INTERRUPT PROGRAMS WOULD, HOPEFULLY, BE OF SHORTER DURATION.

3 YOUR PROGRAM MAY BE NO LONGER THAN 35,000 DIGITS IN SIZE

4 SINCE THE INTERRUPT EXECUTIVE IS BRINGING IN YOUR PROGRAM IN CORE IMAGE FORM (TO SAVE TIME) YOU MAY NOT USE ... LOCAL... PROGRAMS (THE LOCAL PROGRAMS ARE NOT STORED IN

THE DISC SCRATCH AREA).

5 YOU MUST ALWAYS, REPEAT ALWAYS, TERMINATE YOUR INTERRUPT PROGRAM WITH THE FORTRAN INSTRUCTION ...

CALL BACK

WHICH WILL RETURN CONTROL TO THE INTERRUPT EXECUTIVE PROGRAM AND ALLOW RESUMPTION OF THE WORK THAT WAS INTERRUPTED.

ERROR MESSAGES

ER 1 THE EXECUTIVE PROGRAM HAS READ IN 400 CHARACTERS IN AN ATTEMPT TO FIND YOUR DIM ENTRY NUMBER. IT MAY BE THAT YOU HAVE NOT ENTERED AN UPPER CASE SYMBOL IMMEDIATELY FOLLOWED BY THE 5/8 SYMBOL. IT MAY ALSO BE THAT TOO MUCH SLACK TAPE HAS BEEN ACCUMULATED BETWEEN THE TAPE READER AND THE TAPE PUNCH. IN EITHER EVENT, THE EXECUTIVE PROGRAM RINGS THE BELL 24 TIMES (TO GET THE ATTENTION OF THE COMPUTER CENTER PERSONNEL), TYPES OUT THE MESSAGE ... ER1 .. AND ABANDONS YOUR JOB (IT ALSO TYPES OUT A MESSAGE ON THE COMPUTER CONSOLE TYPEWRITER TELLING THE COMPUTER CENTER PERSONNEL TO TAKE THE SLACK OUT OF THE TAPE). IT WILL DO YOU NO GOOD TO RE-INTRODUCE YOUR DATA UNTIL THIS DIFFICULTY IS CORRECTED.

ER 2 TOO MANY CHARACTERS HAVE BEEN READ IN THE ... READ ... ROUTINE

ER 5 BAD DIM ENTRY NUMBER (THERE IS NO PROGRAM STORED USING THIS DIM ENTRY NUMBER)

ER 6 TWO POSSIBILITIES EXIST ... (1) NO SUCH ALPHAMERIC FORMAT NUMBER IS STORED AS THAT CALLED FOR IN THE ... CALL ALPHA .. ROUTINE, OR (2) TOO MANY ARGUMENTS ARE CALLED FOR IN THE ... CALL FORM ... ROUTINE(A MAXIMUM OF 4 ARGUMENTS ARE ALLOWED).

ER 7 MORE THAN 100 ALPHAMERIC CHARACTERS OF STORAGE ATTEMPTED IN CALL ALPHA STATEMENT.

ER 8 TWO POSSIBILITIES EXIST ... (1) WRONG CODE FOR ALPHA INPUT OR OUTPUT (SOMETHING OTHER THAN A 1 OR 2) OR (2) YOU HAVE USED AN INVALID CHARACTER IN THE ... CALL FORM ... ROUTINE.

ER 9 YOU HAVE TRIED A CALL ALPHA OUTPUT WITH NO ALPHA INPUT.

ACCEPTABLE ALPHAMERIC CODES FOR CALL FORM ROUTINE

THE CHARACTERS THAT WILL BE PRINTED OUT ON THE TELETYPE TYPEWRITER WILL, OF COURSE, DEPEND UPON WHETHER THE TYPEWRITER IS IN

UPPER OR LOWER CASE SHIFT. THE CODES FOR UPPER AND LOWER CASE SHIFT AS FOLLOWS ....

37	UPPER CASE SHIFT
38	LOWER CASE SHIFT

DO NOT FORGET THAT IT IS UP TO THE USER TO SPECIFY ALL CARRIAGE RETURNS AND LINEFEEDS ON THE TELETYPE. THESE TWO FUNCTIONS MAY BE PERFORMED WHILE IN EITHER UPPER OR LOWER CASE SHIFT AND ARE AS FOLLOWS ....

36	TELETYPE CARRIAGE RETURN
35	TELETYPE LINE FEED

YOU MUST REMEMBER THAT THE TELETYPE TYPEWRITER STAYS IN THE UPPER OR LOWER CASE SHIFT, ONCE PLACED THERE AND THE USER MUST PROGRAM ANY AND ALL SHIFT CHANGES. THE ONE EXCEPTION TO THIS IS THE SPACE (00) WHICH USUALLY (DEPENDING UPON WHAT TYPE TELETYPE YOU HAVE) CAUSES THE TELETYPE TO DROP INTO LOWER CASE SHIFT AND TO STAY THERE UNTIL UPPER CASE SHIFT IS PULSED. FOLLOWING ARE THE ACCEPTABLE CHARACTERS THAT MAY BE USED IN THE ... CALL FORM ... SUB PROGRAM ....

UPPER CASE SHIFT		LOWER CASE SHIFT	
03	DECIMAL POINT (.)	41	A
04	CLOSED PARENTHESES)	42	B
20	MINUS SIGN	43	C
21	SLASH /	44	D
23	COMMA (,)	45	E
24	OPEN PARENTHESES (	46	F
70	ZERO 0	47	G
71	NUMBER 1	48	H
72	NUMBER 2	49	I (LETTER I, NOT ONE)
73	NUMBER 3	51	J
74	NUMBER 4	52	K
75	NUMBER 5	53	L
76	NUMBER 6	54	M
77	NUMBER 7	55	N
78	NUMBER 8	56	O (LETTER O)
79	NUMBER 9	57	P
00	SPACE	58	Q
		59	R
		62	S
		63	T
		64	U
		65	V
		66	W
		67	X
		68	Y
		69	Z
			SPACE

SAMPLE PROGRAM

```
C      PROGRAM TO SOLVE THE ROOTS OF A QUADRATIC
   8 PAUSE
   1 CALL READ(A)
     CALL READ(B)
     CALL READ(C)
  50 CALL ALPHA(1,100)
  60 CALL ALPHA(2,100)
     P=.36353865
     Q=.41536445
     R=.38005646
     S=.38006737
     E=.71000000
     F=.72000000
     RAD=B*B-4.*A*C
     IF(RAD)10,20,20
  20 X=(-B+SQRT(RAD))/(2.*A)
     X1=(-B-SQRT(RAD))/(2.*A)
     GO TO 30
  10 X=999
     X1=999
C      OUTPUT9*S TO SHOW ROOT IMAGINARY
  30 CALL FORM(4,P,Q,R,S)
     CALL FORM(1,E)
     CALL WRITE(X)
     CALL FORM(4,P,Q,R,S)
     CALL FORM(1,F)
     CALL WRITE(X1)
     CALL BACK
     END
```

NOTE 1 -- THE ...PAUSE... IS NECESSARY AS THE FIRST INSTRUCTION IN THE FORTRAN PROGRAM. WHEN ASSEMBLED BY THE COMPUTER CENTER PERSONNEL, IT IS ALTERED TO A .... NO OPERATION .... CODE.

NOTE 2 -- STATEMENT NUMBER 50 IS USED TO ALLOW ANY PROBLEM IDENTIFICATION, DATE, ETC. TO BE ENTERED FROM TAPE AND LATER OUTPUTTED BY STATEMENT NUMBER 60. THE IDENTIFICATION NUMBER OF THIS STATEMENT IS ...100...

PART II

HARDWARE DESCRIPTION - THE FOLLOWING TELETYPE EQUIPMENT WAS OBTAINED FROM SURPLUS OUTLETS

MODEL 14 TRANSMITTER DISTRIBUTOR (TD)
MODEL 14 TYPING REPERFORATOR (PUNCH)
MODEL 15 PAGE PRINTER (MOD 15)
REC-30 LOCAL LOOP POWER SUPPLY

TO COMPLETE THE INTERFACE WITH THE 1710 COMPUTER, THE FOLLOWING HARDWARE WAS DESIGNED AND CONSTRUCTED BY THE ELECTRICAL ENGIN-

EERING DEPARTMENT

SET OF LATCHING RELAYS AND POWER SUPPLY FOR SAME
USER STATUS INDICATOR AND INTERRUPT GENERATOR
MODIFICATIONS OF THE TD

BRIEF DESCRIPTION OF OPERATING METHOD OF THE TELETYPE SYSTEM

THE TELETYPE MACHINES NORMALLY AVAILABLE FROM SURPLUS OUTLETS ARE 5-LEVEL MACHINES. THIS MEANS THAT FIVE PULSES ARE REQUIRED FOR EACH CHARACTER. SINCE THE MACHINE IS A 60 WORD PER MINUTE MACHINE, 163 MILLISECONDS ARE REQUIRED PER CHARACTER, THUS A MAXIMUM RATE OF ABOUT 6 CHARACTERS PER SECOND. THE COMPUTER IS REQUIRED TO GENERATE OR RECOGNIZE FIVE PIECES OF INFORMATION FOR EACH CHARACTER (ACTUALLY SEVEN SINCE THERE IS A START AND A STOP PULSE FOR SYNCHRONIZING EACH CHARACTER).

METHOD OF GETTING DATA INTO THE COMPUTER (SEE FIGURE 1)

THE 1710 AVAILABLE IN THE COMPUTER CENTER HAS HIGH SPEED CONTACT SENSE (HSCS) WHEREIN THE COMPUTER SCANS A BLOCK OF 20 CONTACTS AND PLACES A 7-DIGIT NUMBER IN CORE DEPENDING UPON THE CLOSED AND OPEN CONFIGURATION OF THE 20 CONTACTS.(*) THE REMOTE TELETYPE PUNCHES A PAPER TAPE IN THE COMPUTER CENTER AND THIS TAPE IS FED TO THE READER PART OF THE TD. THE READER CONTACTS ARE WIRED TO FIVE OF THE HSCS TERMINALS, THUS FOR EVERY TELETYPE CHARACTER SCANNED, A PARTICULAR NUMBER WILL BE STORED IN CORE OF THE COMPUTER. IT IS THEN NECESSARY TO HAVE A DICTIONARY (STORED IN CORE) TO TRANSLATE THESE NUMBERS INTO THE PROPER CHARACTER.

METHOD OF GETTING DATA FROM THE COMPUTER (SEE FIGURE 1)

FIVE CONTACT OPERATES (CO) IN THE COMPUTER ARE WIRED TO THE FIVE OUTPUT SEGMENTS OF THE TD. ANOTHER DICTIONARY (STORED IN CORE) IS REQUIRED TO DETERMINE THE PROPER CONTACT CONFIGURATION FOR EACH CHARACTER WHICH IS TO BE OUTPUTTED. THESE CONTACTS ARE OPERATED AND LATCHED. THE TD THEN TRANSMITS THE PROPER CODE TO THE REMOTE STATION. SINCE THE MINIMUM TIME PER CONTACT FOR THE CO FUNCTION IS 50 MILLISECONDS, THE MAXIMUM OUTPUT RATE IS ABOUT THREE CHARACTERS PER SECOND. THE TELETYPE MACHINE IS CAPABLE OF ABOUT TWICE THIS SPEED SO THE SYSTEM IS NOT VERY EFFICIENT. HOWEVER, IF THE OUTPUT MESSAGES ARE NOT TOO LONG, THE SYSTEM IS USABLE.

MISCELLANEOUS PROBLEMS AND SOLUTIONS

(1) TIMING

SEVERAL TIMING PROBLEMS EXIST IN THIS SYSTEM. PERHAPS THE MORE CRITICAL ARE CONCERNED WITH INPUT AND OUTPUT. IN THE READ ROUTINE, IT IS NECESSDRY FOR THE COMPUTER TO CAUSE THE HSCS TERMINALS TO BE SET UP BY THE TD. WHILE THIS IS BEING DONE(163 MS) THE COMPUTER MUST WAIT. ONCE THE TERMINALS ARE SET UP, THE COMPUTER MUST SCAN THE TERMINALS, TRANSLATE INTO A RECOGNIZABLE CHARACTER, INITIATE THE BEGINNING OF

ANOTHER SET-UP AND AGAIN WAIT. THIS IS ACCOMPLISHED IN THE FOLLOWING MANNER. A CO IS USED TO TELL THE TD TO ADVANCE TO THE NEXT CHARACTER. WHEN THE SET-UP IS FINISHED, THE TD OPERATES A PROCESS BRANCH INDICATOR (PBI) WHICH TELLS THE COMPUTER TO NOW SCAN THE HSCS TERMINALS. ONCE THE CHARACTER IS IN CORE AND TRANSLATED, THE CO IS AGAIN OPERATED FOR THE NEXT CHARACTER.(*) USING THIS PROCEDURE, THE SYSTEM WILL OPERATE AT NEARLY MAXIMUM TELETYPE SPEED.

IN THE WRITE ROUTINE, THE COMPUTER MUST OPERATE CONTACT OPERATES FOR EACH CHARACTER, WHICH ARE THEN EXTERNALLY LATCHED. WHEN ALL SEGMENTS FOR THE CHARACTER ARE SET UP, ONE MORE CO IS OPERATED TO TELL THE TD TO SEND THIS CHARACTER. WHILE THE CHARACTER IS BEING SENT (163 MS) THE COMPUTER MUST WAIT. AT THE END OF THE CHARACTER, THE SAME PBI IS USED TO TELL THE COMPUTER TO ASSEMBLE THE NEXT CHARACTER. BECAUSE OF THE LONG CO TIME, THE MAXIMUM RATE IS ABOUT 3 CHARACTERS PER SECOND.

(2) LATCHES

SINCE THE CONTACT OPERATES ON THE 1710 AVAILABLE WERE NON-LATCHING TYPES, EXTERNAL RELAY LATCHES WERE REQUIRED. SINCE THE SPEED OF OPERATION AND RELIABILITY OF THESE RELAYS HAD TO BE HIGH, RATHER EXPENSIVE MINIATURE SEALED CAN RELAYS WERE USED.

(3) TELETYPE TD MODIFICATIONS

THE TD WAS MODIFIED IN ORDER TO SEPARATE THE READER FUNCTIONS FROM THE OUTPUT FUNCTIONS. THIS REQUIRED THE CONNECTION OF A SEVEN WIRE LEAD TO THE OUTPUT SEGMENTS, SLIGHT MECHANICAL CHANGES IN THE RELEASE MECHANISM AND THE ADDITION OF A MICRO-SWITCH TO SENSE THE END OF A CHARACTER TRANSMISSION.

(4) USER STATUS INDICATOR AND INTERRUPT GENERATOR

IT WAS NECESSARY TO CONSTRUCT A SERIES OF SIGNAL LIGHTS TO LET THE REMOTE STATIONS KNOW WHEN THE SYSTEM WAS AVAILABLE AND TO KNOW WHEN THE SYSTEM WAS BUSY. INCORPORATED IN THIS FUNCTION IS THE INTERRUPT BUTTON. THIS IS SIMPLY A RELAY, REMOTE OPERATED, WHICH ACTIVATES THE PROPER INTERRUPT TERMINALS IN THE COMPUTER.

(5) TELETYPE MACHINE

LITTLE MODIFICATION IS NECESSARY. THE KEYBOARD CONTACTS AND THE RECEIVING SELECTOR MAGNETS ARE CONNECTED IN SERIES. ALL REMOTE STATIONS AS WELL AS A MONITOR PRINTER IN THE COMPUTER CENTER AND THE PAPER TAPE PUNCH ARE CONNECTED THEN IN SERIES WITH A LOCAL LOOP POWER SUPPLY(60 MA. OF CURRENT). WHICH REMOTE STATION OPERATES DEPENDS UPON WHETHER THE MOTOR ON THE REMOTE IS RUNNING OR NOT. THE MOTOR ON-OFF IS CONTROLLED BY THE USER STATUS INDICATOR UNIT.

DESCRIPTION OF MEMORY DATA REGISTER METHOD OF OUTPUT

(6) SPEED OF OUTPUT
AS PREVIOUSLY STATED, THE MAXIMUM OUTPUT SPEED IS ABOUT 3 CHARACTERS PER SECOND. THE TELETYPE IS CAPABLE OF OPERATING AT TWICE THIS SPEED. THE SPEED PROBLEM THUS BECOMES ONE OF SPEEDING UP THE CONTACT OPERATE FUNCTION. SINCE THIS IS NOT VERY FEASIBLE, ANOTHER APPROACH HAS BEEN UNDERTAKEN. OBSERVATION OF THE 1710 CONSOLE PANEL SHOWS THAT ALL OUTPUT DATA FLOWS THROUGH THE MDR (MEMORY DATA REGISTER). THIS DATA CAN BE OUTPUTTED BY MAKING A HIGH IMPEDANCE CONNECTION TO THE PANEL LAMPS. THE PROCESS IS AS FOLLOWS .... (SEE FIGURE 2)
USE A TRANSMIT RECORD CODE ... 31 XXXXX XXXXX
TO MOVE DATA FROM LOCATION XXXXX TO ITSELF. IN THE PROCESS THIS DATA GOES THROUGH THE MDR. AMPLIFY THE INFORMATION PULSES FROM THE PANEL LAMPS OF MDR AND USE TO SET ELECTRONIC LATCHES (RELAYS ARE TOO SLOW). THESE LATCHES, AS BEFORE, SET UP THE CONTACT SEGMENTS OF THE TD. AS SOON AS A RECORD MARK APPEARS IN MDR, INITIATE THE SPIN OF THE TD. LET THE COMPUTER WAIT FOR A PBI AS BEFORE. WHEN THE PBI APPEARS, SEARCH FOR THE NEXT RECORD, RESET ALL LATCHES, FLUSH THIS NEW RECORD THROUGH MDR, ETC. USING THIS PROCESS, MAXIMUM TELETYPE SPEED CAN BE REACHED. SINCE IT TAKES THE COMPUTER ONLY A FEW HUNDRED MICRO SECONDS TO SET UP ALL THE LATCHES, THE SPIN ARM ON THE TD NEVER STOPS. WE HAVE TESTED THIS METHOD AND FOUND IT WORKS, AND ARE NOW IN THE PROCESS OF BUILDING THE ELECTRONIC AMPLIFIERS, GATES, LATCHES, ETC.

* REF. IBM 1710 CONTROL SYSTEM MANUAL
FILE NO. 1710-01
FORM NO. A26-5709-0

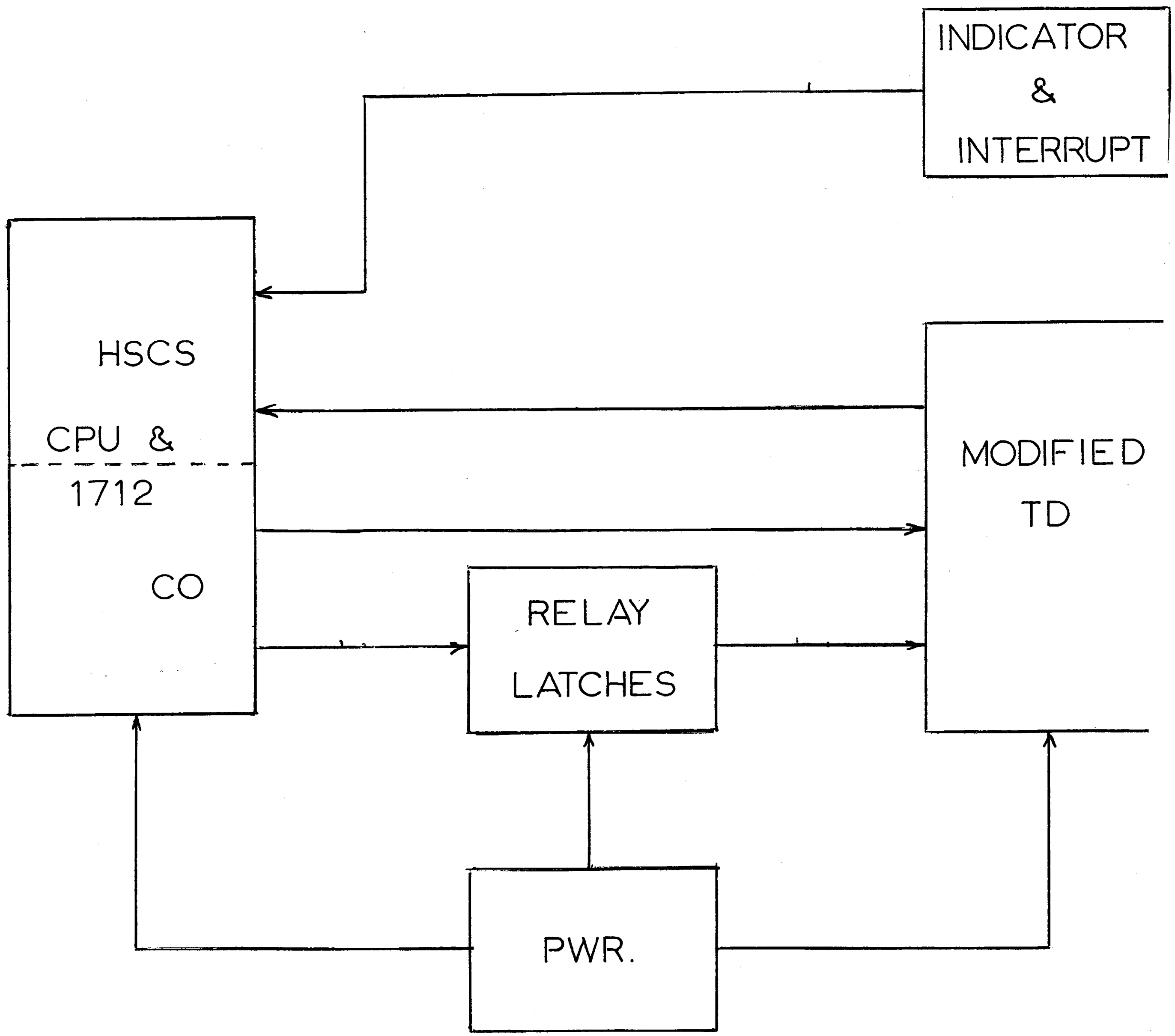

INDICATOR
&
INTERRUPT
HSCS
CPU &
1712
CO
MODIFIED
TD
RELAY
LATCHES
PWR.

FIG. 1.

