PROCEEDINGS OF THE

1620 USERS GROUP JOINT CANADIAN-MIDWESTERN REGION

FEBRUARY 19-21, 1964

AT THE O'HARE INN, DES PLAINES, ILLINOIS

FRANK H. MASKIELL

REGIONAL SECRETARY

TABLE OF CONTENTS

MEETING SCHEDULE

1620 USERS Group

February 19-21, 1964 O'Hare Inn, Des Plaines, Ill.

WEDNESDAY, FEBRUARY 19, 1964

8:45	GEN ERAL SESSION	Convention Hall	
	8:45 Announcements	" "	B. Burrows
	8:55 IBM Announcements	" "	J. Jessee
	9:00 IBM Announcements	" "	G. Percoco
	9:10 Programming Syst. Announcements	" "	L. Foster
	9:30 Index Register Concepts	" "	B. Socks
	9:50 Kingstran	" "	D. Jardine
10:30	Coffee & Rolls	" "	
10:45	SELECTED SPS TOPICS-Elementary	" "	
	SPS WORKSHOP-ADVANCED	Grecian Room	D. Pratt
12:15	Lunch	Convention Hall	
1:30	EXPLORATORY PROGRAMMING	" "	J. Morrissey
3:00	Coffee		
3:30	PANEL OF 1710 USERS	D 4	J. C. Hill
	SPS WORKSHOP-ADVANCED	Grecian Room	D. Pratt
7:30	SOUND-OFF SESSION	" "	J.A. N. Lee

THURSDAY, FEBRUARY 20, 1964

7:45	NEW USERS MEETING (coffee & rolls)	D 3	B. Burrows
8:45	INTRODUCTION TO MONITOR I	Convention Hall	IBM
	PANEL OF 1311 USERS	Grecian Room	B. Burrows
10:15	Coffee & Rolls	" "	& Conven. Hall
10:45	INTRODUCTION TO MONITOR I	Convention Hall	IBM
	MONITOR I WORKSHOP-ADVANCED	D 4	IBM
	Magnetic Tape Users Meeting	D 3	B. Robinson

12:15	Lunch	Convention Hall	
1:30	INTRODUCTION TO MATRICES	" "	C. Maudlin, Jr.
	MONITOR I WORKSHOP-ADVANCED	D 4	IBM
	EDUCATION PAPERS	Grecian Room	
	1:30 A New Course in "Computer Appreciation"	" "	C. Davidson
	1:45 A Survey of the Beginning Programming Course	" "	C. B. Germain
	2:00 Data Processing Technicians; an Integrated Training Approach at Hibbing Area Tech. Inst.	" "	W. J. McGraw
	3:00 Coffee	Conven. Hall, Grec.Rm. & D 4	
	3:30 Introduction to Regression	Conven. Hall	C.Phillip Cox
	MONITOR I WORKSHOP-ADVANCED	D 4	IBM
	APPLICATION PAPERS	Grecian Room	
	3:30 The IBM 1620 as an Analytical & Pre-Compositional Aide in 12-Tone Music		A. Tepper
	3:50 A Family of Test Matrices		A.C.R. Newberry
	4:10A New Random Number Generator		H.T. Wheeler
	4:30 Logic Theorm Detection Program		J. Wheatley
6:30	Movies	Convention Hall	IBM
	MONITOR I & II Demos	D 2	IBM
7:30	Repeat of 6:30 Program	Repeat	
8:30	Repeat of 6:30 Program		

FRIDAY, FEBRUARY 21, 1964

8:45	Panel of Commercial Data Processing	D 4	R. Thomas
	PROGRAMMING SYSTEMS PAPERS	Convention Hall	
	8:45 Magic I and II	" "	J.A. N. Lee
	9:15 Kingstran	" "	D. Jardine
	PANEL ON EDUCATION	Grecian Room	
10:15	Coffee & Rolls	Conven. Hall, Grec. Rm. and D 4	

10:45	ENGINEERING & CONTROL PAPERS	D 4	
	10:45 AUTOSPOT II Pre-Processor Prog.	D 4	D. McManigan
	11:00 Autospotless Numerical Control with the 1620	D 4	E. Ray Austin
	11:30 Montecarlo Techniques applied to Radio Chemistry	D 4	J.K. Lewis
10:45	PROGRAMMING SYSTEMS PAPERS	Convention Hall	
	10:45 Carleton College Compiler		D. Taranto
	11:00 Carleton Binary Simulator		W. Gage
	11:15 A Completely Relocatable SPS		M. Dorl
	11:30 Modifying Monitor I to Include Other Programming Systems		A. Purcell
10:45	PANEL ON EDUCATION	Grecian Room	
12:15	Lunch	Convention Hall	
1:30	GENERAL SESSION	" "	
	Sound-Off Answers	" "	
	Questions & Answers	" "	
3:00	Adjournment		