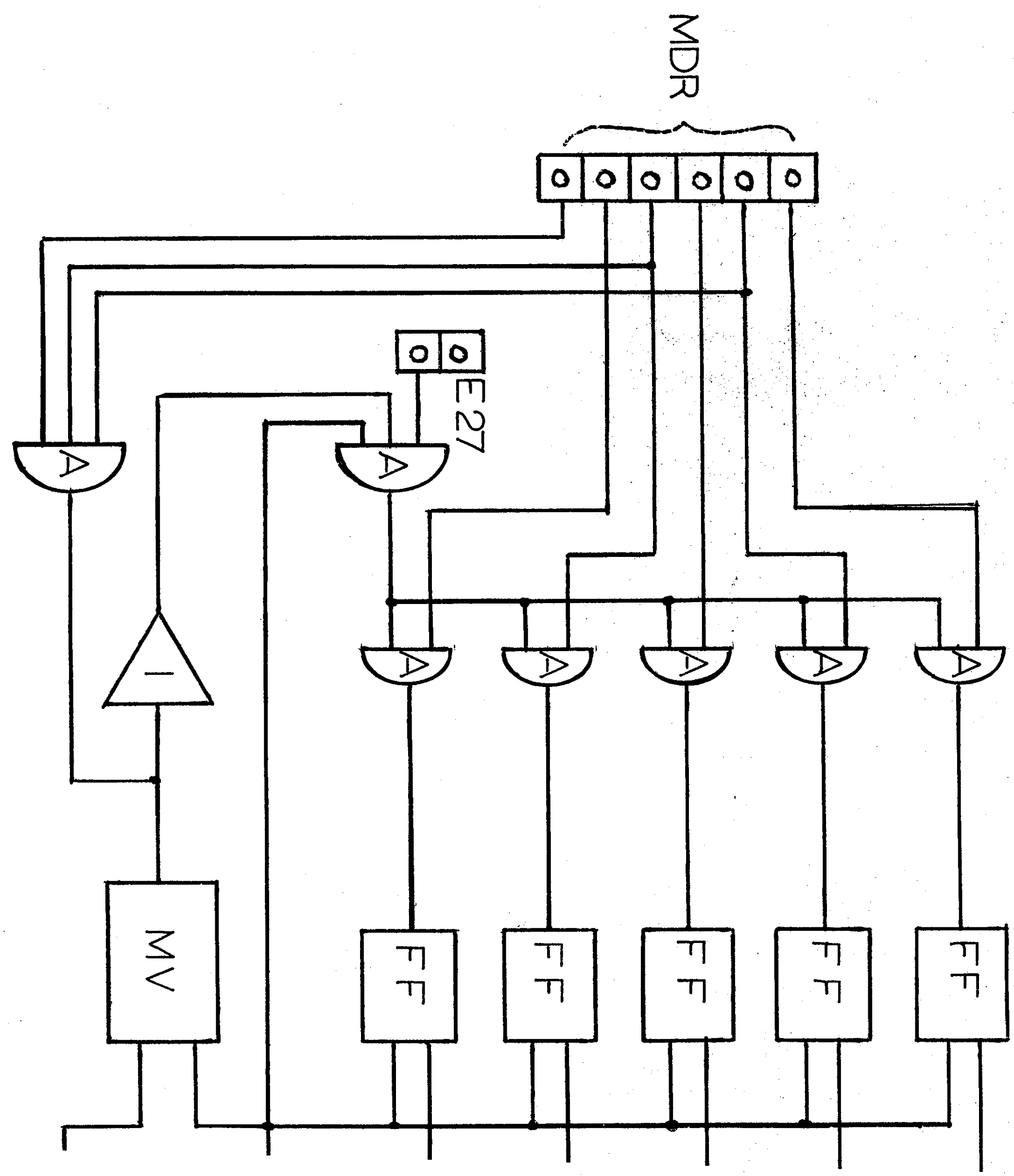


FIG. 2

1620 APPLICATIONS

GENERAL CARD TO PRINTER PROGRAM

Mrs. Janet Allen
Pioneer Hi-Bred Corn Company
1206 Mulberry Street
Des Moines, Iowa

288-3691

Monday, 3:30-5:00 P.M.
Session M-4.8

8 pages
3 exhibits

INDEX

GENERAL CARD TO PRINTER PROGRAM

I. Introduction

II. Control Cards

III. Printer Tape

IV. Operating Instructions

V. Error Messages

Exhibits:

A - General Card to Printer Control Cards
B - Test 1 - Control Cards
Test 2 - Control Cards
C - Cards to load program on Disk

GENERAL CARD TO PRINTER PROGRAM

I. INTRODUCTION

The General Card to Printer Program is designed to handle the transition from a 1620 with card reader/punch only to a 1620-1443 System. Through control cards to the program, the user specifies the input and output fields. If the input cards are of different formats, the user may specify a card code field with which to identify each card. Headings to be printed at the top of the page may be specified. In addition, the user may specify any input field(s) to be printed on one line at the bottom or top, or both, of each page. Other options include merging two cards into one output line, and ability to restore page or space a maximum of three lines on a specified field, a non-blank, or a change in a field.

Switches control single or double spacing and allow a straight card to printer, without formats.

This program requires a 20K Model 1 with 1443 printer, 144 printer positions, indirect addressing, and numeric strip and fill instructions.

II. CONTROL CARDS

See Exhibit A format. All numbers are to be right-justified in the fields, unless otherwise specified. See Exhibit B for examples of the control cards. All specified card codes must be punched the same number of columns as specified in the control card CDCODE. If more columns are punched in the card code field (columns 8 - 17) the data cards with that card code will not be recognized.

A. CARD CODE

This specifies the field with which to identify input cards. Columns 20-21 contain the first column of the field, and columns 22-23 the number of columns (maximum 10). A maximum of ten different card codes may be specified.

If this card is omitted, the program assumes all cards to be the same format. All card code fields in the other control cards must then be blank.

B. RESTORE PAGE AND SPACE

The control cards for these are identical format, with one additional field in the SPACE card.

To compute the maximum number of each control card, the following must be noted: each table for RESTOR and SPACE allows 12 fields. Each control card, however, for a different card code requires an extra field. Therefore, 6 RESTOR cards, all referring to different card codes, would fill the table.

The card code field (columns 8 - 17) specifies the card code to which the control card refers, or is blank if all input cards are the same format. There may be more than one for any card code, but all RESTOR or SPACE cards for that card code must be together.

Columns 20-21, and 22-23 indicate the beginning and size of the field. A maximum of ten columns may be specified. If column 25 contains a "c", any change in the field will cause lines to be spaced (for SPACE) or the page to be restored (for RESTOR).

An "f" in column 25 requires a field to be specified, beginning in column 31. This field must contain any flags that the data card has. When this field is found, the lines are spaced or the page restored. The " @ " in column 31 indicates that any non-blank in the specified field will cause spacing or page restoring.

A blank in column 26 causes the line to be printed before lines are spaced on the page restored. A non-blank causes the spacing or page restoring to occur before printing.

Column 27, on the SPACE card, specifies the number of lines to be spaced. If it is blank, one line will be spaced. A maximum of 3 lines may be specified.

C. MERGE

There may be a maximum of 10 of these control cards. This causes the two input cards whose card codes are specified (columns 8 - 17 and 25 - 34) to be printed on the same line. The program expects the cards to be in the same order as on the control card. See Section V for possible error conditions.

D. HEADINGS

A maximum of eight heading lines may be specified. The user need not specify all - a line will be skipped for each line not specified until either the last specified header line or the channel 9 punch on the printer tape is reached. If the specified header lines go beyond the channel 9 punch, one line will be skipped before the first line of data.

The header field, beginning in column 31, may be continued on the next HEADER card. Columns 22 - 24 on the first card contain the total number of columns of the header field. If this exceeds 50, the next HEADER card is assumed to contain the continuation, beginning in column 31. Any header field may contain a record mark to terminate printing of that line. The character count in columns 22 - 24, however, still controls the number of columns read in.

E. TOP

This control card allows the user to specify information to be picked up from an input card or cards and printed on the first line of a page. This appears before the headings. The I/O fields on these control cards are the same as those of the FORMAT card (Section II H.). There may be a maximum of three different card codes, and a maximum total of seven I/O fields (see Section II G.) All information specified by all TOP control cards will appear in one line at the top of the page. All TOP cards specifying the same card code must be together. In order for the top line on the first page of printed data to contain the proper information, this information must be in the first input card. The program initially skips to the bottom of the page. It then reads a card, stores any of its data to be printed, restores the page, prints the top line and headings, then continues printing and reading cards. Whatever data for the top line not contained in the first card will thus not be printed on the first page.

F. BOTTOM

This control card is the same as TOP except that the specified data will appear in the last line of the page.

G. FORMAT

This card defines the input to output (I/O) format of the card whose card code is specified in columns 8 - 17. If there are no card codes, this field must be a blank. All FORMAT cards for the same card code must be together. If it is desired to shorten the output line, columns 25 - 27 may contain the output position for a record mark. If there is more than one FORMAT card for the same card code, the record mark position must be specified on the first card, or it will be ignored.

Beginning in column 31, the I/O fields, described below, specify the format. These fields must be separated by commas. There may be any number of each FORMAT card, but each field must be complete - it may not be continued on the next card.

The table in the program contains 106 I/O fields. To compute the number that may be specified, the following must be noted:

a). Each card code requires one more I/O field. Therefore, if there are two different card codes specified by FORMAT cards, 104 I/O fields are left.

Each I/O field must be the following format and in this order:

$$\underbrace{\text{I - - N - - O - -}}_{\text{must be present}}\ \underbrace{\text{A S D -}}_{\text{optional}}$$

These must be present:

I - - defines the first column of data on the input card for this field. May be 1 or 2 digits.

N - - is the number of columns of this data field. May be 1 or 2 digits.

O - - is the last (rightmost) print position. May be 1, 2 or 3 digits.

If the number is negative, the minus sign will appear to the right of this position.

These are optional specifications. If none is present, the program assumes the field to be numeric with no decimal and will remove leading zeroes and check for negative. The options S and D are meaningful only if A is not present.

A. Indicates this field is alphanumeric. The field will then be reproduced in the output line just as it appears on the card.

S. Causes the output field to be blank if the input field is blank or zero.

D. Indicates that a decimal point is to be printed. There must be one digit, 0 - 9. Zero causes a decimal point to be printed to the right of the number. This specification must be last in the I/O field.

III. PRINTER TAPE

This is intended to be the same tape to be used with the Monitor, with one additional punch in channel 9.

A. Channel 12 - end of page

B. Channel 1 - beginning of page. This will be the first header line.

C. Channel 9 - first line of data. This punch controls the number of header lines. If a header line is specified past this punch (farther down the page), the specified line will be printed and one line skipped before the data is printed.

IV. OPERATING INSTRUCTIONS

A. SWITCHES

1. Switch 1 - ON - double space
OFF - single space

2. Switch 2 - ON - straight print (no format)
   OFF - FORMAT control cards control printout

3. Switches 3 and 4 not used.

B. DISK

The program is in two parts, both stored on disk. Both use the first 120 sectors of the work cylinders.

To load, see Exhibit C. PRT2 must be loaded with DIM number 400, unless the first part PRINT is changed. PRINT links with PRT2 through DIM number 400.

C. CARD DECK

1. Switch 2 OFF

   The first cards must be the control cards, in any order, followed by a 9's card (9's in columns 1 - 6)

   The card deck is:

   ≠ ≠ JOB 5
   ≠ ≠ XEQ PRINT
   control cards
   9's card
   data cards
   .
   .
   .
   End of Job Card (≠ ≠ ≠ ≠)

2. Switch 2 on - the card deck is the JOB and XEQ cards and the set of input cards followed by the End of Job card.

3. It may be desired to have a straight print, but also have the option of any of the other control cards, except FORMAT, and BOTTOM.

   This may be done in the following way:

   a) Turn off switch 2
   b) Read control cards and 9's card
   c) Turn on switch 2
   d) Read data cards

   Any FORMAT cards read in will be ignored. Note that, if switch 2 is OFF, FORMAT cards must have been read in to get any output.

D. Check printer tape (see Section III) and load cards. When program reads END OF JOB card, it will return to Monitor.

V. ERROR MESSAGES

Error messages appear on the typewriter.

A. Control Card Errors

Unless otherwise specified, the control card is typed after the error message and is ignored.

1. The following are typed when the maximum of the specified control cards have already been read.

   a. MAX RESTORE CARDS
   b. MAX SPACE CARDS

2. MAX CC

   Maximum number of different card codes have been read.

3. I/O FORMTS TABLE FULL

   Maximum number of I/O formats has been reached for TOP, BOTTOM, or FORMAT. The line typed after this message is only those formats not processed.

4. HEADER LINE EXCEEDS MAX

   A header line greater than 8 has been specified.

5. ERROR IN I/O FORMAT

   Although the entire card is typed, only the I/O field in error is ignored.

6. ILLEGAL CONTROL CARD

   Control card code is not recognized by the program.

7. COLUMN SIZE EXCEEDS MAX

   Columns specified on CDCODE, RESTOR, SPACE, or HEADER exceed the maximum, either number of columns or input column number; output column number specified in a HEADER card exceeds 144; or a header line as specified would exceed 144.

8. MAX SPACING IS 3.

   The number of lines specified in a SPACE card is more than 3. Three lines will be spaced.

B. ERRORS IN PROCESSING DATA CARDS

1. cc - - - - - - - - - - - - - - - NOT MERGED

This indicates that the second card specified on the MERGE is missing. Two cards with the first specified card code have been read. The first is typed and ignored.

EXHIBIT A

GENERAL CARD TO PRINTER CONTROL CARDS

CONTROL CARDS	1 - 6 CODE	8 - 17 CARD CODE	20 - 21 FIRST INPUT COL. NUMBER	22 - 23 NUMBER OF COLUMNS	25	26	27	31
1. CARD CODE OF INPUT	CDCODE	--	XX	XX (max - 10)	--	--	--	--
2. RESTORE PAGE	RESTOR	X....X	XX	XX (Max = 10)	"C" or "F"	blank print first	--	Field - (a) (non-blank)
3. SPACE	SPACEb (b=blank)	X....X	XX	XX (max = 10)	"C" of "F"	blank-print first	# spaces	Field - (a) (non-blank)
4. MERGE	MERGEb	X....X	--	--	Col. 25 - 34 = Second Card Code			
5. HEADINGS	HEADER	--	Line # = XX	XXX (Col. 22-24)	Output Col. # (col. 25 - 27)			Field
6. TOP OR BOTTOM	TOPbbb BOTTOM	X....X	--	--	Output col. # for record mark			Field
7. I/O FORMATS	FORMAT	X....X	--	--				Field

EXHIBIT B

```
CDCODE          0102
MERGE       XX              6P
RESTOR      XX  1803 C1
RESTOR      XX  0303 F     510
SPACE       XX  0303 F22   505
SPACE       XX  0704 F     '
BOTTOM      6P             I3N303
BOTTOM      XX             I3N3015A
TOP         XX             I3N305A,I15N7010A
FORMAT      XX             I3N303A, I7N8011A, I15N9021A,              I24N9032A
FORMAT      6P             I30N2072,I32N3075,          I35N2079,I37N3082
FORMAT      6P             I63N20118
FORMAT      6P             I3N3035,I6N3039D1,I9N3043,I12N3047,         I15N2051
FORMAT      6P             I17N3054,I21N1058,I22N3061D1,I25N2065,    I40N2086S
FORMAT      6P                      I42N3089,I45N3093D2,             I48N3097D2
FORMAT      6P             I51N30101D2,I54N30105D2,I57N30109D2,I60N30113D2
TOP         XX             I3N305A,I15N7010A


CDCODE          0102
HEADER          01050001   CODE   VARIETY   MALE PED   FEM PED
HEADER          03092001   AVG HOS AGE AHH CM AHD CM CHD CM GM CM  LM CM DP
HEADER                     L CM BDY BW LEL  EW  HU  BS  SG LOC
FORMAT      XX             I3N303A, I7N8011A, I15N9021A,              I24N9032A
FORMAT      6P             I3N303, I6N307D1, I9N3012, I12N3016, I15N2019
FORMAT      6P             I17N3023, I20N2026, I22N3030D1, I25N2034S
FORMAT      6P             I30N2037S, I32N3041,   I35N2044,   I37N3048
FORMAT      6P             I40N2051,    I45N3059D2,    I48N3064D0
FORMAT      6P             I51N3069D2, I54N3074D2, I57N3079D2,   I60N3084D2
FORMAT      6P             I63N2088
FORMAT      6P             I42N3055SD2
```

EXHIBIT C

(CARDS TO LOAD PROGRAM ON DISK)

*DLOADPRINT		0240202402	CM
*DLOADPRT2	0400	0240202402	CM

The General Format Conversion Program

by

Robert B. Balder

Presented orally at the joint Eastern-Western COMMON meeting November 28-30, 1966, New Orleans, Louisiana.

The General Format Conversion Program

The incompatibility of data between card oriented programming systems has long been a bottleneck in incorporating external data into existing systems. This is in a large part caused by the necessity to reformat the card input into a standard format acceptable to the operational system. Up until now, this usually required a separate program to do the required formating. To all but the most inexperienced trainee this type of program is as best an inefficient use of programming resources, a tedious coding task to the programmer, and an uninspiring chore to the creative thinker. The task becomes more overwhelming when large numbers of this type of program have to be written.

System Description

In an attempt to minimize the programming time and computer debugging time required by a large number of reformating programs, the "General Format Conversion" (GFC) system of programs has been developed. These programs are written in SPS II-D for a 40K IBM 1620 (mod I) with two disk drives under the control of Monitor with the automatic divide and indirect address special features.

The system works in the following manner: control cards are completed and keypunched. These control cards are the elements of a "conversion library" which is defined as all control records necessary to completely transform the cards of one format into another. The conversion library is stored on a disk pack that has been pre-initialized for this system. As many as ninty-nine conversion libraries may be stored on any one disk pack.

Once a conversion library is stored, it can be called by another program of the system which reads in the cards in the original format, makes the transformations defined in the conversion library, and punches the reformated cards. The processing time varies with the number of conversions but is not much slower then the Input/Output speeds.

Statement of the Problem

In order to understand the operation of the system, it is necessary to understand the various control cards and their function. In order to accomplish this objective, the problems encountered in doing any reformating of cards will be stated and the general way by which GFC control cards solve each particular problem will be explained. To begin with, one definition is necessary. In defining a field on a card, there are at least two elements necessary: 1) the number of characters, and 2) the card column (cc) of the "rightmost" or "low order" character. This is the definition of field used throughout the system.

Input Selection

All cards containing information concerning one particular item under observation must be read in at one time. This is probably more than one and possibly can be a variable number of cards. Therefore, there must be some key used to recognize all cards referring to a particular item. This may be a sample number, a date, a location code, a person's name, and so forth, or a combination of fields. In the GFC this is referred to as the "family identification" and can consist of one, two, or three fields and cannot exceed a total of twenty-five characters. The defining of the family identification is done by the FAMILY ID control card which completely defines each field by giving the number of characters and the card column of the low order position of the fields as they must appear on all cards in the family.

F	A	M	I	L	Y		I	D	,	F	L	D	1		C	H	A	R	S	-	X	X	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

C	C	-	X	X	-	,	F	L	D	2		C	H	A	R	S	-	X	X	-	C	C	-
25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

X	X	-	,	F	L	D	3		C	H	A	R	S	-	X	X	-	C	C	-	X	X	-
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72

-					1	0	0
73	74	75	76	77	78	79	80

If a family consists of more than one card, it is probable that these cards are themselves not of the same format although they all contain data referencing the same event and rightfully belong to the family. Therefore, there must be some manner of distinguishing the different cards within the family. This is called the "card identification". In the GFC there is provision for one field of up to five characters in length to serve this purpose. This is defined in the system by the "CARD ID" control card which contains the length of the field (the card column of the rightmost character), and the number of different card types possible at input (maximum of 25).

C	A	R	D		I	D	,	C	H	A	R	S	-	X	-	C	C	-	X	X	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

N	U	M	B	E	R		O	F		P	O	S	S	I	B	L	E	
23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41

C	O	D	E	S	-	X	X	-	-
42	43	44	45	46	47	48	49	50	51

2	0	0
78	79	80

Following the CARD ID card comes one "CODE" control card for each of the various card types. This card contains the card type code (up to five characters), and the minimum and maximum family requirements as to the number of the card type required within each family (if any). That is, if there must be at least one of a particular type of card in each family, the "MINIMUM" is coded "01". If a particular card type is not necessary, then the "MINIMUM" is coded "00". Similarly, if there should not be more than five of a particular type within a family, the "MAXIMUM" is coded "05". If there is a card where there must be one and only one of a particular card type, then both the "MINIMUM" and "MAXIMUM" are coded 01. These requirements are checked during the execution of the conversion library, and if they are not met appropriate messages are typed on the console and the card family is ignored.

C	O	D	E	-	X	X	X	X	X	-	M	I	N	I	M	U	M	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

N	U	M	B	E	R		R	E	Q	U	I	R	E	D	-	X	X	-
20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38

M	A	X	I	M	U	M		N	U	M	B	E	R		A	L	L	O	W	E	D	-
39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61

X	X	-	-													2	X	X
62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80

If there is to be only one card per family and this card will always be the same type, then the FAMILY ID and CARD ID cards need not be present.

Blank Fields

Another problem encountered in reformating is what to do with blank fields. In a particular system it may be necessary for all columns of the cards to be coded. In these cases, there is usually some special character substituted for "not applicable" or "unknown". We allow for this in the GFC by using a "padding" character. This can be any single character, either numeric, alphameric, or special that will be substituted in each column of the output field if the input field is blank. If a padding character is not specified by a PADDING CHARACTER control card, then a blank character is used.

P	A	D	D	I	N	G		C	H	A	R	A	C	T	E	R	-	X	X	-	...	4	0	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	...	78	79	80

Types of Conversions

All fields are defined by the two parameters, length, and card column of the low order character. By including the card identification code (i.e., card type), the field is completely defined within a family of cards. If there is only one type of card per family, this third parameter is not necessary. Since all fields are handled in alphameric mode, one character "fields" are permissible.

The basic for the general reformating of cards is that all conversions fall clearly into three broad categories and a specific control card is provided for each category.[1] The simplest of these categories is the "field to field" conversion. The field as it appears in the input is transferred directly to the output. To this conversion, the code "1" has been assigned. The number of characters in the field is coded, the card type within the family from which the data is to come, if any, is coded, the card column of the input is coded, and the card column of the output is also coded. In this case, we do not need the

[1] Appendix 1 contains the coding instructions for all control cards.

length of field in the output because it is the same as the input since we are transferring the entire field as it appears. As in all conversions, we do not need the card type of the output since only one card is constructed and punched at a time.

The second basic category is used if the reporting codes between the two systems differ. An example of this is if one system uses the reporting code "M" meaning "male" and "F" meaning "female", and the system for which the output is intended uses "1" and "2" for the same purpose. This type of conversion is accomplished by a table look up and is called the "field to table to field" conversion. This is given the conversion code "2". As a description of the input field, we have the number of characters, the card type, and the card column of the rightmost character. To describe the output, there is the card column of the low order character and the number of characters, which, in this case, is required since it may differ from that of the input. There is one additional item required by this conversion category, and that is the number of "entries" in the table (a maximum of 999).

Immediately following a conversion of this type comes the table, one entry per card. These cards contain first the code as it might appear in the input immediately followed by a comma and the corresponding code as it appears in the output. In the execution of this conversion, if a code is not found in the table, a message is typed on the console and the operator has the option of either padding the output field with the padding character previously defined or correcting the card and reentering it through the card reader.

The third and sort of a catch all type of conversion, is the "field to subroutine to field". In this case, information cannot be transferred directly or through a table look up, but instead, must be processed through some mathematical function or data manipulation subroutine. This type of conversion is given the code "3". In the control card, both input and output fields are defined by the number of characters and the card column of the rightmost position. Also, as in the other types of conversions, the input card type is also specified if necessary. There is one other bit of information on this card, the entry point which will be discussed later.

Repetition of Conversion Control Record

A conversion library is defined above as "all control records necessary to completely transform the cards of one system to another". Notice that "cards" is plural. The variability of the input has been shown. The GFC can also produce different card types at one pass of the data and also can produce a variable number of each card type.

Although all cards of a family are read and stored prior to the start of the actual conversion, only one output card is produced at a time. However, any number of output cards can be produced per family. All conversion control cards necessary to produce one output card are together called an "entry" in the conversion library and must contain as the last record the "PUNCH" control card. Besides giving the program the command to proceed with the punch routine, this record contains two other bits of information:
1) whether or not the card type just punched is to be repeated, and 2) whether or not the card type just punched is the last of the different card types to be produced in each family.

Card Selection Within Family

In repeating an entry in a conversion library there must be some way of obtaining data from different cards within the family on each pass through the entry or else the same output card would be produced over and over again. There are two ways one might want to consider a particular field for the output: 1) taken from the same card of the family on each pass through the entry in the conversion library, or 2) taken from a different card although off the same card type on each pass through the entry in the conversion library. For this second case, an index counting the number of passes through the entry is used. Each field is considered individually since it may be desirable to produce certain fields on output cards containing some information that is to be the same on all cases (i.e., family identification, date, and so forth), and other fields containing differing information.

Another problem solved at the same time as the one above is that of to "pad" or not to "pad". If a blank field is encountered somewhere in the input family, it may be of minor importance, therefore, little is lost if the "padding" character defined by the "PADDING CHARACTER" control card is substituted in the output field.[2] However, it may be of such major importance that if it does not appear, it is not desirable to punch the card at all. In this case, the program proceeds either to the next conversion entry or begins to read the next card family. It is the last feature that is the mechanism for getting out of the loop. If there are N cards of input card type X and N cards of output card type Y are to be produced, where each pass through the entry in the conversion library producing card type Y is to get data from a different card type X, on the N + 1 pass through the entry card type X will not be found. Therefore, the field is not found and the program does not punch the N + 1 type Y card and proceeds to the next entry. If there is no next entry, the next input family is read in; all indices are initialized and the process begins anew.[3]

Other Methods of Placing Data in the Output

Besides these three basic types of conversions, there are two other ways of placing information into the output card that cannot really be called "conversions" in the strictest sense of the word. One of these solves the problem of placing a constant field in the output that does not appear anywhere in the input. This is done by the "immediate to field" conversion control card in the following manner: the output field is defined in the same way as in the other conversions. No input field description is necessary. After the output field description comes the exact constant as it is to appear in the output. There is a limit of 48 characters, which can be alphameric, numeric, or special, per control card of this type.

[2] If no "PADDING CHARACTER" control card is used, a "blank" is substituted.

[3] Appendix 2 contains the possible conversion control codes.

Also, it is possible to store the status of particular counters directly into the output. There are three counters that can be used. Counter one contains the number of cards in the family, counter two the number of cards of a particular card type which is specified in the control card, and counter three is the value of the "index" which is being used for card selection. This conversion is coded "02".

The Usage of Subroutines

It is the ability of the user to easily incorporate subroutines into the GFC system that has made this system the versatile tool it has proven to be.

Basically, subroutines are written in SPS. They are entered through an unconditional branch instruction rather than a branch and transmit or a branch and transmit immediate. Therefore, the address registers are free to be used by the subroutine themself. Data enters the subroutine in alphameric form by the use of an indirect address established by the program. The results of the subroutine are stored immediately prior to the first executable instruction in alphameric form for a flag over the zone position of the leftmost character. Exit from the subroutine is made by an unconditional branch back to a specific instruction in the main program.

It is possible to have any number of subroutines. Each subroutine should be written and debugged independently of the conversion program and other subroutines.

If subroutines are to be used, an additional input control record is necessary and is inserted after the "CARD IDENTIFICATION CODE" control cards and prior to the "PADDING CHARACTER" control card. There are two elements of information on this card: 1) the number of entry points into the subroutine package (an entry point is the address of the first executable instruction in a subroutine, therefore, the number of entry points is the number of individual subroutines in a subroutine package), and 2)

The Disk Identification Map (DIM) Entry number of where the subroutines are to be loaded on the Monitor disk.

S	U	B	R	O	U	T	I	N	E		E	N	T	R	Y		P	O	I	N	T	S
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

-	X	X	-	D	I	M		E	N	T	R	Y		N	U	M	B	E	R	-
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

X	X	X	X	-
45	46	47	48	49

3	0	0
78	79	80

When all subroutines are debugged, they are stacked together and assembled as a package, and then stored in object form on the Monitor disk using the DIM entry number specified by the "SUBROUTINE" control card. During the final assembly of the subroutine package, it is best to obtain a listing of the package. This is necessary to ascertain the addresses of the entry points which are required for execution of the library. In the execution of the program that actually makes the conversion of format, there is only one type of control card necessary, and then only if subroutines are used. This is the "SUBROUTINE ENTRY POINT" card. There is one of these cards for every individual subroutine in the package. There are two fields on this card, the entry point number, which are sequential beginning with one, and the address of the first executable instruction of the subroutine, which is determined from the program listing.

Conclusion

This system has been used by the U.S. Public Health Service, Division of Radiological Health, for only a short period of time. However, it has proven itself to be a versatile and useful tool in cutting programming resource requirements necessary for incorporating external data into existing programming systems. The flaw uncovered so far is that the fields used to define the family identification and card identification must be uniform in all cards. But if these fields are not in a consistant location in all cards, then each card type has to be converted separately if possible. This is not a major problem.

A person with a fair knowledge of the system can usually code and debug an average conversion library in three to six hours, depending upon the number of transformations and the proficiency of the person doing the coding. Compared to writing and debugging each conversion program separately, this system saves both time and programming costs.

Appendix 1

GFC Control Card Coding Instructions

Family Identification Card

If there is a family identification which has only one field, code A below, if two fields code A and B below, if three fields code A, B, and C below. In any event, code 100 in cc 78-80.

A.

F	A	M	I	L	Y		I	D	,	F	L	D	1		C	H	A	R	S	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

		-	C	C	-			-
22	23	24	25	26	27	28	29	30

Code the number of characters in the first family identification field in cc 22-23. Code the card column of the units position of the first family identification field in cc 28-29.

B.

,	F	L	D	2		C	H	A	R	S	-			-	C	C	-			-
31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51

Code the number of characters in the second family identification field in cc 43-44. Code the card columns of the units position of the second family identification field in cc 49-50.

C.

,	F	L	D	3		C	H	A	R	S	-			-	C	C	-			-	-
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73

				1	0	0
74	75	76	77	78	79	80

Code the number of characters in the third family identification field in cc 64-65. Code the card column of the units position of the third family identification field in cc 70-71.

Card Identification Card

If the cards within a family carry an identification, then this card should be coded. Only one field is allowed for card identification and it may not be more than 5 characters in length. If this card is coded, then code cards must also be included (see code card). The number of codes may not exceed 25. Format:

C	A	R	D		I	D	,	C	H	A	R	S	-		-	C	C	-			-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

N	U	M	B	E	R		O	F		P	O	S	S	I	B	L	E	
23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41

C	O	D	E	S	-			-	-
42	43	44	45	46	47	48	49	50	51

2	0	0
78	79	80

Code the number of characters in the card identification (a number from 1 to 5) in cc 15. Code the card column of the units position of the card identification in cc 20-21. Code the number of possible codes for the card identification in cc 48-49. Code 200 in cc 78-80.

Code Card

If there is a card identification card, then there must be code cards equal in number to the number coded in cc 48-49 of the card identification card. Format:

C	O	D	E	-						-
1	2	3	4	5	6	7	8	9	10	11

M	I	N	I	M	U	M		N	U	M	B	E	R		R	E	Q	U	I	R	E	D
12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34

-			-	M	A	X	I	M	U	M		N	U	M	B	E	R	
35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53

A	L	L	O	W	E	D	-			-	-
54	55	56	57	58	59	60	61	62	63	64	65

2		
78	79	80

Code the card type in cc 6-10, right justified. Code the minimum number of cards required of this card type for each family in cc 36-37. Code the maximum number of cards of this type allowed for each family in cc 62-63. Code a sequence number in cc 79 and 80, beginning with 01 for the first card type code.

Subroutine Card

If subroutines are to be used in the conversion process, they should all be written as one program; stored on the disk under some dim entry number. The subroutines card should be used only if there are subroutines. There will be a number of entry points into the subroutine set. The actual entry point address will be given when the conversion library is effected (P17.03). Format:

S	U	B	R	O	U	T	I	N	E		E	N	T	R	Y		P	O	I	N	T	S
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

-			-	D	I	M		E	N	T	R	Y		N	U	M	B	E	R	-
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

				-	3	0	0
45	46	47	48	49	78	79	80

Code the number of entry points in cc 25-26. Code the dim entry number of the subroutine set in cc 45-48.

Padding Character Card

When blank fields in the input cards are detected and in some cases when cards are missing, the field in the output will be printed. If this card is not included, the padding character is assumed to be a blank. However, by including this card any legitimate character may be used as a padding character. Format:

P	A	D	D	I	N	G		C	H	A	R	A	C	T	E	R	-		-	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

4	0	0
78	79	80

Code the padding character to be used in cc 19.

Conversion Descriptions

Field to Field Conversion Card

This card is used when a field somewhere in the input is to be transferred directly without change to some location in the output card. Format:

C	O	N	V	E	R	S	I	O	N	-		1	-	C	H	A	R	S	-			-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

F	R	O	M		C	A	R	D		T	Y	P	E	-						-
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

C	C	-			-	T	O		C	C	-			-
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

Code a 2, 4, 6, or 8 in cc 12 according to the rules described in the abstract. Code the number of characters in the input field in cc 21-22. Code the card type containing the input field, if applicable, in cc 39-43, right justified. Code the card column of the units position of the input field in cc 48-49. Code the card column of the units position of the output field in cc 57-58.

Field through Table to Field - FTTTF

This card is used when a field somewhere in the input is to be looked up in a table and the corresponding table entry is to be transferred to some location in the output. Format:

C	O	N	V	E	R	S	I	O	N	-		2	-	C	H	A	R	S	-			-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

F	R	O	M		C	A	R	D		T	Y	P	E	-						-
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

C	C	-			-	T	O		C	C	-			-
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

C	H	A	R	S	-			-	E	N	T	R	I	E	S	-				-
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80

Code a 2, 4, 6, or 8 in cc 12 according to the rules described in the abstract. Code the number of characters in the input field in cc 21-22. Code the card type containing the input field, if applicable, in cc 39-43, right justified. Code the card column of the units position of the input field in cc 48-49. Code the card column of the units position of the output field in cc 57-58. Code the number of characters in the output field in cc 66-67. Code the number of table entries there are in cc 77-79.

Table Entry Cards

If a "Field through table to field" conversion card is used, it must be followed by table entry cards. The number of table entry cards that must be coded is equal to the number coded in cc 77-79 of the FTTTF card. Each table entry card contains the field to be looked up and the corresponding field to be placed in the output area.

Code the field being looked up beginning in cc 1. The length of this field should correspond to the number coded in cc 21-22 of the FTTTF card.

Code a comma (,) in the card column immediately following the field being looked up.

Beginning in the card column after the comma, code the corresponding field to be placed in the output area. The length of this field should correspond to the number coded in cc 66-67 of the FTTTF card.

Field through Subroutine to Field Card

This card is used when a field somewhere in the input is to be used by a user written subroutine to determine a field to be transferred to some location in the output. Format:

C	O	N	V	E	R	S	I	O	N	-		3	-	C	H	A	R	S	-			-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

F	R	O	M		C	A	R	D		T	Y	P	E	-						-
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

C	C	-			-	T	O		C	C	-			-
45	46	47	48	49	50	51	52	53	54	55	56	57	58	59

C	H	A	R	S	-			-	E	N	T	R		P	T	-			-	-
60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80

Code a 2, 4, 6, or 8 in cc 12 according to the rules described in the abstract. Code the number of characters in the input field in cc 21-22. Code the card type containing the input field, if applicable, in cc 39-43, right justified. Code the card column of the units position of the input field in cc 48-49. Code the card column of the units position of the output field in cc 57-58. Code the number of card columns in the output field in cc 66-67. Code the number of the subroutine entry point to be used in entering the subroutine in cc 77-78.

Immediate Entry to Field

This card is used when a certain constant of information is to be placed directly into the output card regardless of the input. Format:

C	O	N	V	E	R	S	I	O	N	-	0	1	-	C	H	A	R	S	-	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

	-	T	O		C	C	-			-
22	23	24	25	26	27	28	29	30	31	32

Code the number of characters to be transmitted to the output in cc 21-22. Code the card column of the units position in the output area where the characters are to be transferred in cc 30-31. Beginning in cc 33 code the characters to be transmitted to the output. The number of characters should correspond to the number coded in cc 30-31.

Counters to Field Card

This card is used when: 1) the number of cards in the family, or 2) the number of cards of a particular card type, or 3) the sequency number of a card within a particular card type is to be transferred to some location in the output card. Format:

C	O	N	V	E	R	S	I	O	N	-	0	2	-	F	R	O	M		C	A	R	D
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

	T	Y	P	E	-						-	C	O	U	N	T	E	R	-	
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44

-	T	O		C	C	-			-
45	46	47	48	49	50	51	52	53	54

Code the card type if 2 or 3 is coded in cc 44. Code 1, 2, or 3 in cc 44:

1 means count of cards in family

2 means count of cards of a particular type

3 means sequency within a particular type

Code the card column of the units position of the counter field in the output in cc 52-53. All counters are 2 characters in length. Although, the leading character may be overlaid by another conversion.

Punch the Output Card

This conversion card is used when all fields in an output card are developed and it is to be punched. All output developed by conversion entries from the beginning of the conversion section or from the previous punch output card is punched. Format:

C	O	N	V	E	R	S	I	O	N	-	0	3	-	R	E	P	E	A	T	-		-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

L	A	S	T		E	N	T	R	Y	-		-
24	25	26	27	28	29	30	31	32	33	34	35	36

If the development of a particular output card is identical, except each time the next card in sequence within each card type is to be used in all conversions beginning with 6 or 8, then code 1 in cc 22 of this card. If the development is not to be repeated, code 0 in cc 22. If there are no more conversions for this family, code 1 in cc 35, if not the last code 0.

Subroutine Entry Point

If surboutines are required for any of the conversions to be performed by this library, Subroutine Entry Point card(s) must be used. One of these cards must be completed for each subroutine used. Format:

S	U	B	R	O	U	T	I	N	E		E	N	T	R	Y		P	O	I	N	T	-
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23

		-	A	D	D	R	E	S	S	-						-	-
24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41

Description:

In cc 24-25 code the entry point number. These numbers are to be sequential beginning with 01. The address of the entry point is coded in cc 35-39. This is the address of the first executable instruction.

Appendix 2

GFC Codes for Various Conversion Types

	CONVERSION CODES				
	Independent	Non-Indexing No Padding	Non-Indexing Padding	Indexing No Padding	Indexing Padding
FIELD TO FIELD		21	41	61	81
FIELD TO TABLE TO FIELD		22	42	62	82
FIELD TO SUBR TO FIELD		23	43	63	83
FIELD IMMEDIATE TO FIELD	01				
CARD COUNTER IMMEDIATE TO FIELD	02				
PUNCH OUTPUT CARD	03				

SYSTEM DIRECTOR

Vernon T. Boyer
IBM Corporation
Dept. 237
Bldg. 062

11-22-66

SYSTEM DIRECTOR

The System Director forms the operating center of the TSX system. It has the responsibility of directing interrupt servicing, loading user core loads, directing time-sharing, servicing the interval timers and servicing error conditions. The System Director is core resident and is storage protected to ensure that it is not accidentally altered. When the computer is operating under control of the System Director control is passed to it by TSX calls, interrupts and errors. The System Director is that portion of TSX (other than I/O) that must be in core at all times in order to respond to the real time world. Basically, the System Director is made up of five control programs and two data areas.

Control Programs

- Program Sequence Control (PSC) - controls the sequencing and initiates the loading and execution of user-specified process core loads.
- Master Interrupt Control (MIC) - automatically determines the type of each interrupt as it is recognized and transfers control to the proper interrupt servicing routine.
- Interval Timer Control (ITC) - provides a programmed real-time clock, a timer for TSC, nine programmed interval timers, and control of two machine-interval timers.

- Time-Sharing Control (TSC) - controls the time-sharing of variable core between process and nonprocess core loads.
- Error Alert Control (EAC) - provides the following functions whenever an error occurs:
  1) optionally saves core for future reference
  2) optionally branches to a user's program for further error analysis
  3) prints an error message
  4) executes a specified recovery procedure

Core Layout of the System Director is as follows:

ICLT Address
Disk Save Area Addresses
Error Work Level
Queue Table
ICLT
Work Areas
MIC
PSC
TSC
ITC
EAC

<u>ICLT Communication Table.</u> The ICLT Communications Table contains the address of the ICLT entry for each interrupt level defined in the system. Since the MIC program does not reside in a fixed area of core, these addresses are made available by means of this table to various routines and subroutines which are sent to the user in object format.

<u>Save Area Table.</u> The Save Area Table contains the word counts and sector addresses of the various save areas on disk. It also contains the disk address of the Error Disk Program (EDP) called by EAC. These disk addresses are obtained from FLET by the Skeleton Builder when the System Skeleton is built.

<u>EAC Work Area.</u> This work area is used by EAC when processing errors.

QUEUE TABLE

Priority	Word Count	Sector Addr.	Priority	Word Count	Sector Addr

Queue Table. The Queue Table is a table of 3-word entries, one for each unique call to the QUEUE subroutine. A unique entry is one in which the sector address and/or the priority are different from any other in the queue. Each entry appears as follows:

Word	Contents
1	Priority
2	Word count of the core load
3	Disk address of the core load

The size of the Queue Table is defined by the user. Entries are removed from the Queue Table by the UNQ and VIAQ subroutines.

Interrupt Core Load Table (ICLT)

IN SKELETON
IN MAINLINE
RECORD
RECORDED
WORD COUNT OR START ADDR
SECTOR ADDR

Interrupt Core Load Table (ICLT). The interrupt Core Load Table contains an entry for each interrupt level assigned by the user. The size of the entry for each level is determined by the user. The format of an ICLT entry is as follows:

Word	Contents
1	In-core-with-Skeleton indicator word. A bit is set on in this word corresponding to each PISW bit which is serviced by an interrupt servicing routine included in the Skeleton.
2	In-core-with-mainline indicator word. This word is filled in by the PSC program as each core load is loaded. It is obtained from word 2 of the IST entry for the same interrupt level.
3	Record indicator word. This word is filled in by the PSC program as each core load is loaded. It is obtained from word 3 of the IST entry for the same interrupt level.
4	Recorded indicator word. A bit is set on in this word by MIC whenever an interrupt is recorded. The bit set on corresponds to a bit in the Record indicator word (word 3).
5-6	If the program is not in-core-with Skeleton, these two words contain the word count and sector address of the program servicing the interrupt associated with the first PISW bit on this level. If the program

Word	Contents
	is in-core-with Skeleton, word 5 equals the starting address of the program in cire. Word 6 is unused.
7-8	Same as above for the second PISW bit.
n-n+1	Same as above for the last PISW bit on this level.

LEVEL WORK AREA

-4 TO 40	41 57	58 99
MIC	FAC	REENTRANT SUBROUTINE
	QZSAV	WORK AREA
	TVSAV	
	FORTRAN ERROR	

Work Areas. Each interrupt level specified by the user has a work area. The area is 104 words in length (subject to adjustment by the user). In addition to the areas for user-assigned interrupt levels, there is a work area for a process core load. The space in the work area for MIC is not subject to adjustment. The contents of a work area are as follows:

Word	Contents
-4-40	Save locations for Accumulator, Extension, index registers, Status; constants, work area interrupt processing before entry to MIC.

Word	Contents
41-57	Save locations for FORTRAN FAC, and error indicators; save locations used by TVSAV and QZSAV.
58-99	Available words for use by other reentrant coded programs and subroutines.

MASTER INTERRUPT CONTROL PROGRAM

The master interrupt control (MIC) program controls the servicing of interrupts. An interrupt may occur at any time but it will not be recognized by MIC unless the interrupt is on a level that is not masked and is of a higher priority than the present level of machine operation. The user-specified assignment for interrupt levels determines the priority of a particular interrupt. The user-assigned interrupts can be delayed from being recognized by masking the level to which they are assigned. The servicing of process and programmed subroutines can also be delayed by recording their occurrence.

Basically, there are two types of interrupts: internal and external. Internal interrupts are those associated with any input/output device, interval timer, trace, or error condition. Internal interrupts, except trace, are serviced by IBM-provided subroutines as soon as they are recognized. Programmed interrupts are treated the same as external interrupts.