AGGERGAARD PAUL L	GREEN GIANT COMPANY	LE SUEUR	MINN
ALCORN HERBERT R	MISSOURI SCHOOL OF MINES & MET	ROLLA	MO
ALDRICH F C	CLDSMCBILE DIVISICN	LANSING	MICH
ALTENHOFF JOHN	ELECTRO-MOTIVE DIV-GMC	LA GRANGE	ILL
AMORT ANTHONY	BELCIT CORPORATION	BELOIT	WISC
ARENA A	IBM CORPORATION	WHITE PLAINS	N Y
ATKINS JOSEPH T	E I DUPONT DE NEMOURS CO INC	PARKERSBURG	W VA
ATKINS DR D FERREL	EASTERN ILLINOIS UNIVERSITY	CHARLESTON	ILL
AUSTIN E R	COMBUSTION ENGINEERING INC	CHATTANOOGA	TENN
AUSTIN HUBERT	TRI-STATE COLLEGE	ANGOLA	IND
BABIONE ROBERT C	USAF - ACIC	ST LOUIS	MO
BACHHUBER JOHN J	INST OF PAPER CHEM	APPLETON	WIS
BAEVERSTAD HAROLD L	SUNDSTRAND MACHINE TOOL	BELVIDERE	ILL
BARBUTES ROBERT F	IBM CORPORATION	YOUNGSTOWN	OHIO
BARRON JD	IBM	TORONTO	ONT
BARTH WILLIAM	EDO CORPORATION	COLLEGE POINT	N Y
BATHURST LYNN L	RUST ENGINEERING CO	BIRMINGHAM	ALA
BATSON WILLIAM B JR	NASHVILLE BRIDGE COMPANY	NASHVILLE	TENN
BEAUDREAU DOREEN	UNIVERSITY OF WISCONSIN	MILWAUKEE	WISC
BELONOS S P	COLUMBIA GAS SYSTEM	COLUMBUS 12	OHIO
BEMENT LYLE W	MALLORY TIMERS COMPANY	INDIANAPOLIS	IND
BERNIER J L	GREAT NORTHERN OIL COMPANY	ST PAUL	MINN
BEST ALBERT	GLIDDEN COMPANY	JACKSONVILLE	FLA
BEST WILLIAM R	VA BIOSTATISTICAL RSCH SUPP	HINES	ILL
BICKFORD PAUL	OKLA UNIV RESEARCH COMP CTR	OKLAHOMA CITY	OKLA
BILLINGHAM CAROL	LEONARD REFINERIES INC	ALMA	MICH
BLACK RICHARD H	UNIVERSITY OF WISCONSIN	MILWAUKEE 11	WISC
BLOMQUIST ROY	ILL INST OF TECH	CHICAGO	ILL
BOEKHOFF PETER G	MRD GATC	NILES	ILL
BONYUN DAVID A	LOYOLA COLLEGE	MONTREAL	QUE
BOWMAN RUTH T	E M KLEIN & ASSOC	CLEVELAND	OHIO
BRIDGFORD ROBERT	UNIVERSITY OF WISCONSIN	MILWAUKEE	WISC
BRIDGER DAVID	WASHINGTON UNIVERSITY	ST LOUIS	MISS
BRITTAIN PAUL E	U S NAVAL ACADEMY	ANNAPOLIS	MD
BROWN ROBERT W	GENERAL MOTORS INSTITUTE	FLINT 4	MICH
BUCHLER DANIEL	ANACONDA WIRE AND CABLE	SYCAMORE	ILL
BUHLIG W L	SPENCER CHEMICAL COMPANY	KANSAS CITY	MO
BURKETT S B JR	PAN AM AIRWAYS	PATRICK AFB	FLA
BURNS R E	AMERICAN WELDING & MFG CO	WARREN	OHIO
BURROWS WILLIAM A	DRAVO CORPORATION	PITTSBURGH	PA
CALLIS CLIFFORD L	GREENVILLE TEC	GREENVILLE	SC
CAPPS JOAN	UNIVERSITY OF ILLINOIS	CHICAGO	ILL
CARLSON MILTON E	NORTHERN ILLINOIS UNIVERSITY	DE KALB	ILL
CARLSON KERMIT H	VALPARAISO UNIVERSITY	VALPARAISO	IND
CASLIN JAMES C	U S AIR FORCE	WRIGHT-PAT AFB	OHIO
CHERNIAK DR E A	CARLETON UNIVERSITY	OTTAWA	ONT
CHRISTIANSON G	DOUGHBOY INDUSTRIES INC	NEW RICHMOND	WISC
CHRISTOFK RUSSELL E	ELECTRIC MACHINERY MFG CO	MINNEAPOLIS 13	MINN
CHRISTIANS DONALD D	U S NAVAL ACADEMY	ANNAPOLIS	MD
COLE WESLEY G	PIONEER HI-BRED CORN COMPANY	DES MOINES	IOWA
COOK LEROY L	DIV OF RADIOLOGICAL HEALTH	ROCKVILLE	MD
COSTELLO DONALD F	WIS STATE COLLEGE	OSHKOSH	WIS
COUCH JOHN D	KANSAS STATE TEACHERS COLLEGE	EMPORIA	KAN
COX ROBERT C	HUMBLE OIL & REFINING CO	BATON ROUGE	LA
CRULL JAMES	KEARNEY & TRECKER CORPORATION	WEST ALLIS	WISC
DABE RODNEY G	CONSOER TOWNSEND ASSOC	CHICAGO	ILL
DALTON CLARENCE H	SOUTHEAST MISSOURI STATE COL	CAPE GIRARDEAU	MO
DALY JAMES E	DIV OF RADIOLOGICAL HEALTH	ROCKVILLE	MD
DANON REBECCA	WESTERN RESERVE UNIVERSITY	CLEVELAND	OHIO