Interrupt Servicing

In the servicing of interrupts the following applies:

1. Only one ILSW bit is serviced per entry to a level.
2. Programmed interrupts are serviced whenever a "no bit" condition occurs in an ILSW or whenever an exit from either an I/O servicing routine or a process interrupt servicing routine occurs.
3. All bits on in a PISW are serviced before exiting from a level and before servicing of a programmed interrupt.
4. Interrupts on levels that are serviced by out-of-core interrupt core loads are serviced in the masked mode so that they cannot be interrupted by another interrupt serviced by an out-of-core routine. Only one level of exchange is maintained.

EXTERNAL INTERRUPTS

External interrupts are those associated with the process and programmed interrupt features. They are serviced or recorded by one of four types of user-written routines: (1) Skeleton interrupt routine, (2) Mainline interrupt routine, (3) Interrupt core load, or (4) Mainline core load.

The different types of routines are provided to permit flexibility in the use of core storage, and in the response time requirements of a specific interrupt (i.e., the time required to enter an interrupt routine after the interrupt is recognized).

Interrupt routines are assigned to the skeleton area by control cards when the system skeleton is initially assembled. They are normally used to service process interrupts that require immediate response, have high priority, or that occur frequently.

Skeleton interrupt routines are required only if the user considers it necessary for the routine to always be in core storage.

External interrupts not serviced by skeleton interrupt routines can be serviced by routines included as part of a mainline core load. The response time of a mainline interrupt routine approaches that of a skeleton interrupt routine only if the mainline core load containing the interrupt routine is in core when the interrupt occurs.

A mainline core load is required for the servicing of each external interrupt that might be recorded and serviced at a later time.

Interrupt core loads are required for those interrupts that meet either of the following conditions.

1. User specifies the interrupt servicing routine to be out of core.
2. User specifies the interrupt servicing routine to be in core as part of a mainline core load.

If a time-sharing operation in in progress when an interrupt occurs, the interrupt (if not recorded) is serviced with the skeleton interrupt routine, if it exists, or with the interrupt core load. Even if the mainline that called for time-sharing has an interrupt routine for the interrupt that occurred, the interrupt core load associated with that interrupt is brought in (to core) for the servicing.

When recognized, external interrupts may either be recorded or serviced, as specified by the user. If recognition is recorded, it can be serviced later or cleared.

If not recorded, external process interrupts are serviced as soon as one of the following conditions becomes true.

1. The servicing routine is located within the system skeleton, the interrupt level is not masked, and an interrupt of higher priority is not being serviced.
2. No other external interrupt is being serviced, and the servicing routine is in core as part of the core load.
3. No other external interrupt is being serviced, the servicing routine is out of core, and no I/O operation is in progress unless

its associated interrupt routine and I/O area are in the skeleton. This requires an exchange operation (an operation wherein a specified portion of the variable area of core is saved and the interrupt core load is read in for execution). Following execution of the interrupt core load, the original operating program is restored.

The option of recording or servicing any external interrupt may be different from one mainline core load to the next. The designation is made by control cards when the core load is being formed.

Programmed Interrupt - CALL LEVEL

This statement causes an interrupt (by programming) on any assignable interrupt level (0-23). The format is:

CALL LEVEL (I)

where

I is an integer constant (0-23) that specifies the interrupt level desired.

This call, which can be used only in process (mainline or interrupt) programs, causes a pseudo ILSW bit to be set on the level specified.

Programmed interrupts are treated the same as process interrupts in that they can be recorded or serviced, in-core or out-of-core, etc.

The programmed interrupt servicing routines must follow the rules of process interrupt servicing routines. There can be only one programmed-interrupt routine per assignable interrupt level.

The programmed interrupt is recognized immediately when called from a lower level. When the servicing routine exits to MIC, program operation at the calling level is resumed with the statement following the CALL LEVEL statement.

A programmed interrupt called from a higher level is recognized after the calling program is completed and after any intervening interrupts are service. If a level is called and any ILSW bit is on when the level is recognized, the programmed interrupt is recognized after the first ILSW bit that is on is serviced.

Interrupt Exit-CALL INTEX

All interrupt routines serviced on an interrupt level must return control to MIC. The CALL INTEX statement, which has no parameters, is normally used for this purpose. It must be used as the last logical statement in skeleton interrupt routines, mainline interrupt routines, and it can be used in interrupt core loads.

RECORDED INTERRUPT SERVICING

External interrupts whose occurrences are recorded are serviced with mainline core loads. The mainline core load performing the servicing is the same as any other mainline core load, except it is queued for execution by a CALL QIFON statement. Since it is a queued core load, it should have a CALL VIAQ as the last logical statement. (It could, of course, be the first core load of a special series and, as such, would end with CALL CHAIN to get the next core load of a sequence, but a CALL VIAQ eventually must be executed.)

COMBINATION CORE LOAD

In the descriptions given thus far there is only one major difference between an interrupt core load and a mainline core load used for servicing recorded interrupts. That difference is in the last logical statement, which must be CALL INTEX for an interrupt core load and CALL VIAQ for the mainline core load.

If an external interrupt is serviced immediately some times and recorded other times, it requires two core loads that might be the same in all respects, except for their last logical statement. To eliminate this situation, a combination exit statement, CALL DPART, is provided.

Departure- CALL DPART

The CALL DPART statement causes the level of operation to be tested and

1. If the present level is an interrupt level, a CALL INTEX is executed.
2. Otherwise, a CALL VIAQ is executed.

Thus, CALL DPART eliminates duplication of core loads. An interrupt that is sometimes recorded and sometimes serviced, when it occurs, can be serviced under either condition with the same core load. The core load operates from an interrupt level when servicing is specified; it is queued and operates from the mainline level when the interrupt is specified as recorded.

INTERRUPT ASSIGNMENT RESTRICTIONS

The following interrupt assignment restrictions must be observed for proper operation of the TSX system.

1. All I/O device interrupts must be assigned to a higher priority interrupt level than external interrupts unless the external interrupt is serviced by a skeleton interrupt routine.
2. If external interrupts and I/O devices are both assigned to the same level, the external interrupts must be serviced by skeleton interrupt routines.

3. A skeleton interrupt routine cannot use an I/O device whose interrupt is assigned to the same or a lower priority level, except for disk, 1053 printer, and 1443 printer; however, the 1053 test function cannot be used.
4. ILSW bits must be assigned continuously beginning with position 0.
5. FORTRAN READ and WRITE statements (except for disk) can be used only on interrupt levels lower than the 1816.

PISW ASSIGNMENT RESTRICTIONS

PISW (Process Interrupt Status Word) groups can be assigned to interrupt levels either as a single group per level or in multiple groups per level. The following rules and restrictions must be observed for proper operation of the TSX system.

One Group Per Level

Normal usage of process interrupts requires that only one group of process interrupts be assigned to each interrupt level. Process interrupts assigned in this way can each be serviced with separate interrupt routines. The servicing routines must reside in the skeleton area only if their associated interrupt level is the same as or higher than any I/O device interrupt level.

When only one PISW is connected to a level, the correlation of the interrupt level number to the PISW group number is as follows.

Interrupt Level	PISW Group
0	1
1	2
2	3
3	4
.	
.	
22	23
23	24

The MIC program does the ILSW and PISW sensing and transfers control to the proper interrupt servicing routine.

Multiple Groups Per Level

In special cases it is desirable to have more than one PISW group assigned to an interrupt level and this is possible with the TSX system; however, the following restrictions must be observed. The interrupt servicing routine must:

1. Reside in the skeleton area.

2. Sense all non standard PISWs assigned to the level.

3. Upon completion, exit to MIC via the I/O exit (BSC I 90).

When assigned in this way there is no correlation restriction between the interrupt level number and PISW group number.

Combination PISW Assignments

It is possible to combine the two assignment methods and have some interrupt levels with only one PISW each and some levels with more than one PISW. The same rules and restrictions for each type as outlined above still apply. For example, to have to groups of four PISWs each assigned to interrupt levels 4 and 5 one valid combination is:

Interrupt Level	PISW Group
0	1
1	2
2	3
3	4
4	one group of four PISWs
5	one group of four PISWs
6	7
7	8
.	.
.	.

(Continued)

Interrupt Level	PISW Group (Cont'd)	
17	18	
18		
19		Not assignable; usage assumed on levels 4 and 5.
23		

Any combination can be used for the PISW assignments on levels 4 and 5. The user written routines used to service the interrupts must be coded as an I/O RPQ subroutine.

PROGRAM SEQUENCE CONTROL PROGRAM

Program sequence control (PSC) is a control program that handles the flow of control from one mainline core load to the next. PSC functions are initiated by execution of PSC CALL statements in the user's program. The specific functions of PSC are:

1. Execute the next sequential mainline core load. The new core load overlays the one that contained the call.
2. Save the mainline core load in progress (on disk) and load a special core load for execution.
3. Restore the core load that was saved in item 2 and continues execution from where it left off (the statement following the CALL SPECL).

4. Queue mainline/core loads associated with interrupts whose occurrence has been recorded.

5. Execute the highest priority mainline core load listed in the core load queue.

6. Insert mainline core load entries into or delete them from the core load queue.

For PSC to perform the above functions, a CALL statement must be executed for each one. The specific CALL statements and their parameters are described below.

Functionally, the CALL statements are divided into two groups: those for direct sequence, in which one mainline core load calls another, and those for queuing, in which either the highest priority mainline core load named in the queue is called or the core load queue is modified by inserting or deleting an entry.

The CALL statements from both groups provide the user with the flexibility or implementing his unique scheduling requirements. Unless otherwise stated, the results are unpredictable if these CALL statements are used in a nonprocess program or if they are used incorrectly in process core loads.

DIRECT SEQUENCE STATEMENTS

The direct sequence statements are used to

- CALL the next mainline core load to be executed.
- Save the present mainline core load (on disk) and CALL a special mainline core load for execution.
- Restore and continue execution of the saved mainline core load.

Normal Call - CALL CHAIN (NAME)

This call terminates execution of the mainline core load and transfers control to PSC, which loads the named mainline core load into core storage for operation. This is the last logical statement in a mainline core load; it calls the next mainline core load into operation.

NAME is the name of the mainline core load being called.

Special Call - CALL SPECL (NAME)

This call suspends execution of the current mainline core load and transfers control to PSC, which

1. Saves the return address (i. e., the address of the instruction following the CALL SPECL statement).
2. Stores the current mainline core load on disk.
3. Loads and transfers control to the new (special) mainline core load.

NOTE: Only one mainline core load can be saved. Thus, if CALL SPECL is used in a core load that was called by a CALL SPECL, the mainline core load saved originally is lost.

Return Saved Mainline - CALL BACK

This statement is normally used as the last logical statement in a special mainline core load. When executed, it terminates the core load and transfers control to PSC which restores the last previously saved core load. Execution of the saved core load commences with the statement that follows the CALL SPECL statement.

QUEUING STATEMENTS

The queuing statements are used to

- Insert entries into or delete entries from the core load queue table.
- Execute the highest priority mainline core load specified in the queue.

These functions are performed by subroutines, which can be located with either the calling program or the system skeleton.

Insert Into Queue - CALL QUEUE

This statement is used to place a mainline core load name and priority in the queue. If the same name and priority are already queued, they will not be placed in the queue a second time; however, the same name with a different priority can be inserted into the queue. The format of the statement is:

CALL QUEUE (NAME, P, E)

where

NAME is the name of a mainline core load that is to be entered into the queue.

P is the integer expression that specifies the execution priority for the core load. One (1) is the highest priority number. The allowable range of P is from 1 to 32767.

E is an error parameter used to specify the action to be taken when the queue is full.

E = 0. Ignore call after printing an error message.

E = 1 through 32766. Replace the lowest priority entry in the queue with the name and priority in this call, if the priority of the queue entry is loser (numerically larger) than E. If there is no queue entry with a lower priority, the restart core load specified for this core load is executed.

E = 32767. Execute restart core load.

Different core loads can be assigned the same priority number, if desired. When two or more queue entries have been assigned the same priority, these entries have a priority among themselves on a first-in-first-out basis.

Delete From the Queue - CALL UNQ (NAME, P)

This statement is used to delete a mainline core load entry from the queue. If the name and priority parameters do not match a queue entry, the statment has no effect. The CALL UNQ statement can be used in any program.

Execute Highest Priority Core Load - CALL VIAQ

This statement is used as the last logical statement in a mainline core load. It terminates the present core load and causes execution of the highest priority core load named in the queue.

When the CALL VIAQ statement is executed and there are entries in the queue, the highest priority entry is removed, and used to call the core load it references.

If there are no entries in the queue, the process is considered to be in an idle condition (i. e., the process does not require any action at this time). Since variable core is not being utilized by process core loads, control is transferred to time-sharing control (TSC) for execution of nonprocess core loads. The time-sharing operation will continue for the period of time specified at assembly time or until terminated by an interrupt (see CALL ENDTS statement in the Time-Sharing Control Program section). A CALL VIAQ operation is automatically performed when the time-sharing time is terminated.

Therefore, if an interrupt program has placed a name in the queue, the named core load will then be automatically executed. (This is not true if time-sharing was initiated by a CALL SHARE statement.)

The CALL VIAQ statement can be used only in mainline core loads.

Queue Core Load If Indicator is ON - CALL QIFON

This statement is used to place a mainline core load name and priority in the queue table if its associated recorded interrupt indicator is on. Recorded interrupts are those that do not require service when they occur and can be recorded for servicing at a later time.

When an interrupt that is to be recorded is recognized by MIC, the interrupt is reset and a programmed indicator is set. It is the programmed indicators (set by MIC) that the QIFON subroutine interrogates. The QIFON subroutine than automatically clears the interrogated indicator. The statement format is:

CALL QIFON (NAME, P, L, I, E)

where

NAME is the name of a mainline core load.

P is the execution priority to be assigned to the mainline core load named.

L is the interrupt priority level or indicatr (see L and I Combinations).

I is the PISW bit position indicator or CALL COUNT indicators (see L and I Combinations).

E is an error parameter used to specify the action to be taken when the queue is full.

E = 0. Ignore call after printing an error message.

E = 1 through 32766. Replace the lowest priority entry in the queue with this call if the priority of the queue entry is lower than E. Restart if there is no entry lower than E.

E = 32767. Execute restart core load.

L and I Combinations. The combination of L and I are

L	I	Reference
0-23	0-15	Process interrupts
0-23	-n	Programmed interrupts (see CALL LEVEL)
-n	0-31	Subprogram number of CALL COUNT statement (see Programmed Timers).

(-n means any minus number)

The CALL QIFON statement can be used in any process program.

Clear Recorded Interrupts - CALL CLEAR

The CALL CLEAR statement is used to clear the recorded interrupt indicators. In this way, specific interrupts or all external interrupts can be removed from their recorded status. The format of the statement is

CALL CLEAR (M, L, I, L, I. . . .)

where

M is an integer constant that specifies the number of parameters to follow.

If M equals 0, all indicators specifying recorded status are cleared.

L and I are the same as for the Call QIFON statement.

TIME-SHARING CONTROL PROGRAM

The time-sharing control (TSC) program controls the amount of time allowed for nonprocess program operations.

Time-sharing can be initiated in two ways:

1. Execution of a CALL SHARE statement in a process mainline program.
2. Execution of a CALL VIAQ statement when the core load queue table is empty. This causes the VIAQ subroutine to execute a CALL SHARE statement.

The first method can be utilized when time-sharing is desired at specific times and for different durations. When time-sharing is initiated in this way, the process core load is saved and the non-process monitor (or an unfinished program is read into core and executed. When the specified amount of time has elapsed, the nonprocess program is saved (if not completed) and the process core load is restored. The maximum time for a time-sharing operation initiated in this manner is set by each CALL SHARE statement. Operation of the process core load is resumed with the statement following CALL SHARE.

The second method permits time-sharing when the computer is not being utilized for the process. The maximum time for a time-sharing operation initiated in this manner is specified in a control card by the user when the

system is loaded, and remains constant. At the completion of the specified time, another CALL VIAQ is automatically executed by the system. If, in the interim a core load has been queued, it is then executed; however, another time-sharing operation will be initiated if nothing has been entered into the queue.

Normally the CALL VIAQ method is used, but in special cases, the CALL SHARE method is also desirable.

All interrupts that occur during the time-sharing operation are handled by MIC the same as if a process mainline program were in operation. After the interrupt is serviced (or recorded) control is returned to the nonprocess program unless a CALL ENDTS statement is executed in the interrupt routine.

If the nonprocess program is not completed before time runs out, it is saved and continued when the next time-sharing operation is executed.

The following statement is used to initiate time-sharing operations for a specified time interval.

CALL SHARE (I)

where

I is an integer expression that specifies the number of time intervals allowed for nonprocess program operation.

CALL ENDTS

This call can be used only in an interrupt routine, and it sets the time-sharing clock to indicate zero time. The first timer C interrupt that occurs that checks the programmed timers after control is returned to the nonprocess program causes the time-sharing operation to be terminated and control is then returned to the process mainline program.

If time-sharing is not in effect, the CALL ENDTS statement has no effect.

Two additional functions performed by TSX are CALL LINK and CALL EXIT when called from nonprocess programs.

INTERVAL TIMER CONTROL PROGRAM

The interval timer control (ITC) program provides for FORTRAN language control of four types of timers:

1. Two machine interval timers (A and B).
2. Nine programmed interval timers.
3. A programmed real-time clock.
4. A timer for time-sharing control.

The ITC also performs three additional functions.

1. Resets the operation monitor during time-sharing.
2. Tests for no response from 1053 printers.
3. Performs end of time-sharing.

The third machine interval timer (C) is used for items 2, 3, and 4.

Machine Interval Timers

The two machine interval timers should be used to measure relatively short time intervals. They are controlled by the following statement.

CALL TIMER (NAME, I, INT)

where

NAME is the name of the user's subprogram to be executed when the specified time elapses (NAME must also appear in an EXTERNAL statement;

I is an integer expression, whose value must be:

1-for machine interval timer A. (word 4)

2-for machine interval timer B. (word 5)

INT is a positive integer expression that specifies the number of intervals counted before the user's subprogram is executed.

The subprogram specified in a CALL TIMER statement must be in core storage when the interrupt generated by the timer is recognized. The interrupt occurs when the time specified has elapsed, but is is recognized only when the level of operation is lower than the timer interrupt level and the timer level is unmasked. The timers are stopped and reset to zero when the specified time has elapsed and the interrupt is recognized (zero is a not-busy condition).

It is the user's responsibility to ensure that the subprogram NAME is in core when the timer interrupt is recognized. This can be accomplished in two ways:

1. NAME skeleton subroutine.
2. NAME is a mainline only routine, all interrupt levels with out-of core interrupts must be masked, and the core load exit is not allowed while the timer is busy.

The subprogram name is automatically loaded with the calling core load (unless previously loaded with the system skeleton). Also, the subprogram must return control to the ITC program (RETURN statement or assembler language equivalent). The program is executed at the interrupt level to which the interval timers are assigned and cannot be recorded.

It is not recommended that periodic programs (programs initiated by internal timers) be executed on the timer level. If this is allowed to happen, some timer interrupts may be missed during execution of lengthy programs. The CALL LEVEL statement (see Programmed Interrupt) is designed to handle this situation, and in this case, should be used to create an interrupt at a lower level of machine operation. The periodic program is then executed at the programmed interrupt level.

Example: Assume machine interval timer A is wired for the .125 ms time base.

CALL TIMER (SCAN1, 1, 35)

When this statement is executed, ITC initializes timer A (sets it to -35) to count 35 intervals and return control to the statement following the CALL TIMER statement. When 35 intervals (i.e., 35 x .125 ms, or 4.375 ms) have elapsed, an interrupt occurs and control is transferred to the subprogram named SCAN1.

Clock and Programmed Timers

The programmed real-time clock and the nine programmed interval timers are updated by the third interval timer (C).

The time interval used for updating the clock (termed the interrupt time base) is the product of the wired-in time base interval and a number chosen by the user at system generation time. For example, assume interval timer C is wired for an 8-ms time base, and the clock is to be updated every second. The number necessary to accomplish this is 125 (8 ms x 125 = 1000 ms, or 1 second), and when the ITC program is assembled, the number -125 would be specified by the user. The third timer would then cause an interrupt every second. A minus number must be specified because the interval timer is incremented and causes an interrupt when it reaches a zero.

Summary. Interrupt time base = wired-in time base x assigned number.

The interrupt time base is used specifically for the programmed real-time clock and as a primary base for the programmed timers and time-sharing clock. It is also used as the reset interval for the operation monitor during time-sharing operations.

Clock

The programmed real-time clock maintained by the interval time control program records time to the nearest thousandths of an hours. Clock accuracy depends on the assigned interrupt time base previously described. The clock is reset on a 24-hour basis (i. e., it is incremented from 00:000 to 23:999 and then goes to 00:000).

To set the clock at a desired time, the following statement is used.

CALL SETCL (I)

where

I is an integer expression that specifies the desired time setting. The time setting must be expressed in hours and thousandths of hours (i. e., 00000 through 23999).

To read the programmed real-time clock, the following statement is used.

CALL CLOCK (I)

where

I is an integer variable where the time is to be stored.

Programmed Timers

The nine programmed timers should be used to specify long time periods. In particular, they can be used for periodic program execution or to initiate execution of a program at some later time.

If the called program is in the skeleton when the specified time elapses, the program is executed. The called program must return control to the ITC program (RETURN statement or assembler language). The program is executed on the same level that the interval timers are on.

If the called program is not in the skeleton when the specified time elapses, it must be in the same form as a mainline core load. Out-of-core programs are handled as recorded interrupts, i. e., the program will not be placed in the queue until requested by a CALL QIFON statement, and will not be executed until a CALL VIAQ finds that the queued program is the highest priority in the queue.

To provide the user with large time intervals, a larger time base can be specified for the programmed timers. The programmed timer base for the programmed timers is a user-assigned multiple of the interrupt time base used for the programmed real-time clock. For example, if the interrupt

time base is one second, and the user wants the programmed timers to operate at 15 second intervals, then 15 is specified when the ITC program is assembled.

Summary. Programmed timer base=interrupt time base (previously assigned) x assigned number.

The programmed timer base is used specifically for the programmed timers and the time-sharing clock. This base is the smallest interval of time that can be specified for the programmed timers or for nonprocess program operation (i. e., time-sharing, see Time-Sharing Control Program).

The programmed interval timers are controlled by the following statement.

CALL COUNT (IN, I, INB)

where

IN is an integer constant or integer variable that specifies the number (0 through 31) of the program to be executed or recorded when the specified time elapses. These numbers are assigned by the user when the skeleton executive is prepared.

I is an integer expression that specifies the number (1 through 9) of the programmed interval timer.

INB is an integer expression that specifies the number of intervals to be counted **before** the called program can be executed (multiple of programmed timer base). Program numbers are used in place of names to provide the recorded interrupt option.

ERROR ALERT CONTROL PROGRAM

When an internal error occurs, or when an I/O subroutine detects an I/O error or invalid interrupt condition, or when a system subroutine detects an invalid call or system failure, the EAC **program** is called to process the error and select a recovery procedure.

The EAC program is comprised of four logical segments:

-EAC In-core

-Error Disk Program (EDP)

-Error Decision Subroutines

-EAC Exit

The Error Alert Control (EAC) program receives control from

1. Any input/output subroutine when the subroutine cannot correct an error or interrupt condition
2. The queue subroutine when the core load queue table would overflow

3. The master interrupt control program when an internal machine error occurs (i. e., invalid operation code, parity, or storage protect violation)
4. Other control programs

Upon entry (EAC is entered at word 120), EAC receives the error identification and other pertinent data. From this information, the core and disk portions of EAC will perform the following operations:

1. Optionally, dump core storage to disk (not performed for internal machine errors).
2. a. If in a nonprocess program terminate the program if the error cannot be operator corrected.

   b. If in a process program branch to the user-written error subroutine that is with the core load (this step is bypassed for internal machine errors or if an error subroutine is not included).
3. Update error counters maintained on disk.
4. Execute a subroutine (IBM written) for the device or error condition, print an error message on the EAC printers, and set up possible recovery action.

   a. If in a nonprocess program returns control to the monitor supervisor program if recovery is not possible.

b. If in an interrupt routine terminate the program, service any other interrupts, and perform the action specified by the user or the device error subroutine.

c. If in a mainline core load perform the action specified by the user or the device error subroutine.

When the EAC program is initially assembled, the option of the core dump in item 1 can be selected.

Also, when assembling the skeleton programs, a back-up unit of the same type can be specified for 1053, 1816, and 1443 printers. Backup for the EAC printer is achieved by defining multiple EAC printers at TASK assembly time (if the EAC printer is defined as a 1053). When an output error occurs, or if the unit is not ready (interrupt response not received), EAC will logically disconnect the unit in error and substitute the back-up unit. When backup is initiated because of a hardware malfunction, the message in progress on the failing unit is not continued on the backup device. When the error condition is corrected, the unit can be restored to its original status by using the C. E. interrupt routine.

Error Subroutine

A user-written error subroutine can be optionally included with each process core load. The purpose of this subroutine is to allow the user to have control before EAC overlays the variable area with the disk portion of EAC.

For example, there may be special data or other information that the user wants to save. Output, such as special core dumps, messages, or contact operate functions, can also be executed. The error subroutine cannot be written in FORTRAN language.

SYSTEM DIRECTOR ERROR ALERT CONTROL

The Error Alert Control (EAC) section is entered when an error occurs or when a condition arises that requires an operator's intervention. A message is printed on the EAC printer and EAC then takes one of four exits. Each error is analyzed in order to decide which exits may be taken for that error. Where more than one exit exists for a given error, the user can specify the desired exit in the user's error subroutine.

Error Exits

1. Continue at the point of interrupt - When an I/O device finishes its operation an interrupt is generated. This interrupt causes control to be transferred to the I/O interrupt routine when then determines the error condition and branches to EAC. This exit option then returns control to the point in the program at which the interrupt occurred (i. e., EAC does not return to the I/O interrupt routine).

2. Return to the routine which detected the error.
3. Restart - EAC or the user's error subroutine has decided that the present core load cannot be continued. The core load specified as the restart core load when the present core load was loaded to disk is given control when the mainline level is reached.
4. Reload - When an error occurs which indicates that a part of the skeleton may have been destroyed. EAC reloads the skeleton into core and transfers control to the core load specified on the cold start card.
5. Cold start. User must reload the skeleton using cold start cards.

Not-Ready

A not-ready condition is processed by EAC in one of three ways. The method is dependent upon the type of device.

The first method of processing the not-ready consists of printing the message and returning to the I/O routine which again senses the device. If the device is still not-ready, another entry to EAC is made to print the message and return. EAC keeps count of the consecutive not-ready for a device and, on the fourth attempt, terminates the core load by causing a restart.

The second method concerns the 1053 and 1443 printers. When the fourth not-ready check has been made, EAC attempts to logically replace the not-ready unit with its backup unit and return to the I/O routine. If there is no backup unit specified, an indicator is set so that all future calls to that printer will be ignored. If the printer is a 1443, EAC exits through restart (return to the I/O routine user exit option).

The third method concerns the card read punch unit. After the message is printed and control is returned to the I/O routine, the I/O routine waits in an unmasked mode until the unit is made ready.

SYSTEM DIRECTOR ASSEMBLY REQUIREMENTS

The System Director must be assembled and stored before the TSX system skeleton can be built. The user must fill out the supplied EQU cards for system director in order for it to be tailored to his specifications.

System Director EQUATE Cards

Ni100 to Ni123

Meaning

Labels NIL00 thru NIL23 define PISW 1 thru 24, respectively. XX equals 1 plus the highest numbered PISW bit assigned to a process interrupt. XX equals 0 if no process interrupts are assigned to a level.

For example, level 02 has process interrupts wired so that when sensed, using IOCC for PISWs, the following bit configuration would appear in the A-register if all bits were on.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0

In this example NIL02 would be equated to 7 (rightmost but assigned plus one). Note that although process interrupts are assigned in groups of 4, the highest bit position actually used (+1) determines XX.

Use00 to Use23

Labels USE00 through USE23 define the level work areas. If a label is equated to zero, no work area is included for that level. If it is equated to one, a work area is included on that level. For example, if NULEV is equated to five, USE00, USE01, USE02, USE03, and USE04 must be equated to one and USE05 through USE23 are equated to zero.

Use 014 is used to org out the XIO's for level 14-23 if it is set to zero.

NB00 to NB23

Labels NB00 through NB23 are equated to the rightmost bit plus one that is assigned to an ILSW for a level. For example, if level 02 has a process interrupt group 1 and two input/output devices assigned the following bits would appear in the A-register if all bits were on (when sensed by the IOCC for ILSW).

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0

In this example, NB02 would be equated to 3. If there are no bits on a level, the label must be equated to zero.

NULEV	EQU	1 through 24	Number of interrupt levels to be compiled in the system director. This value is 1+ the highest numbered interrupt level used. If levels 0-9 are used, the NULEV is equated to 10.
NUQUE	EQU	XXX	This label defines the number of entries to be allowed in the QUEUE table. This number should be large enougn so that the QUEUE table will not overflow under normal conditions. Three words are required for each entry point.
NITP1	EQU	1 to 16	Number of CALL COUNT subroutines 0-15.
NITP2	EQU	1 to 16	Number of CALL COUNT subroutines 16-31.
NLWS1	EQU	1 to 14	Number of levels for the programmed interrupts 0-13.
NLWS2	EQU	1 to 10	Number of levels for the programmed interrupts 14-23.
VCORE	EQU	XXXXX	The starting address of variable core. This address must be even.
CBASE	EQU	XXXXX	The number of times the time clock is updated before the programmed timers are updated.

TBASE	EQU	-XXXXX	The negative number put in timer C (word 6) to be counted down for the time clock base. For example, if the clock is to be updated each second when the hardware base is 8 milliseconds, TBASE would be equated to -125.
OPMO1	EQU	0 or 1	This label is equated to zero if the user is to reset the operations monitor, or equated to one if the operations monitor is to be reset by ITC when time-sharing is in progress.
TIMES	EQU	0 or 1	This label is equated to zero if time-sharing is not to be used or to one if it is to be used.
TIME1	EQU	/XXXX	Labels TIME1 and TIME2 are a hexadecimal equivalent of a double precision number which specifies the time in milliseconds that is calculated by the equation TBASE *HARDWARE BASE for timer C.
TIME2	EQU	/XXXX	TIME1 is /0000 except when the calculated time exceeds 65, 535 milliseconds, in which case two words are required. For example, if TBASE is -125 and the hardware base is 8 milliseconds, label TIME1 would be equated to /0000 and TIME2 would be equated to /03E8.
ITCUS	EQU	0 or 1	If this label is equated to zero, the ITC program will not be included in the system director. If it is equated to one, the ITC program will be included in the system director.
TISHA	EQU	XXXXX	This label is equated to the number of times the programmed clock is to be updated before time-sharing is terminated.

DUMP1	EQU	0 or 1	If a zero is specified, the routine which dumps core to disk will not be included. If a one is specified, the routine will be included.
ICLL1	EQU	/XXXX	These labels are used to define out of core interrupt levels. For example, if the number of levels is 12, and the user has out of core interrupts on levels 9, 10, and 11, then ICLL1 is equated to
ICLL2	EQU	/XXXX	/00FF and ICLL2 is equated to /FFFF.

Core size of the System Director

The core size of the system director may be computed using the following formula:

System director = 1116 for MIC, PSC, EAC constants,

and work areas

+ 276 (ITC is included)

+ 95 (EAC dump is required)

+ 111 X number of interrupt

+ 3X number of queue entries

+ 2X number of process interrupts (NIL00 through NIL23)

+ 2X number of program interrupts on levels 0 through 13 (NLWS1)

+ 2X number of program interrupts on levels 14-23 (NLWS2)

+ 2X number of count routines 0-15 (NIPT1)

+ 2X number of count routines 16-31 (NIPT2)

+ 271 (TSC is included)

+ 64 (if more than 14 levels are used)

+ 7 (if more than 14 levels are used and ITC is included)

+ 6 (if more than 14 levels are used and TSC is included)

Users Responsibilities

The user is responsible for defining his configuration correctly via the EQU cards supplied and using his tailored system to do the particular job required by him.

1800 PAPERS

Hybrid Computer Simulates Steel Processes

by

E. L. Keener
Applied Research Laboratory
United States Steel Corporation

Address

Applied Research Laboratory MS44
United States Steel Corporation
Monroeville, Pennsylvania

Phone: 372-1212

Tuesday 8:30 - 10:00am

Session T-1.3

Text Pages 6

Graphics Pages 10

Hybrid Computer Simulates Steel Processes

by

E. L. Keener
Applied Research Laboratory
United States Steel Corporation

Abstract

The Applied Research Laboratory of the U. S. Steel Corporation has used a hybrid computer to simulate and to model various processes involved in iron- and steelmaking. The hybrid computer can be an important tool for engineers, operators, and mathematicians to examine ways to improve production and reduce operating costs of existing industrial processes, and to evaluate new processes. This paper demonstrates and illustrates several hybrid computer simulations of steel processes.

Introduction

What is a hybrid computer and how can it be used? These two questions have been asked many times. A hybrid computer can be considered as a universal pilot plant or as a tool to solve problems.

A hybrid computer consists of a general-purpose analog computer linked to a general-purpose digital computer. Figure 1 is a line drawing of the hybrid computer system used by U. S. Steel.

Component Operation

Without a detailed discussion of either the digital or the analog computer, it will suffice to say that the forte of the digital computer is the solution of algebraic equations and the forte of the analog computer is the solution of differential equation. With the hybrid computer, it is possible to partition the problem

and fit parts onto either the analog or the digital computer. Thus the best features of each machine may be used.

The complement of computer equipment is listed in Figure 2. The analog-to-digital converter (ADC) may read and convert up to ten analog voltages into four digital decimal numbers. The ADC is a 10-volt solid-state converter capable of 6000 conversions per second. A solid state multiplexer selects the desired analog voltage channel. Upon command all ten channels are read sequentially in less than 500 microseconds. It was felt that time skew in the 500 microseconds could be neglected for the first problems, and therefore track and store amplifiers were not provided in the interface equipment. However, this time skew can be a source of error in hybrid computing, and therefore considerable care must be exercised in programming hybrid problems. Because the converted number is normalized and has a maximum value of 0.9999, normalized scaling should be used on the analog computer. The normalized scaling makes it easier to scale between computers and eliminates a source of confusion when transferring data back and forth.

The digital-to-analog converter (DAC) converts and transfers digital data from the digital computer back to the analog computer. The DAC selects the proper connection on a voltage divider to give the desired transformation ratio across the divider. The DAC is a potentiometer and correction for the potentiometer loading is made in the digital program. The value is corrected for a 1 megohm load on the potentiometer. This restriction has not caused any problems to date. The execution of a DAC operation, including the loading calculation correction, requires 30 milliseconds. This link places some restrictions on operating speed and for some problems it is necessary to slow the analog computer.

The contact sense (CS) gives the digital computer the ability to determine the status of on/off conditions of as many as 25 switches. A CS provides a means of changing the digital computer program when an outside event occurs to change the on/off condition of one of its 25 switches.

The contact operate (LCO) enables the digital computer to control on/off conditions in the analog computer. The computer opens and closes the switches on command. Currently LCO 0, 1, 2, and 3 are used for mode control of the analog computer. This provides digital operation of the four mode-control buttons on the analog computer so that the analog mode control is automated by the digital computer. The CS and LCO provide logic elements for use in simulation studies.

The interrupt-process branch indicators (PBI) are for program interrogation of conditions in the analog computer or process area if the computer is connected to a process. Each process branch indicator reflects the on/off condition of a contact in the process area---if the contact is closed, its associated indicator is on; when the contact opens, the indicator is turned off. Thus as the contacts in the analog computer or process open and close, their respective indicators in core storage are turned off and on.

Process branch indicators can be assigned to signals in the process that are not as critical as those requiring the interrupt feature but which must be frequently interrogated by the computer program. These indicators can be individually scheduled for interrogation by the program at timed intervals.

The interrupt (INT) is a hardware interrupt (in contrast to a PBI, which is a program or software interrupt) that is immediately recognized and serviced by the computer when the associated contact closes. The clock (RTC) is also a hardware interrupt with minimum interrupt time of 0.001 hour. The PBI and INT are usable for alarms and program jumps, and the clock interrupt is useful for timing so that logging may be done at regular intervals. The INT has the highest priority of the interrupts.

The main hybrid subroutines and their execution times are listed in Figure 3.

Simulations

Ammonia Reactor

Figure 4 lists some typical hybrid computer problems that U. S. Steel has considered. The example of an N-Stage ammonia reactor, referred to earlier, is an excellent problem of a general type that may be time-divided between analog compute, transfer data, digital compute, transfer data, and return to analog compute. See Figure 5.

Figure 6 shows the general layout and flow sheet for an N-stage ammonia reactor. Reaction rate equations may be written for each reactor. The rate of reaction is a function of bed temperature, gas flow, ammonia concentration, and catalyst condition. This nonlinear differential equation is programmed on the analog computer. The mixing of the gas between each reactor bed is defined by linear algebraic equations. These equations are programmed for the digital computer. Division of the problem between computers is clear cut.

By using time sharing, only one reactor bed needs to be programmed on the analog computer. The initial conditions and parameters of the model are changed for each bed.

The digital computer in effect "leap frogs" the analog program from reactor to reactor. Upon starting, the digital computer sets the DAC corresponding to T_1, Q_1, Z and L for bed one. This operation initiates the analog circuit to simulate the first reactor bed. The digital computer then puts the analog computer into the IC mode, delays 1.5 seconds, and then puts the analog computer into the operate mode. The analog computer monitors time, which is equivalent to distance along the reactor, and when the given particle reaches the end of the bed, the analog is switched into the hold mode. At this time the digital computer reads and stores the analog voltages. The digital takes the output gas conditions from bed one and mixes them with the by-pass gas to get inlet conditions for bed two. The analog computer is re-initiated corresponding to these inlet conditions at bed two or, is prepared to solve

for bed two. The above cycle is repeated for each bed. Final conversion values are stored and compared with previous runs. Parameters are varied and the cycle repeated. Best operating may be determined from all the runs. However, with optimizing techniques, such as steepest descend or dynamic programming, the best operating conditions may be found directly.

Fluidized Bed Reactor

A 2-stage reactor is used to reduce finely divided iron ore. The ore is partially reduced with a hydrogen-rich fluidizing gas. Residence time and temperature are controlling factors in the operation and must be maintained within close limits. Both the incoming ore and the reducing gas are preheated to a temperature in excess of bed operating temperature. The sensible heat of the solid and gas streams supplies the endothermic heat required for reduction in each stage of the fluid-bed and the radiation losses from the reactor.

The mathematical model of the fluid-bed reducer is characterized by a set of simultaneous algebraic and differential equations. These equations are based on analysis of heat transfer, mass transfer, and kinetic relationships. In deriving these equations, special effort is directed toward the utilization of physical parameters that are readily measured and controlled. The simulation is accomplished by programming the equations on the hybrid computer.

Evaluation of the mathematical model, conducted by comparing computer simulation data with experimental pilot-plant data, was based on available steady-state information; the simulation results provide good correlation with the pilot-plant data. Ultimately the model will be used to predict the dynamic and steady-state response characteristics of a commercial, fluidized bed reactor. It is anticipated that it will provide a means for specifying process parameters, hardware-design parameters, operating guide lines, and instrumentation requirements for future commercial fluidized bed facilities.

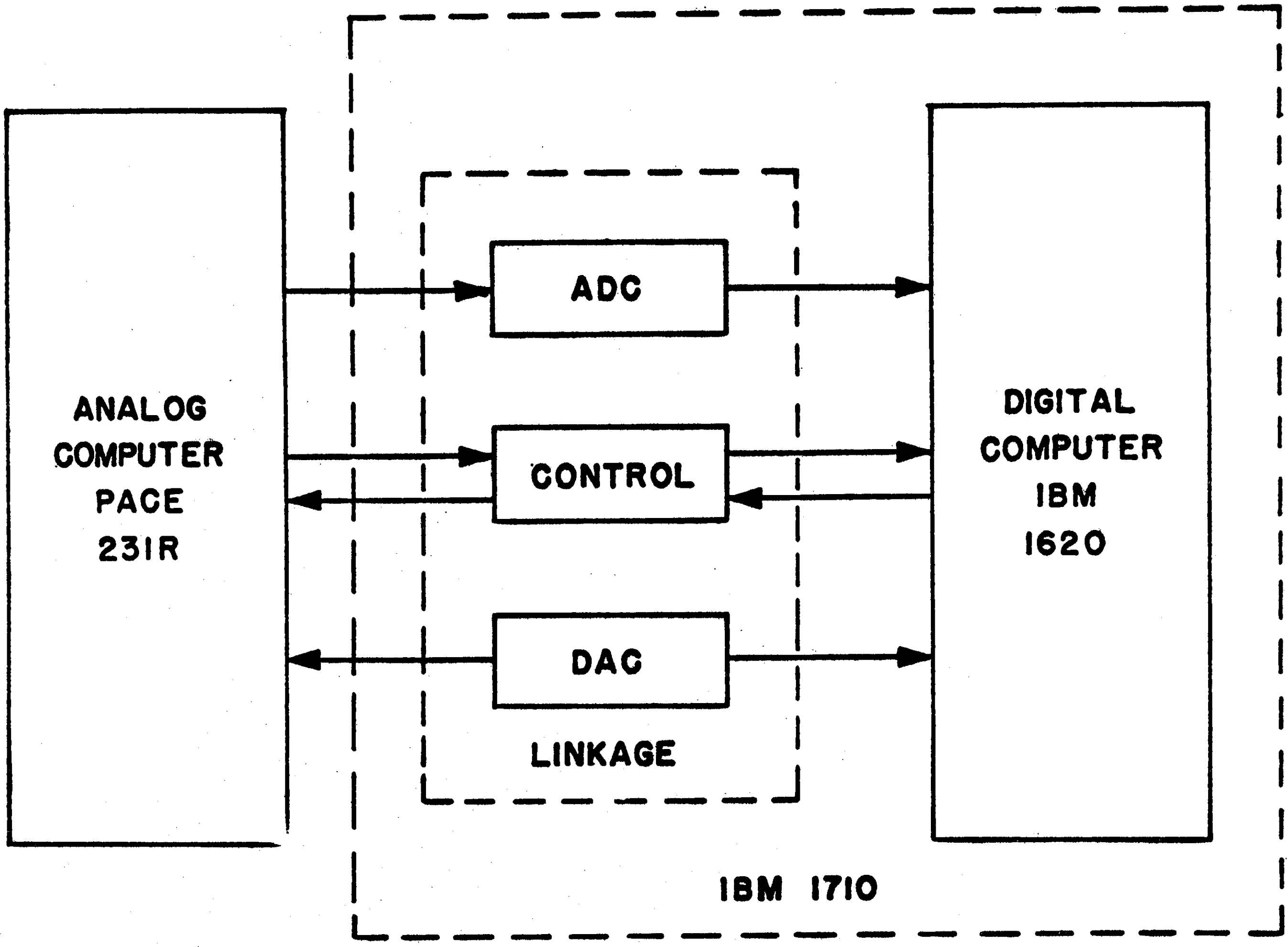
Rolling Model

A mathematical model of the cold-rolling process was developed and programmed for the analog computer. See Figure 7. Later, when the elastic entry and elastic exit zones were added to the model, the up-dated model was reprogrammed for the hybrid computer. With the rolling model, studies of problems associated with the cold-reduction of steel strip can be conducted using the hybrid computer.

The rolling process is defined by nonlinear, differential equations with split boundaries. The differential equations are solved by the analog computer. Boundary conditions are checked, compared and reset by the digital computer; thus rapid iterations to the convergent solution are possible with the hybrid computer. After the correct solution is found, the time scale of the problem is changed and graphical records of the solution are made on an X-Y recorder. The digital computer then steps to the next set of parameters and proceeds to the next solution. Approximately four hours of hybrid computer time are equivalent to 200 hours of analog time.

Magnetic Taconite Processing

Taconite processing consists of four main steps: 1) mining, 2) crushing, 3) concentrating, and 4) agglomerating. See Figure 8. The simulation on the hybrid computer is for the crushing and concentrating. Figures 9 and 10 are the crusher and concentrator flow sheets. The simulation is divided so that the crusher is simulated by the digital computer and the concentrator is programmed on the analog computer, the combination operating together to form a hybrid simulation. The digital simulation of the crusher determines the ore characteristics leaving the crushing for various train schedules. Effects on the concentrator of any changes in ore characteristics are readily determined from the simulation. Sensitivity and control study are possible with the simulation. Eventually the digital computer may be programmed to check out control algorithms for the actual plant.