DAVIDSON JAMES L	LONG ISLAND LTG	HICKVILLE	NY
DAVIDSON CHARLES	UNIVERSITY OF WISCONSIN	MADISON	WISC
DAWSON J	IBM CORPORATION	CALGARY	ALB
DECK JAMES C	INLAND STEEL COMPANY	EAST CHICAGO	IND
DEMERATH ALBERT J	KIMBERLY-CLARK CORPORATION	WEST CARROLLTON	OHIO
DENISON GEORGE B	FAIRBANKS MORSE CO	BELOIT	WIS
DEUEL MICHAEL	KEARNEY & TRECKER CORPORATION	WEST ALLIS	WISC
DEVENNEY WILLIAM S	WABASH COLLEGE	CRAWFORDSVILLE	IND
DICKERSON RICHARD F	CONTAINER CORP OF AMERICA	CHICAGO	ILL
DILLINGER DR J L	UNIVERSITY OF ILLINOIS	CHICAGO	ILL
DORL MICHAEL	UNIVERSITY OF WISCONSIN	MADISON	WISC
DOUGLAS LEO C	ARGONNE NATL LAB	LAMONT	ILL
DOULOFF A A	TRANS CANADA PIPE LINES	TORONTO	ONT
DRESSLER BYRON B	KENT STATE UNIV	KENT	OHIO
DUGGAN JOHN	HAWKER SIDDELEY CANADA LTD	TORONTO	ONT
DULICK ROBERT W	YOUNGSTOWN SHEET AND TUBE CO	YOUNGSTOWN	OHIO
DUNN JACK T	AVCO CORPORATION	HUNTSVILLE	ALA
DYE DAVID	IBM	WHITE PLAINS	NY
EDIE ROBERT	STATE UNIVERSITY OF NEW YORK	BUFFALO	N Y
EDWARDS DAVID O	E I DU PONT	CIRCLEVILLE	OHIO
EDWARDS DR M LLOYD	KANSAS STATE TEACHERS COLLEGE	EMPORIA	KAN
EIKENBERRY PROF R S	UNIVERSITY OF NOTRE DAME	NOTRE DAME	IND
ELWELL WALTER G	NEBRASKA WESLEYAN UNIVERSITY	LINCOLN	NEBR
ESCHBACH DAREL	THE UNIVERSITY OF TOLEDO	TOLEDO	OHIO
FIELD A WAYNE	FIRST AEROSPACE CONTROL SQUAD	COLO SPRINGS	COLO
FIELD J A	UNIVERSITY OF TORONTO	TORONTO 5	ONT
FLATT JAMES G	PUB UTIL DIST #2 GRANT COUNTY	EPHRATA	WASH
FLIESS MANFRED	JONES & LAUGHLIN STEEL CORP	PITTSBURGH 30	PA
FODOR JOYCE	UNIVERSITY OF WISCONSIN	MADISON	WISC
FORSS LEONARD	SUNDSTRAND AVIATION	ROCKFORD	ILL
FORSYTH D W	CONTINENTAL CAN COMPANY	CHICAGO	ILL
FOSTER C L	IBM CORPORATION	SAN JOSE	CAL
FOX J BAINE	WASHINGTON & LEE UNIVERSITY	LEXINGTON	VA
FRAILING WILLIAM D	UPJOHN COMPANY	KALAMAZOO	MICH
FRANK DR WERNER	BOWLING GREEN STATE UNIVERSITY	BOWLING GREEN	OHIO
FREDERICK DR KENNETH J	ABBOTT LABORATORIES	NORTH CHICAGO	ILL
FUCHS IVAN	UNITED AIRCRAFT OF CANADA	MONTREAL	QUE
GADE EUGENE T	ABBOTT LABORATORIES	NORTH CHICAGO	ILL
GAGE WILLIAM	CARLETON COLLEGE	NORTHFIELD	MINN
GANIERE CARL	KEARNEY & TRECKER CORPORATION	WEST ALLIS	WISC
GAUNT WENDY	IBM CORPORATION	MINNEAPOLIS	MINN
GERMAIN CLARENCE B	COLLEGE OF ST THOMAS	ST PAUL	MINN
GILBERT PAUL	NW STATES PORTLAND CEMENT	MASON CITY	IOWA
GILBERT PAUL F	USAF - ACIC	ST LOUIS	MO
GIVENS CLYDE JR	NORTHERN ILLINOIS UNIVERSITY	DE KALB	ILL
GLANDER HAROLD	CARROLL COLLEGE	WAUKESHA	WISC
GOLDSMITH TT	LAWRENCE COLLEGE	APPLETON	WIS
GONZALES RICHARD L	BRADLEY UNIVERSITY	PEORIA	ILL
GRANT JEAN M	J & L STEEL-GRAHAM RES LAB	PITTSBURGH 30	PA
GRAVES EARL	CONDUCTRON CORPORATION	ANN ARBOR	MICH
GREGG ROBERT D	LINK-BELT COMPANY	CHICAGO 9	ILL
GRIBBEN CLIFFORD J	INDIANAPOLIS POWER & LIGHT CO	INDIANAPOLIS	IND
GRIFFITH WALLACE	CENTRAL MO STATE COLLEGE	WARRENSBURG	MO
GUTEKUNST HANS	ELECTRO-MOTIVE DIV-GMC	LA GRANGE	ILL
HADLOCK ALAN	BELOIT CORPORATION	BELOIT	WISC
HAGER HAROLD W	SOUTHEAST MISSOURI STATE COL	CAPE GIRARDEAU	MO
HAJNAL TIBOR E	TRANS CANADA PIPE LINES	TORONTO	ONT
HAKE DAVID A	LOUISIANA POLYTECHNIC INST	RUSTON	LA
HAMILTON JAMES R JR	IBM	ARLINGTON	VA