U.S. STEEL HYBRID COMPUTER SYSTEM

FIGURE 1

ANALOG

200	AMPLIFIERS
300	POTENTIOMETERS
40	SERVO MULTIPLIERS
9	QUARTER-SQUARE MULTIPLIERS
18	LIMITERS
20	COMPARATORS
10	DIGITAL DIODE FUNCTION GENERATORS
1	I/O DESK (ADIOS)

DIGITAL

1620 MODE 2

60 K

DISK

CARD READER

CARD PUNCH

TYPEWRITER

LINKAGE

10	ADC	(ANALOG-DIGITAL CONVERTERS)
10	DAC	(DIGITAL-ANALOG CONVERTERS)
25	CS	(CONTACT SENSE)
28	LCO	(LATCHING CONTACT OPERATE)
4	PBI	(PROCESS BRANCH INDICATORS)
4	INT	(HARDWARE INTERRUPTS)
1	CLOCK	(TIMED INTERRUPT)

COMPLEMENT OF EQUIPMENT

FIGURE 2

ROUTINES	OPERATING TIME
CALL HYRAI	15 MILLISECONDS
IF HYICS̄	15 MILLISECONDS
CALL HYNRPT	15 MILLISECONDS
CALL HYS̄TPT	30 MILLISECONDS/CHANNEL
CALL HYS̄LCØ	15 MILLISECONDS/CHANNEL
CALL HYS̄RTC	
CALL HYRTC	
CALL HYPTS̄T	
CALL HYIC	
CALL HYHØLD	
CALL HYØPRT	

HYBRID FORTRAN CALLS

FIGURE 3

1. AMMONIA REACTOR

2. HEAT FLOW

   A. BLAST FURNACE STOVE

   B. INGOT COOLING & SOLIDIFICATION

3. MODEL - ROLLING MILL

4. FLUIDIZED BED REACTOR

5. PROCESS OR SIGNAL IDENTIFICATION

   (FOURIER TRANSFORM)

6. TANK TRUCK SIMULATION

   (ROAD PROFILE GENERATOR)

7. CRUSHING PLANT SIMULATION

8. TRANSPORT DELAY SIMULATION

9. FUNCTION GENERATION

HYBRID PROBLEMS

FIGURE 4

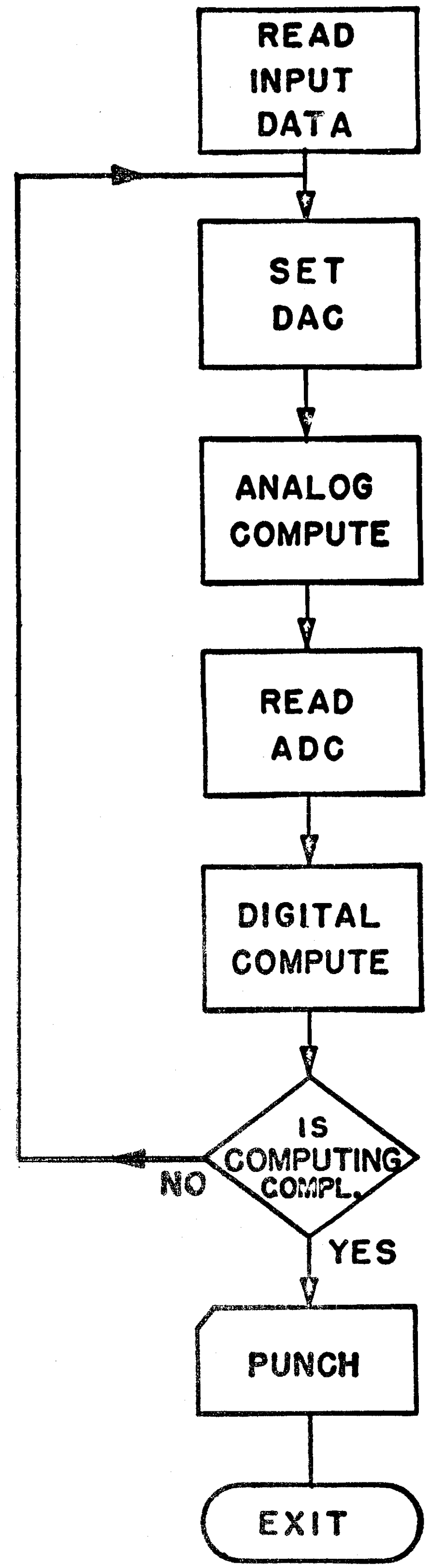
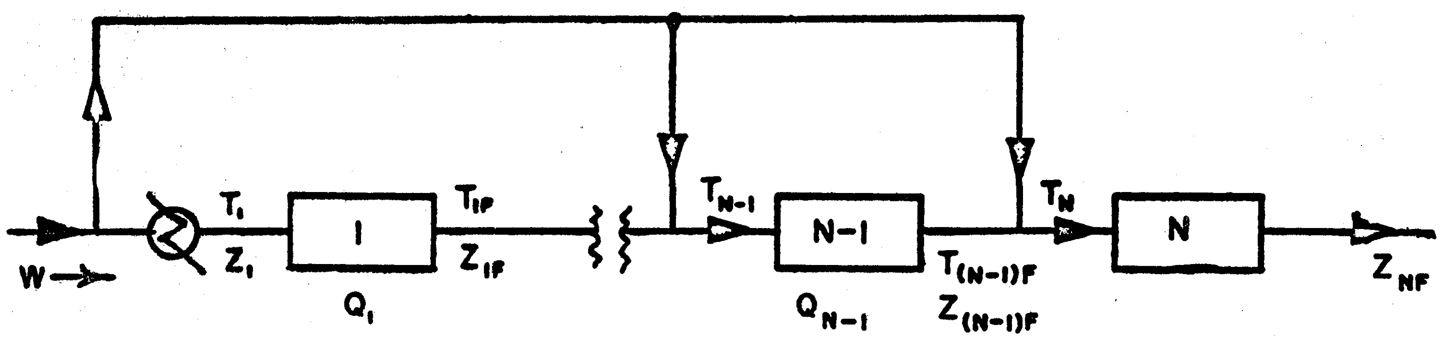

READ INPUT DATA
SET DAC
ANALOG COMPUTE
READ ADC
DIGITAL COMPUTE
IS COMPUTING COMPL.
NO
YES
PUNCH
EXIT

FIGURE 5

AMMONIA REACTOR



W - POUNDS GAS FLOW / HR

T - TEMPERATURE

Q - % GAS FLOW

Z - % AMMONIA

FIND OPERATING CONDITIONS FOR MAXIMUM Z_{NF}

FIGURE 6

ROLLING MILL

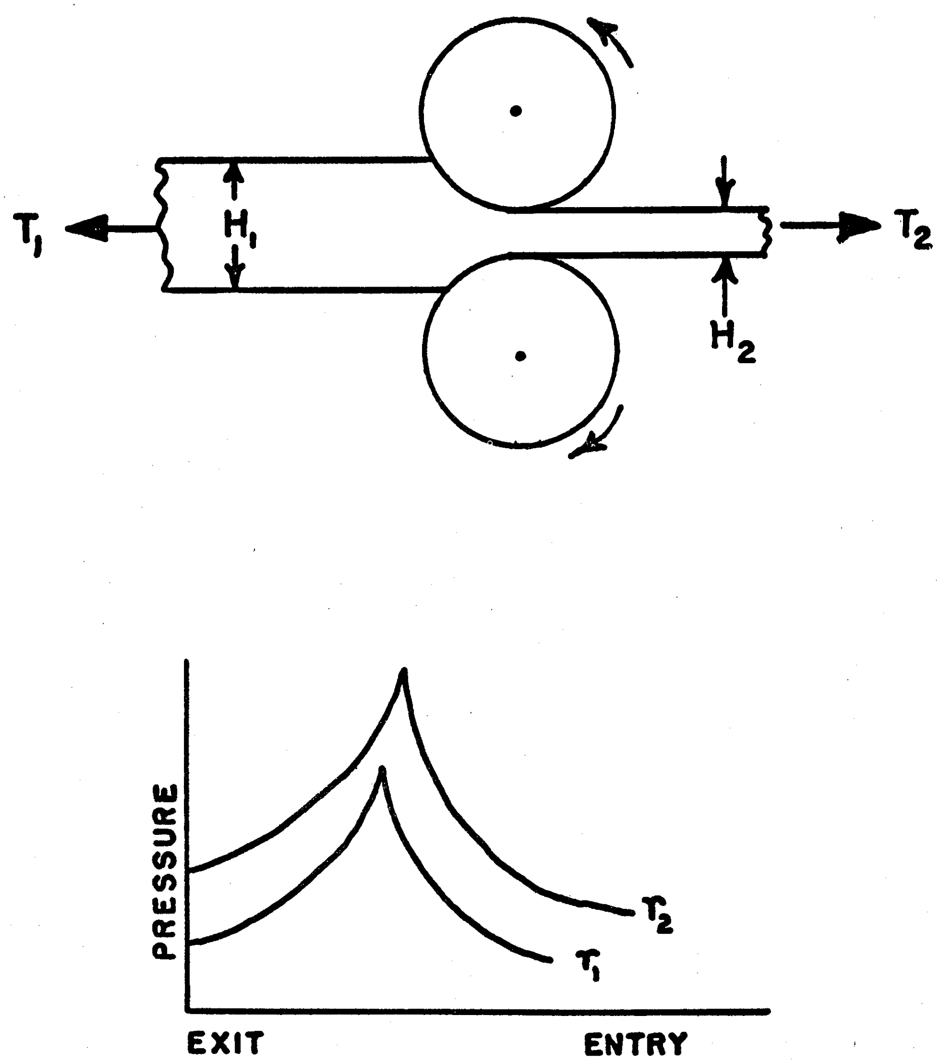


FIGURE 7

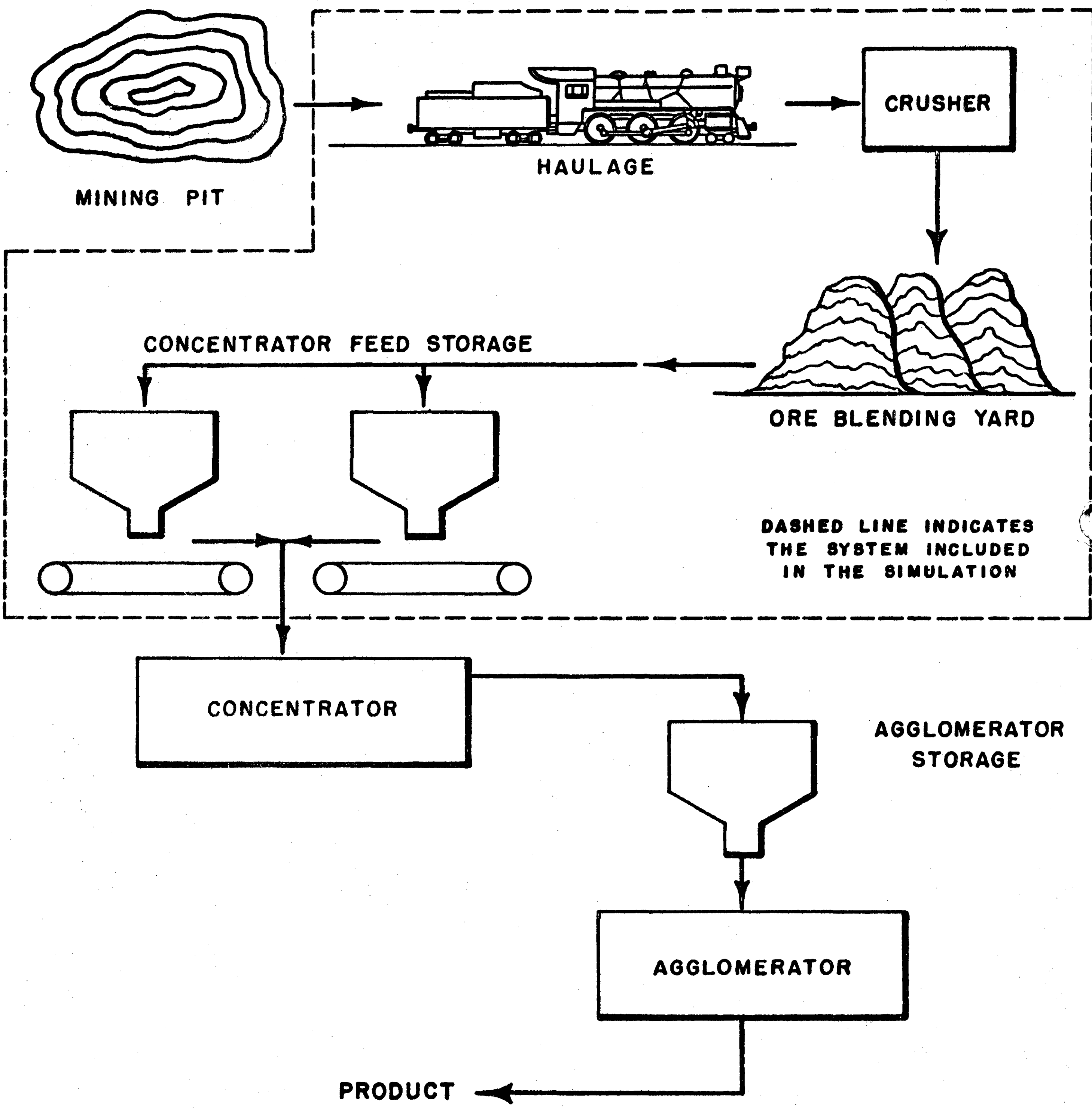


FIG. 8 TACONITE PROCESSING

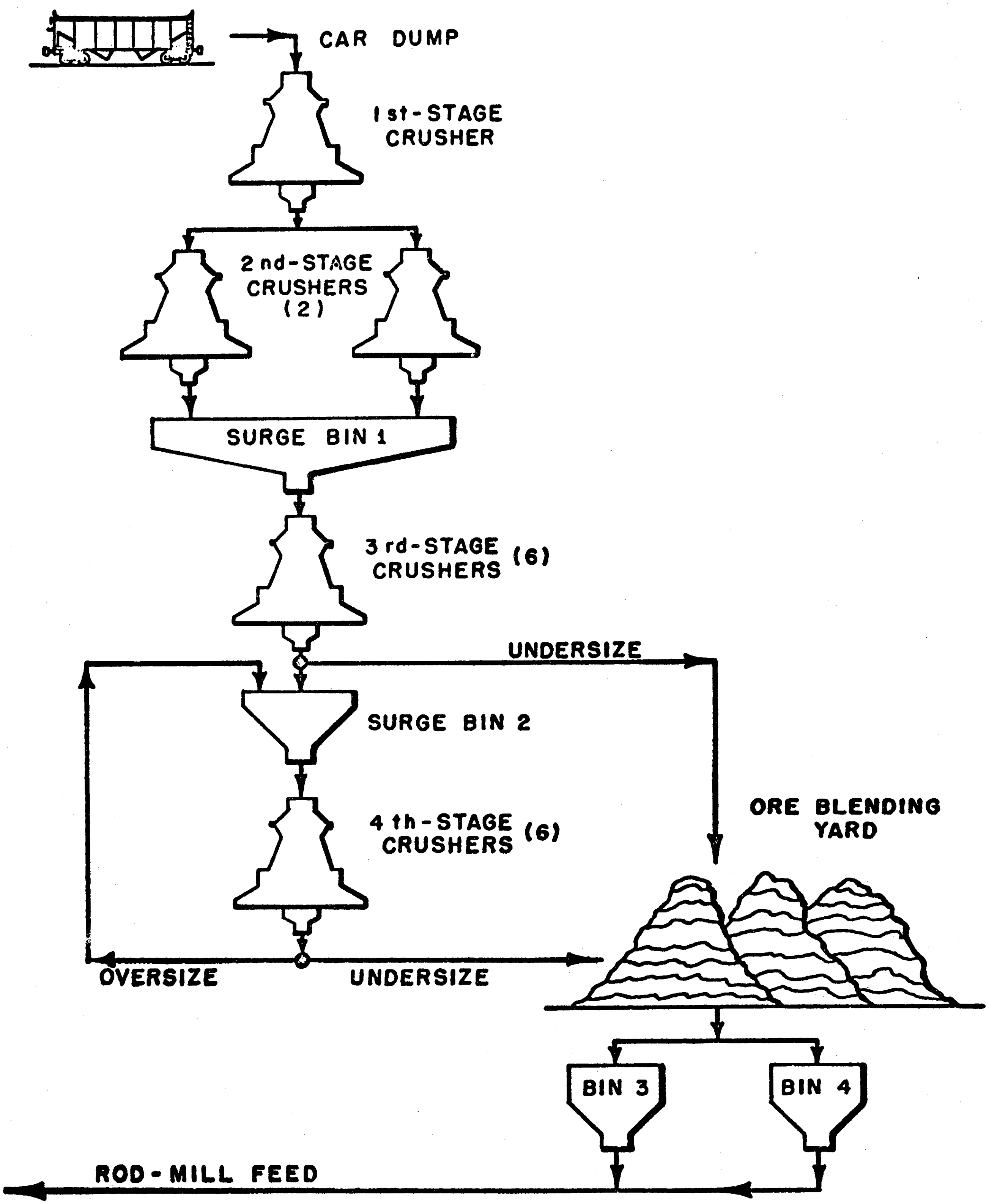
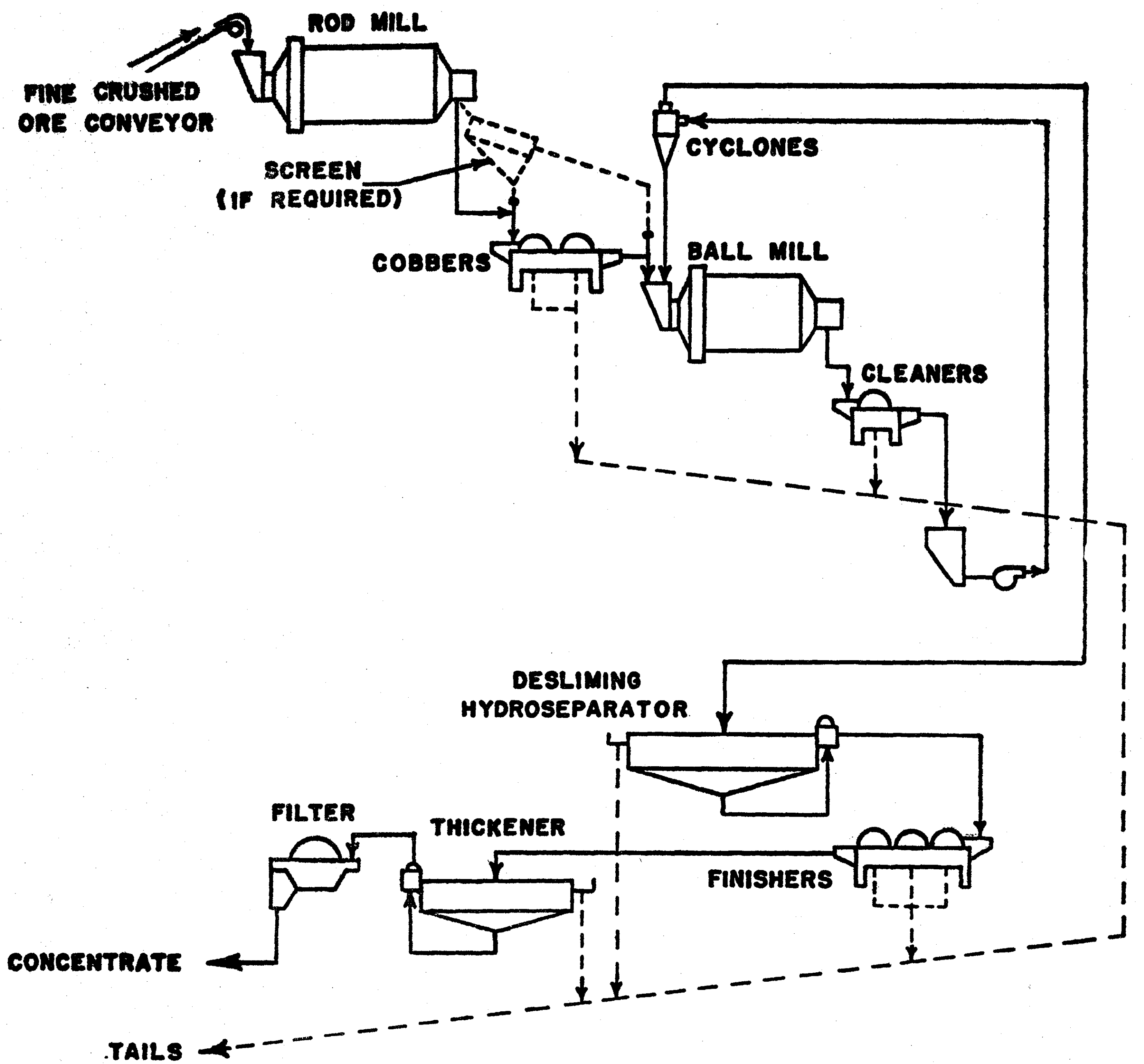


FIG. 9 CRUSHER



TYPICAL CONCENTRATOR FLOW SHEET

FIGURE 10

The present 1130 Commercial Subroutine Package had its origins in the FORCOM package which was created for the 1620 system at an IBM branch office in Detroit, Michigan, in the summer of 1961. The original 1620 FORCOM package consisted of subroutines which provided a basic character handling capability for use with 1620 FORTRAN. In subsequent years the package was expanded to operate with later versions of the FORTRAN compiler and to operate under control of the 1620 Disk Monitor system.

In the fall of 1965, it was decided to adopt as many as possible of the 1620 routines for use on the 1130 to provide a basic commercial data processing capability. This work resulted in the announcement of FORCOM-1130, a type III program which became part of the 1130 program library in March of 1966. This program was subsequently upgraded to a type II programming announcement and re-named "The 1130 Commercial Subroutine Package" in August of 1966. The type III 1130 FORCOM package was subsequently withdrawn on October 18, at which time 390 users had requested this package. The 1130 Commercial Subroutine Package had already accumulated 198 registered users by the first week in October.

The present 1130 Commercial Subroutine Package includes eight subroutines to provide alphabetic compare, editing, zone punch manipulation, stacker select and related operations useful in commercial applications. The present package is independent of input/output considerations, and all of the subroutines except the stacker select subroutine are written in FORTRAN to permit easy modification. The eight subroutines currently in the package are:

MOVE	To move a variable length, alphameric data field
EDIT	To edit a data field for subsequent printing
GET	To extract and float a data field from an input area
PUT	To unfloat and place a data field in an output area
NCOMP	To compare two variable length, alphameric data fields and branch on high, low or equal
NZONE	To test or to modify a zone punch on a single card column
FILL	To fill a variable length data field with a single specified character
STACK	To cause the next card to be selected to the alternate stacker on the 1442 card read punch.

1130 PROGRAMS

1130 Commercial Subroutine Package

R. K. Louden
IBM
Monterey and Cottle Roads
San Jose, California

Telephone: 227-7100, Ext. 2075

November 29, 1966

The 1130 Commercial Subroutine Package is designed to run upon an IBM 1130 system with 8192 words of core storage, with card input and output and with or without disk storage.

All of the character manipulation performed in this package is based upon the concept that the bit pattern used to represent a single character in A1 format is the same bit pattern used to represent some specific binary integer. This binary integer, written as a decimal number, can be considered to be the decimal equivalent of the character code in A1 format. For example, the decimal equivalent of the letter A is - 16064, the decimal equivalent of the digit 0 is -4032, and the decimal equivalent of a dollar sign is 23360. The concept of a decimal equivalent for each character makes it possible to manipulate characters as though they were integer variables within FORTRAN programs.

One facility which is required in commercial applications is the ability to read unformatted records. FORTRAN does not provide this ability; in FORTRAN it is necessary to specify a format statement which requires a knowledge of the format of the record to be read. In the commercial subroutine package, this problem is avoided by reading all input records in A1 format. For example, an 80 column card would ordinarily be read by a format of 80A1. Each character from each card column occupies a separate core storage word, and FORTRAN subroutines using the decimal equivalents of the characters can be used to extract data fields from the card and convert the data fields to floating point numbers as required for floating point calculations. This technique makes it possible to read a card, test one or more card columns to determine the format of the card, and only then proceed to pick up those data fields from the card which are required for subsequent processing. The GET subroutine is used to pick up and float data fields from character strings in A1 format; the PUT subroutine is used to unfloat and place calculated results inside character strings.

The concept of a decimal equivalent for any character in A1 format makes it easy to write a subroutine to compare character strings and branch on some predetermined collating sequence. As it turns out, the decimal equivalent for the letter A is the lowest number (- 16064) of any decimal equivalent in the 1130 character set. Since the decimal equivalent codes are in a sequence, a subroutine containing a FORTRAN IF statement can be used to compare one character string against another, one character at a time, and branch if a high or low condition occurs. This is the function of the NCOMP subroutine.

Another facility needed for commercial applications is the ability to both sense and manipulate zone punches. It has long been a common practice in commercial applications to identify negative data fields by an 11 overpunch on the low order digit; this convention is followed in the commercial subroutine package. The manipulation of zone punches is based upon the fact that the zone of a character may be altered without affecting the other punches which constitute the character by simply adding a multiple of 4096 to the decimal equivalent code of the character. For example, the addition of 4096 to the decimal equivalent code for the letter A produces the decimal equivalent code for the letter J. The letter A, of course, is represented on a punched card by a 12 zoned 1 punch, and the letter J is represented by an 11 zoned 1 punch. Through similar manipulations the NZONE subroutine is able to identify which of five classes of zone punches appear on any character and to change the zone of any character to any specified zoning.

Certain editing functions are also required to present commercial results in an acceptable format upon the printed page. For example, it may be desirable to suppress leading zeros in calculated results. If checks are to be written, it is frequently desirable to replace any leading zeros with asterisks to prevent additional high order digits from being written upon the check after the check is printed. This technique is known as asterisk check protection. A similar effect may be obtained through the use of a floating dollar sign which is positioned immediately to the left of the high order non-zero digit in the result field. It may also be desirable to merge strings of calculated numbers with arbitrary strings of characters, perhaps to produce social security numbers, alphameric inventory part numbers and similar results. All of these capabilities are provided by the EDIT subroutine.

The EDIT subroutine uses a mask field which contains special characters to be merged in with the field to be edited. The special characters are used to indicate the suppression of leading zeros, asterisk check protection, floating dollar sign and other special characters to be merged with the data field. The EDIT subroutine also makes it possible to indicate a negative field by a minus sign to the right of the low order digit, or to indicate a credit field by the letters CR to the right of the low order digit. Although the logic of the EDIT subroutine is complex, the subroutine operates quite simply by identifying the decimal equivalents of the characters involved and manipulating the characters accordingly.

Two minor subroutines, MOVE and FILL, are also provided to permit the easy transmission of character strings from one array to another. Finally, a stacker select subroutine, STACK, is provided to select a card into the alternate stacker on the 1442 card read punch. The STACK subroutine is written in assembly language because it refers to a hardware function which cannot be described in FORTRAN.

Character strings to be manipulated by these subroutines are always stored in memory as singly dimensioned arrays in A1 format. Control statements specifying one word integers and extended precision should be used with these programs to minimize the amount of memory devoted to the character arrays, while at the same time maximizing the precision to be obtained from the floating point calculations.

A further convention in zone punch manipulation is required to conform to the 1130 character set. Since a 12 zoned 0 is not a valid character in the 1130 character set, this combination is converted to a zero by the NZONE subroutine. Similarly, an 11 zoned 0 is replace by this subroutine with the character code for a minus sign.

It seems surprising to many people that subroutines can be written in FORTRAN to create floating point numbers from character strings and vice versa, as is done by the GET and PUT subroutines in the commercial subroutine package. These subroutines are made possible by the fact that there exists a simple equation relating each binary digit to its decimal equivalent code in A1 format. The equation is:

Binary digit = (decimal equivalent code + 4032)/256

The GET subroutine uses the above equation to convert decimal equivalent codes for integers into binary integers, and then collects these binary integers times suitable powers of ten into floating point variables. The PUT subroutine essentially performs the same operation in reverse.

The 1130 commercial subroutine package does not pretend to provide a complete commercial programming language for the 1130. Many requests have been received to expand the capabilities of this package to provide additional functions. These requests have fallen primarily into the categories of extended precision arithmetic, overlapped input/output and the ability to manipulate packed A2 format as well as the unpacked A1 format currently utilized by the package. The feasibility of providing these additional capabilities to the package is currently being evaluated, but there is no commitment at this time to provide these extensions.

In the area of extended precision arithmetic, a study is currently underway to determine the practicality of providing variable length add, subtract, multiply and divide subroutines to operate directly upon strings of decimal integers in A1 format. These subroutines would theoretically possess the ability to perform arithmetic computations on fields of any specified lengths while being completely free from floating point round-off errors which remain a significant problem in commercial operations involving large dollar amounts which must be accurate to the penny. An obvious problem with this approach lies in the long execution

times which may result from multiply and divide operations upon long character strings.

To provide overlapped input/output operation, we are currently investigating the feasibility of calling the overlapped input/output subroutines available for 1130 assembly language programs directly from FORTRAN programs. If feasible, this capability would make available overlapped input and output operations at the FORTRAN level.

The ability to manipulate packed characters in A2 format could conceivably be provided by subroutines to convert between A1 and A2 format at high speed. The ability to store data in A2 format is desirable since it reduces considerably the amount of disk storage required to maintain data files in, for example, inventory control applications.

Although work to determine the feasibility of extending the power of the 1130 commercial subroutine package is proceeding along the lines described above, it must be emphasized that this work represents a feasibility study on the part of the author and does not, at this time, represent any commitment whatsoever upon the part of the IBM Corporation to provide these extended capabilities.

A detailed description of the calling sequences and functions of the present 1130 commercial subroutine package may be obtained from the 1130 Commercial Subroutine Package Program Reference Manual (IBM Publication No. 1130-SE-25X).

R. K. Louden

University Education Papers
Computer Requirements for the Undergraduate College

Robert C. Bushnell
Oberlin College
Oberlin, Ohio 44074

Tuesday, Nov. 29, 1966 - 8:30 A.M.
Five Pages
Session T-1.6

Phone: (216) 774-1221, ext. 3160

COMPUTER REQUIREMENTS FOR THE UNDERGRADUATE COLLEGE

Robert C. Bushnell, Oberlin College

There is little doubt that practice and experience in the use of digital computers and a knowledge of digital computer applications and methods in his field of major interest is a necessity for the undergraduate student in engineering or technology and physical or social science. Given the trend of present research it will not be many years before the same statement can be made of students in languages, history, art and music. Therefore the educational institution which would furnish adequate preparation to its students must necessarily include computation in its program. Most educators recognize this but not being computer men, do not understand that all computers are not equal, and that _having_ one is not synonymous with _using_ one.

In particular I believe there is insufficient awareness among educators, and perhaps even among some computer people, of the advantages, in an actual use sense, of more-than-minimum core, line printers, disk storage and monitor or operating systems. This paper will report the experience of Oberlin College in two academic years, one in which we possessed only a 20K 1620 and a 1622, and the second in which this equipment was augmented by a 1311, a 1443 and a 1623 with additional 20 K storage.

On December 1, 1965 Oberlin began use of the basic 1620 computer with card read-punch. Series of lectures on both SPS and Fortran had been presented and a considerable number of library programs obtained. Use built quickly: five hours the first week, 7 the next, 11 the third, 14 the next (skipping Christmas vacation), 17 the fifth, then 25, 29 hours per week. Then the build-up stopped, hours per week for subsequent weeks are 17, 23, 11, 19, 25, 15, 17, etc.; a plateau clearly had been reached. An average over the 21 weeks of the academic year for which time use was recorded shows that the computer was used 19 hours per week. The cost of the computer for that five month period was, (@ $1,868) $9,340, or the cost per hour $23.35.

On September 7, 1965, the college added the additional equipment mentioned: additional core storage, 1443 and 1311. During that academic year, from September 7 through June 2, 3,623 hours were logged. If we deduct immediately the 785 hours logged against the computer center itsel , which includes all the hands-on experience for the lecture courses offered by the center itself, and if we deduct the 167 hours logged for non-academic purposes, we still are left with 2,671 hours run on academic application oriented work, or approximately 297 hours per month of academically oriented work. 1620 machine rental for this period was $4,560 per month, yielding a cost of $15.35 per productive hour. I emphasize that this conservative calculation has charged no burden to any non-academic purpose or administrative purpose. In other words, adding equipment had

the effect of reducing the per hour charge. In marginal, or incremental terms, the change was even more striking. For an additional outlay of $2,225 per month, Oberlin College obtained an academic user increase of 215.3 hours per month at a marginal or incremental cost of only $10.35 per hour.

This citing of incremental cost is not pedantry but the kernel of the problem in computing in the private undergraduate college and a cause of deep concern to some of us who believe that the private undergraduate college has a unique and necessary place in the total pattern of education in the United States. For low average computer costs per hour of time used can only be attained for a high total cost, a cost which is beyond the capability of most such institutions. Oberlin is probably in the best financial position of any of these institutions and yet it is only with difficulty that we have been able to finance such a program. In such circumstances it is both natural and proper that we turn to governmental sources on the grounds that investment in higher education has a demonstrable effect on the level of future national output. Unfortunately, there are few programs applicable to such institutions. In 1965, Oberlin applied to the National Science Foundation for funds under the Undergraduate Instructional Scientific Equipment program and was fortunate enough to receive $40,000, nevertheless a fraction of what we requested. When we reapplied in 1966 for further funds, our request was disallowed. Privately, we were informed that $40,000 was much larger than any similar grants, and that total funds were limited. However, we somehow gained the impression that many members of the panel felt that our proposal was too ambitious and not necessary for an institution involved primarily in undergraduate instruction. It is one of the purposes of this paper to try to correct that attitude.

Before doing so, however, I would like to point out the differences in situation between the universities and the colleges and try to maintain that it would be proper and economically effective for NSF to allocate more of its total computer funds to undergraduate use. Presently, NSF will allocate millions of dollars to major institutions for computing, but as we have found, only $40,000 to an undergraduate college. As a case in point, Princeton University has an accelerator and a plasma physics lab and a graduate school of engineering. Research is intensive and heavily supported by NASA, AEA, DOD and other governmental agencies. Computing is a requirement of such work; total usage is high; hourly cost of computing is consequently low, and the burden on the unversity for the support of undergraduate computing on the 7094, 7044 and soon, the 360/67 is nominal and bearable. Princeton undergraduates emerge from their undergraduate education with good experience in computing. Oberlin, on the other hand, has almost no such contracts, the college must bear the entire cost of providing computing in the academic program except, as it

is directly aided by NSF or other grants. The undergraduate population of the two institutions are approximately equal, yet Oberlin annually produces more entrants to graduate schools than does Princeton, and in historical terms only 10 institutions of any nature or size whatsoever have in absolute total terms produced more graduates who later have attained the Ph.D. degree in science than has Oberlin.

Oberlin trustees do not wish this situation to change. We believe that should we not invest heavily in computing facilities this situation will change. In point of fact, the awareness that the lack of sufficient computing facilities was impairing our ability to attract faculty and students alike in the physical sciences was the greatest single incentive in our present program of computing. In five years, the same will definitely be true for the social sciences and I don't think I am going too far out on a limb to say that five years after that the same also will be true of philological and historical studies. (The Chairman of the Theory Department in the Oberlin Conservatory of Music believes the same will be true in his department in two years.) Oberlin fortunately will have sufficient computing facilities, but what about other not so fortunate institutions of equal intellectual merit? It certainly seems to me that the case must be made for governmental support for adequate as distinct from minimal computing facilities for undergraduate colleges. Probably most institutions can scrape up $100,000 or so. Another $100,000 to these institutions from NSF or similar sources could work toward an adequate facility. For only a million dollars 10 of these institutions each with at least an enrollment of 2,500 students could be given proper computational training and support in their undergraduate curricula. Surely this is as important a use of seed money as a grant of a million to a major institution.

What then do we mean by an adequate facility for a computing center? The answer can only be given in terms of user service. If the library had only one chained copy of each book and students had to wait in line to use that copy, though that book be the most valuable in the world students would find other ways to accomplish their purpose. Too many computing centers are in fact like that library which would never be. I would define "adequate user service" as service that would allow each user to obtain the output of his compilation or debugging run in under 10 minutes. I would define "an adequate computing facility" as one which provided such user with work space and technical advice such that, under normal circumstances, he is able to work persistently and continuously on his program. For as we in computing know, programming is 20% inspiration and 80% debugging. You and I probably have our percentages down to about 50-50; that's what makes us expert, but to the tyro, the most alarming thing is that the automatic electronic computer seems to do maddeningly little automatically. Thus, if we are to initiate the casual user, we must make the experience as pleasant as possible, lest we do him the disservice of making him reject computing.

After all, most students should not be interested in the computer per se but in the computer as a tool, and a tool to be useful must be usable. If this seems utopian, and I hope it does not, let me point out that this is the way things will be in research and industry inside of five years to the extent that it is not already, and that as educators we must be a great deal more interested in the future with which our students will function than in the present in which we exist. Second, more practically, it is not utopian because we have done it at Oberlin.

What does it take to make an adequate computer center -- I would identify three ingredients: (1) adequate hardware, (2) fast turn-around software, and (3) user interest. For hardware, we defined adequate as a 40K 1620 with 1622, 1311, and 1443. The 1311 is required for the operation of Monitor I, a good job-stream processor. The 1443 is required to provide unbound machine-man communications. Finally, if Fortran is to be the principal user language because of its ease of use, then one must allow for its sloppy use of storage by providing storage sufficient that the user does not have to concern himself continuously with running out of core.

For software, three needs were seen. First we determined that the Fortran II compiler was too slow to provide the required turn-around. Therefore, we implemented the Forgo compiler for 1620 disk as developed by the University of Wisconsin (1620.2.0.043) and modified by us for use with the 1443 printer. This compiler allows us to process the typical learner's job in about 20 seconds. Our second major software implementation was the provision for an interrupt system. Our goal here was simultaneously to provide service both to users with long jobs and to tyro users (or experienced user writing a new program) who needed repeated submissions of a compilation or short test. What we developed was a sense switch 2 test in the monitor printer output routine. Every user whose program may run over 10 minutes is required to use a 200 core position subprogram in his program which furnishes the proper linkages so that when switch two is turned up, his program, all core, all work cylinders on disk, etc. is stored away elsewhere on the disk and the next, supposedly short job, is read in. At the conclusion of the short jobs, execution of a standard job called RESTORE resumes operation of the interrupted program. By such means, we retain our ability to service short jobs on call while still devoting all otherwise unused time to the processing of long jobs.

Our third major software project was to provide a printer plotting routine with the same parameters as the UMPLOT routine for the 7094. This is a self scaling set of programs which makes user plotting in Fortran foolishly simple.

Last and most important, the third ingredient, users, were dumped at our door by the Mathematics Department, which made Fortran programming a two-week item in the Introductory Calculus menu.

I would be lying and belying hard work if I asserted that from that point on the computer ran itself, but I can say that with these facilities we had all we could do just to keep up with the users -- there was no need to sell computing, just to service the demand. Twenty students working a total of 170 hours per week and the director were the total staff in that first year of Monitor operation. In that space these students found time to operate, answer user questions, and still provide over 100 canned programs and disk stored subprograms complete with published user instructions.

To those educators who assert that a computer is a computer and that their institution has one, the moral of the foregoing should be obvious. All computers are not alike; a machine capable of receiving the software support necessary to insure easy user access and quick turn-around is the sine qua non of a successful computer center and the justification of a computer as a tool, not as a toy.

COMMON

New Orleans, Louisiana

University Education Papers

Library Circulation

Guy George Jr.
University of Southwestern Louisiana

Box 382 U.S.L.
Lafayette, Louisiana 70501
CE47321 Ext 274

T1.6
Tuesday
November 29, 1966
8:30-10:00 A.M.

Text 4
Graphîcs 2

LIBCIR

A Library Circulation Program

The University of Southwestern Louisiana's library installed an IBM 357 to accelerate the process of circulation. In this process, each borrower has a plastic ID card with a borrower number punched on it. Each book has a master card with information such as call number etc. A borrower selects his book from open shelves and presents the book with his ID card to the circulation desk. The ID card and the master card are placed in the IBM 357 which causes a charge card and a return card to be punched on an IBM 026. The library retains the charge card while the return card and master card are placed in the book. When a book is returned, the return card is removed and the book is placed on the shelves.

The processing of these charge cards and return cards can solve three basic problems of library circulation:

1. Update the outstanding file
2. Purge the outstanding file
3. Check file for overdue books

In solving these problems, the library can be supplied with an accurate list of books which are outstanding on a daily basis or more frequently if necessary. Also, overdue notices can be given to the library management to be handled as they desire.

LIBCIR is an interpretive system written for the IBM 1620 to solve the three basic problems of library circulation. The hardware must include the following: 40K memory, automatic divide, indirect addressing, card read, card punch, disc drive, and a on-line printer. One disc pack is necessary for maintaining the file and for storing the programs which make up the interpretive system.

Before discussing the programs which make up the interpretive system, it is necessary to explain the method of storage for charge card images. The charge card and return card are identical, and contain a combination of the date, the information from the ID card, and the information from the master card. These cards have the following format:

col 1-6	Due Date
col 7	Blank
col 8-13	Item
col 14-25	Class
col 26-33	Cutter
col 34-35	Year
col 36-38	Volume
col 39-40	Copy
col 41-43	Purchase Number
col 44-53	Borrower Number

The files for listing purposes are to be in collating sequence for columns 14 to 33 which makes up the call number for the book. These columns contain a mixture of alpha, numeric, and special characters. Each image is assigned one disc sector which includes the information from the charge card in a packed form along with a counter for the number of times the book has been overdue. This requires a total of 85 digits per sector. The other positions are left vaccant for possible additional information.

The disc is divided into two distinct portions. Cylinders 0 to 97 are reserved for the charge card file. Cylinders 98 and 99 are used for storing the programs, and a table of reference information about the contents of cylinders 0 to 97. The system is designed with expansion in mind. There is ample room on cylinders 98 and 99 to include many more routines.

The book images are maintained in the desired order at all times. A cylinder will be utilized at a minimum 25% of its storage capacity. The total number of books is divided by 50 and if this does not require more than 98 cylinders, this minimum capacity is utilized. If, however, more than 98 cylinders are required, the efficiency of storage is increased and the number of images per cylinder is redetermined. There is a maximum storage allowed of 190 images per cylinder. With the idea of non maximum utilization of storage, the file can be maintained in order at all times. As long as there are vaccant sectors within a cylinder, another charge card can simply be inserted in the file in the correct location. This method was chosen to avoid having to preform a sort other than for the first day of operation.

With the file open for ready insertion of additional books or easy purging, certain amounts of directional information must be supplied. The first sector of each cylinder 0 to 97 (if being utilized at all) contains the following information:

1. The last sector in the cylinder being used
2. The double digit alpha code fot the first 20 locations of the call number for the first image stored on the cylinder
3. The double digit alpha code for the first 20 locations of the call number for the last image stored on the cylinder

To further index the images stored in cylinders 0 to 97, a reference table is maintained by each program and stored in the first 8 sectors of cylinder 98. This table consists of 100 entries. Entries 0 to 97 will be composed of two fields. The first is a six digit field which can have one of three values:

1. $\bar{9}99999$ if the cylinder was not required for the storage of the images
2. $\bar{0}00000$ if the cylinder was used but all of its images have been deleted through processing
3. $\bar{X}XXXXX$ where the X's represent the first 6 digits of the double digit coding for the call number of the first image on a cylinder

The second field is two digits and ranges from $\bar{0}0$ to $\bar{9}7$ and indicates the cylinder number for the six digit field mentioned above. With this table the programs can quickly locate the cylinder for an entry. Entry 98 is a field of 8 nines and acts as a buffer to initial searches. Entry 99 is a five digit field which is the current number of images being stored.

With the method of storing images in mind, the interpretive system will be discussed briefly. The heart of any interpretive system will be a supervisory program to direct the calling of the correct stored programs. LIBCIR's supervisor accepts the following control cards:

$EDIT CHARGE	(must include the date)
$EDIT DISCHARGE	
$LOAD INITIAL	
$CHARGE	
$DISCHARGE	(must include the date)
$LIST	(must include the date)
$DISTRIBUTE	
$OVERDUE	(must include the date)
$END OF JOB	

If the date is necessary, it is punched in columns 17 to 24 as XX-XX-XX (month, day, and year). The supervisory program has a list of acceptable control cards and the disc control fields which indicate where the desired routine can be found.

EDIT CHARGE This routine will check all data which is about to be added to the file to see if it contains the correct date and the correct types of information.

EDIT DISCHARGE All discharge cards are verified for the correct types of information. The dates are not checked.

LOAD INITIAL This program is designed to set up the original layout of the file storage. The charge cards for the first day are read and written onto disc. A tag is built from the first 20 digits of the call number and a sector address for the stored image. This tag is sorted into order before the next card is processed. A binary search and insertion sort is employed to conserve time. With the sorted tag file, 50 images are written per cylinder; the first sector of each cylinder is created; and the table entry is made for each cylinder. This prepares the file for all future processing.

CHARGE The charge cards are processed for a regular day. Once a card is read, the table is scanned to determine which cylinder the image might be placed on. The first sector of this cylinder is checked to see if it can in fact be placed on this cylinder. If not it checks the previous cylinders until the image can be stored. Once the correct cylinder is located, it is read and a scan is made for the correct location for inserting an image. An opening is made by writing the entries before the insertion on disc, then the new entry is written, and then the entries following the insertion are written onto disc. The first sector of the cylinder is updated, and the table entry is updated.

DISCHARGE This works in the same fashion as the CHARGE, only the cards read are return cards. When the entry is located, the entry is deleted through one transmitt record. The first sector is updated as well as the table entry for the cylinder involved. This routine will automatically call for the LIST routine to produce the list of books left after the purge. It will also automatically cause the books to be redistributed evenly for all cylinders. This is done by a call placed for the DISTRIBUTE routine.

LIST If a copy of the file is desired, this program will produce a listing of the outstabding file. This is the routine which is called by DISCHARGE to give a list for the day. Since it exists as an option, it is possible to obtain a list at any time.

DISTRIBUTE This routine is used to distribute the books evenly to the cylinders needed, to prepare all first sectors, and to produce a revised table. Once the CHARGE and DISCHARGE options have been completed, there will not be a evenly distributed set of images. Since the routines will function faster with an even distribution, the DISTRIBUTE can be used to produce the rearrangement of the images.

In operation, it writes all images as far back on the disc as possible. Once this has been done, the number of books per cylinder is determined by using the last entry in the table. The images are then placed in the cylinders as desired. The first sectors are adjusted, and a revised table is produced.

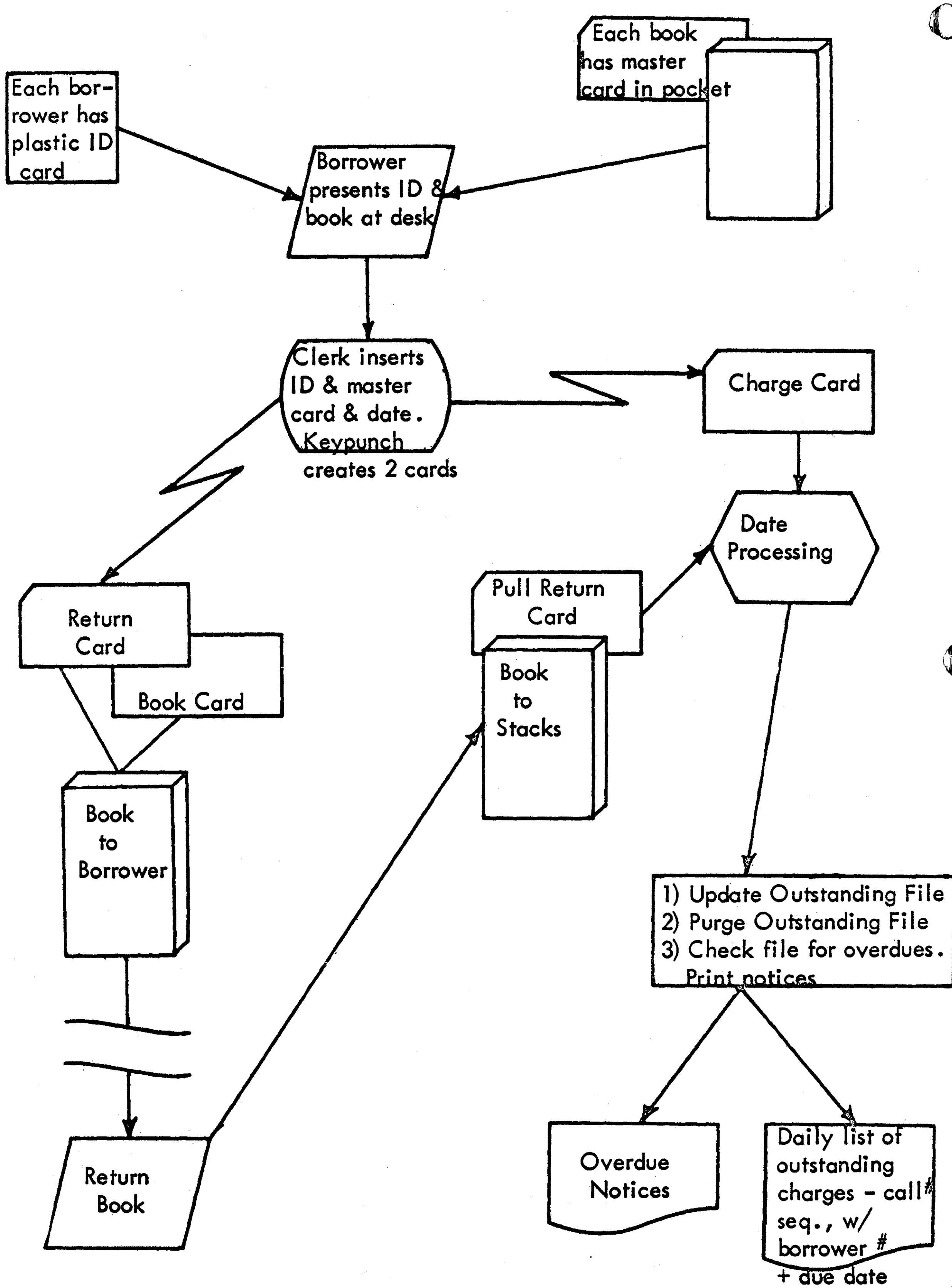
This program is automatically called after DISCHARGE's list. If CHARGE should not be able to add an image to a cylinder that should contain the image, a message is given to distribute and begin with a new CHARGE.

OVERDUE A list is produced of all books which are past due according to the date given on the control card. Any books with more than two overdue notices will be indicated by an asterisk. As the list is made, a card is also punched for each overdue book. This card is used on the collator to select the borrower's address card. This matching can be used to produce the overdue notice.

END OF JOB This will simply type out a message which indicated the end of the job.

As each option is processed, the control card is typed on the console typewriter. For any options which have data cards (charge or return), the last card should have 80 nines punched in it. This signals to the program that there are no more cards to process. The supervisory program can be easily modified to include more options.

Although this by no means solves all problems involved with library circulation, it does solve the three basic problems with a small computing system. It also suggests a very useful method of file storage and processing which might be applied to other types of inventories.


Each borrower has plastic ID card
Each book has master card in pocket
Borrower presents ID & book at desk
Clerk inserts ID & master card & date. Keypunch creates 2 cards
Charge Card
Date Processing
Return Card
Book Card
Book to Borrower
Pull Return Card
Book to Stacks
1) Update Outstanding File
2) Purge Outstanding File
3) Check file for overdues. Print notices
Return Book
Overdue Notices
Daily list of outstanding charges - call# seq., w/ borrower # + due date

1123661 2086390A76---------.T713-----------0866600300-

DUE DATE	ITEM	CLASS	CUTTER	YR	VOL	CY	PSN	BORROWER NO.
MO. DA. YR								

THE BOOK IS DUE ON THIS DATE

RETURN CARD

Do not remove this card from pocket. A charge will be made if this card is lost or mutilated.

UNIVERSITY LIBRARIES
UNIVERSITY OF SOUTHWESTERN LOUISIANA
Lafayette, Louisiana

2086390A76---------.T713-------

ITEM	CLASS	CUTTER	YR	VOL	CY	PSN

ITEM	CLASS	CUTTER	YR	VOL	CY	PSN

MASTER CARD
Do not remove this card from pocket. A charge will be made if card is missing or damaged when item is returned.

UNIVERSITY LIBRARIES
UNIVERSITY OF SOUTHWESTERN LOUISIANA

LAFAYETTE, LOUISIANA

LEWIS 21095

MASTER CARD
Do not remove this card from pocket. A charge will be made if card is missing or damaged when item is returned.

GEORGE GUY HAVARD JR
NAME

52035 65 10 42
STUDENT NUMBER YEAR DATE OF BIRTH

THE UNIVERSITY OF SOUTHWESTERN LOUISIANA
STUDENT IDENTIFICATION
(NOT VALID UNLESS SIGNED)

SIGNATURE

SPR SUM
UNIVERSITY VALIDATION

UNIVERSITY EDUCATION PAPERS

"U. S. L. STUDENT SCHEDULING"

Jack D. Testerman - U. S. L.
Earl K. Turner, Jr. - Shell Oil Co.

Box 133 U. S. L.
Lafayette, La. 70501

Phone: 318 CE-4-7349

Speech: Tuesday, November 29, 1966

Session Number: T-1.6

Pages of Text: 3

U. S. L. STUDENT SCHEDULING BY THE IBM 1620 COMPUTER

Earl K. Turner, Jr.
Jack D. Testerman

There is a tremendous amount of clerical work involved with the scheduling of students, most of which is on the shoulders of the registrar's office. The accompanying headaches tend to increase as the enrollment of the school increases, making it increasingly difficult to run an orderly and accurate registration. The old method of scheduling - by hand while the students are en masse - often needs to be revised or replaced.

In general, any registration process must include three kinds of activity, namely:

1. Creating a semester class schedule
2. Advising students about their academic programs
3. Assigning students to sections of courses.

The phase of scheduling that calls for the most work - much of it clerical - is student assignment. So this is obvious candidate for machineprocessing.

There have been several experiments in the area of student scheduling systems in the past few years, but on the whole, very few of these could be termed an ideal system. Nearly all of these systems involve the student requesting particular course sections and the computer program either accepting his choices, thereby acting as a tallyer, or rejecting his schedule requests because of the unavailability of one or more of his requested sections. There has also been the problem of controlling the number of drop-and-adds that will occur in any computerized system.

Of course an ideal system depends on the needs of the school - just how much of the system is desired to be automated, and the quantity of machines and personnel at hand.

Any registration system should have two main goals. First, to register students accurately, and secondly, to make registration for the student as simple and speedy as possible. In keeping with these aims, an IBM 1620 computer program was written at the University of Southwestern Louisiana (U. S. L.) which would accomplish these goals.

The U. S. L. procedure consists of the following:

1. An early registration period requiring the student to turn in a schedule request card at least one month in advance of the semester in which he wishes to enroll.
2. Requires an advising system for the student
3. Schedules the student's classes by computer
4. Mails the schedule to the student
5. Distributes schedules to the deans and department heads
6. Takes care of any absolutely necessary changes at the end of the registration period
7. Sends preliminary class rolls out to the professors before classes begin, and sends corrected ones after drop and add changes are completed.

The IBM 1620 Program

In order for any system to work efficiently, one must have the proper tools on hand. For this system, including the programs there are, of course, certain necessary pieces of equipment.

Equipment necessary:

1. IBM 1620, 40K digits of memeory, with disk and indirect addressing
2. IBM 1622 card read/punch unit
3. A keypunch
4. An off-line printer, i.e. IBM 407
5. A card collator
6. A card sorter
7. A card interpreter
8. A schedule decollator and burster

Features include:

1. A student check, so as not to allow duplication of scheduling
2. Student time exclusions
3. A section request option
4. A lunch hour built into the system
5. Scheduling done on the basis of 30 minute time intervals
6. Scheduable time extending from 7:30 a.m. to 9:30 p.m.
7. A maximum of 15 classes allowed per student
8. An average scheduling time of 10 seconds per student
9. Ability to handle up to 6900 class sections
10. Maximum of 99 sections per course
11. Ability to set maximum number of students per section as high as needed
12. Maximum of 99 different curricula codes (department numbers) for university use

13. Section balancing feature
14. Notifies the student if it is unable to schedule a particular class
15. A one-pass program - it registers the student and produces the class cards directly (these are listed as the student's schedule).

Differences of the U. S. L. System from Other Systems

Uniqueness - We have a big problem for small computers. Most computerized scheduling systems just use the computer as a tallying device; that is, the computer program either accepts or rejects the students' sectioned request. This program is built to assign sections of the requested classes when they are available; hence, it is a sectioning program. It will also schedule classes with pre-assigned section numbers as other scheduling programs do.

Some Disadvantages

1. The procedure demands precision of all aspects of the scheduling of students. At times, this is a difficult rule to follow.
2. There is no instant communication with the student. If the student has some difficulty with his schedule, correction time is limited. After receiving his schedule by mail, the student must see his advisor, make some provision for a change, then bring the requested change to the Office of the Registrar.

Advantages of the System

Some of the advantages of the U. S. L. computerized scheduling system are the following:

1. The symplicity of registration afforded the student
2. Allowing the administration to know the number of students enrolling very early so that any extra faculty required could be obtained in plenty of time. Also, estimates can be given other facilities such as the University Bookstore for purposes of predicting the number of books they will need in any courses
3. Not tieing up all of the faculty and offices in a great whirlwind session of one week; rather, spreading this over a much greater period of time where personnel who can be trained for this can handle it on a continuing basis
4. The centralized location of all results
5. The instant tabulation of numbers
6. An increase in speed with quite a bit less manpower for registration.

FORTRAN LANGUAGE TIMINGS ON THE 1620 MODELS I AND II, AND THE 360 MODELS 30, 40, 44, and 50

by

Robert L. Shutt
Sacramento Peak Observatory, AFCRL, Sunspot, New Mexico
(High Altitude Observatory Solar Project at Sacramento Peak)

I. Introduction

Mr. Champagne, Mr. Session Chairman, fellow members of COMMON, and guests at this meeting:

This paper presents the results of a series of Fortran language tests of the internal speeds of the IBM 1620 Models I and II, and the IBM 360 Models 30, 40, 44, and 50. I had intended this material for informal presentation in a discussion group, but at Mr. Champagne's suggestion, I have worked it into a paper. I was a little reluctant because when we made the first timing tests of this series, we had no idea that we were going to use the same programs to test all these machines. The programs were made up rather hastily and the tests conducted in a less than rigorous fashion. However, after several years, we more or less discovered that we had run the same programs on a number of IBM machines, and we used the results in the selection of our next computer. I believe these results are of general interest, especially to anyone who, like us, is going from a 1620 to a 360 system and wants to know what increase in internal speed he is buying.

II. The Computing Environments at the Sacramento Peak Observatory

The tests were performed for the Sacramento Peak Observatory of the Air Force Cambridge Research Laboratories. My employer is the National Center for Atmospheric Research, which provides certain research services, including data reduction, for the Air Force at the Sacramento Peak Observatory. Sacramento Peak Observatory is a solar observatory, devoted entirely to the study of the sun. We are located on a mountaintop in southern New Mexico at an altitude of 9200 feet. We have about a dozen telescopes, a variety of electronic instrumentation and data reduction equipment, a resident staff of six Ph.D. Astronomers, several visiting scientists and about 40 people providing a variety of support services.

We became "computerized" four years ago with a basic 1620 Model I, and we are now just outgrowing our 1620 Model II-1311 disk system. We operate on an open shop basis and use the computer as a research tool for the study of the sun. Our computer work is all scientific, involving computer solutions of mathematical models, computation of astronomical coordinates, the filtering, smoothing, and massaging of data, and correlational and other statistical studies of solar data.

We are currently pretty firmly in the IBM fold. We like their equipment and supporting services, and, in any case, not very many manufacturers will come way out into the boondocks to do business with us. We are fifty miles from our IBM representative, and he is a hundred miles from the Branch Office.

III. Description of the Tests

In 1963 we decided to upgrade our basic 1620 Model I computer. We were considering the Model II, floating point hardware, 1311 disk drive, extra memory, and other optional goodies. Since nearly all our work is done using Fortran, we were interested in Fortran language tests of

internal speeds, transfer rates (from the disk), and systems throughput. We made up a seven-part Fortran coded test package which we have run on at least four different 1620 systems. When we became interested in the 360 systems last year, we tried to make a similar evaluation. However, at that time such features as the 3211 disk were not supported in BPS Fortran and the I/O configuration varied. So we were able to meaningfully measure only internal speeds for Fortran operations. This is still very useful. At our installations, we do a lot of statistical work, such as cross-correlation of data, where the execution time goes as the square, or higher power, of the number of data points. We collect very long data strings, and it is no problem at all to use all the computing power a given machine has. We wanted to be able to tell our staff that with the Model N 360, you will be able to cross-correlate two data strings of so many data points in this many minutes. or you can invert a given size matrix in so many minutes. Our staff uses this type of information in planning the research projects to be conducted six months or a year from now.

A second use of this information was the evaluation of competitive bids on our new computer system in terms of "bangs per buck."

The tests which we made on all the systems are the following:

a. The time to do Fortran coded arithmetic driven by a "DO" loop. The arithmetic operations tested are: A + B, A * B, and sine (C) where

$$A = 3.1825E + 08$$
$$B = 5.8362E + 03$$
$$C = .85$$

b. A synthetic program with a mix of Fortran instructions. At the time I wrote this program, I had never heard of a Gibson mix, but my intent was the same as the intent of Mr. Gibson, or whoever designed his mix; that is, to measure the time a given machine requires to perform a "typical" scientific program. In our case, we used floating point variables. I point this out because some machines look much better, comparatively, doing fixed point arithmetic than they do doing floating point arithmetic, as you are aware.

IV. Results and Comments

The results are presented in Tables I and II. Table I gives the numbers of operations per second for A + B, A * B, and sine (C). It also gives the time to perform 10 iterations of the synthetic program. Table II lists some parameters of the systems tested.

Comments

1. The timings were made using the second hand of a wrist watch. I only claim accuracy to about 5% for these results. Since the tests were made over a period of about three years, and since my filing and notation system is atrocious, there may be some mistakes in these numbers or systems parameters, but I don't think there are many.

2. The tests on the 360/44 were performed by an operator in New York City with decks mailed there. The rest were performed by me on a "hands on" basis.

3. It is interesting to note that a 1620 Model II with all optional hardware is nearly a factor of 10 faster than a basic 1620 Model I.

4. the 360/40 is roughly 3 times as fast as the 360/30, and the 360/50 again 3 times as fast as the 360/40.

5. It is instructive to look at speed ratios for A + B, and for the synthetic program.

for A + B	$\frac{360/30}{1620\ II} = 5$	$\frac{360/40}{1620\ II} = 17$	$\frac{360/50}{1620II} = 53$
for the synthetic program	$\frac{360/30}{1620\ II} = 8$	$\frac{360/40}{1620\ II} = 25$	$\frac{360/50}{1620\ II} = 90$

I interpret this to mean that the 360 system has a more efficient logical design in that less "overhead" of data handling and transfer is paid in the execution of a program than is the case in the 1620.

6. The tests really measure a hardware-software system and not just CPU speeds. I have tacitly assumed the various Fortran compiler produces equally efficient object code.

As a result of these tests and a competitive evaluation, which considered other factors than internal speeds, we have ordered a 360/44 for installation next spring. The speed performance of this machine is tremendous. We are getting a speed improvement of better than 100 for a cost increase of something like 25%. The trade-off is inferior software support. For example, we will not get PL1, RPG, SPOOL, etc. And we will have to do a good deal of systems work to tailor the system to our needs. But we still feel we are getting a bargain. We believe, from some informal estimates, that the 44 is about 1/2 as fast as a 7090, and we know from a separate test that it is about 1/10 as fast as the CDC 6600. This makes the 44 a very fast machine but still well within the reach of a small to medium sized scientific installation like ours.

Thank you for your attention.

V. Acknowledgement

My special thanks to Mr. Frank Bird of the IBM El Paso branch office for arranging test time on most of these machines and for assistance in solving the proglems of running programs on new machines with unfamiliar software systems.

TABLE I. INTERNAL SPEEDS FOR THE IBM 1620 MODELS I AND II, AND THE IBM 360 MODELS 30, 40, 50, and 44

	1620 Model I	1620 Model II	360 Model 30	360 Model 40	360 Model 50	360 Model 44
Arithmetic Operations (number/sec)						
A + B	57	910	3000	9100	23,600	38,000
A * B	42	345	1760	5900	18,200	29,000
Sine (C)	3.5	34	250	910	2,860	3,800
$A = 3.1825 \times 10^8$ $B = 5.8362 \times 10^3$ $C = 0.85$						
Synthetic Program (time for 10 iterations)	1,130 sec	125 sec	15.5 sec	5 sec	1.4 sec	1.0 sec

TABLE II. SYSTEMS PARAMETERS

Systems	Floating Hardware	Memory Cycle Time	Significant Digits In Floating Word Mantissa	Significant Digits In Fixed Word	Software System	Other Features
1620 Model I	No	20 microsecs per 1 digit	8	5	Fortran with format	
1620 Model II	Yes	10 microsecs per 2 digits	8	5	Fortran II-D	Index register indirect addressing
360 Model 30	Yes	2 microsecs per 1 byte	7.2	4.5	BPS 16K Fortran IV	
360 Model 40	Yes	2.5 microsecs per 2 bytes	7.2	4.5	BPS 16K Fortran IV	
360 Model 50	Yes	2 microsecs per 4 bytes	7.2	4.5	BPS 16K Fortran IV	
360 Model 44	Yes	1 microsec per 4 bytes	7.2	4.5	BPS 16K Fortran IV	High speed general registers

IBM

245 Marquette Avenue
Minneapolis, Minnesota 55401
612/338-8541

International Business Machines Corporation

January 4, 1967

Mr. Champagne
University of Southwestern Louisiana
La Fayette, Louisiana

Dear Mr. Champagne:

The presentation on "1620 - 360 Simulation" covered the following:

1. Organization of the 1620 Simulator as it is received from IBM.

2. Generation of the 1620 Simulator.

3. /360 core utilization of the 1620 Simulator.
    - a. 1620 Simulation - no disk
    - b. 1620 Simulation - disk
        1. Core resident disk simulation
        2. Disk resident disk simulation.

4. Simulated 1311 disk records on the 2311.

5. Simulated sector arrangement.

6. 1620 Simulator Program Logic.

The session concluded with a discussion of topics of general interest to 1620 Simulator users, such as timings, 2311 formattings, and comparisons with the 1620 Emulator.

Sincerely,

H. P. Klysen

H. P. Klysen
Field Systems Center

HPK:mj