HARR STEPHEN B	IBM	PITTSBURGH	PA
HARRIS EDIE	IBM	DETROIT	MICH
HATFIELD FRED A	LINE MATERIAL INDUSTRIES	ZANESVILLE	OHIO
HATTORI M	KOZO-KEIKAKU STRUCT ENG FIRM	TOKYO	JAP
HELLER NORA L	MONSANTO RESEARCH CORP	MIAMISBURG	OHIO
HERSHEY COLLIN	DRAVO CORPORATION	PITTSBURGH	PA
HETHERINGTON RICHARD	UNIVERSITY OF KANSAS	LAWRENCE	KAN
HEWETT SYLVIA	E I DUPONT DE NEMOURS	FLINT	MICH
HEYWORTH A	UNIVERSITY OF ALBERTA	EDMONTON	ALB
HILL JOHN CARROLL	PURDUE UNIVERSITY	LAFAYETTE	IND
HINTZ ANN	IBM CORPORATION	CHICAGO	ILL
HOLLMEIER RONALD J	PIONEER SERVICE & ENGINEERING	CHICAGO	ILL
HOLMES JOHN W	COOPER-BESSEMER CORP	MOUNT VERNON	OHIO
HOCK EVERETT L	WAGNER ELECTRIC CORP	ST LOUIS	MO
HORRIGAN TIMOTHY J	COOK ELECTRIC CO	MORTON GROVE	ILL
HOTCHKISS GARY	ETHYL CORPORATION	FERNDALE	MICH
HUGHES NORMAN L	VALPARAISO UNIVERSITY	VALPARAISO	IND
IYENGAR SRINIVASA H	SKIDMORE OWINGS & MERRILL	CHICAGO 3	ILL
JANSEN JOHN H	CENTURY ELECTRIC COMPANY	ST LOUIS	MO
JARDINE D A	DUPONT OF CANADA LTD	KINGSTON	ONT
JESSEE J J	IBM CORPORATION	CHICAGO	ILL
JORGENSEN ODD	DRAVO CORPORATION	PITTSBURGH	PA
KALLER C L	UNIVERSITY OF SASKATCHEWAN	SASKATCHEWAN	CAN
KAPPLE FRANK	WHEATON COLLEGE	WHEATON	ILL
KAUFMAN MARVIN	UNIVERSITY OF MISSOURI	COLUMBIA	MO
KEEFER EDWARD H	AMERICAN CYANAMID	BOUND BROOK	N J
KELEHAR BRANT	IBM	BIRMINGHAM	ALA
KELLER DR ROY	UNIVERSITY OF MISSOURI	COLUMBIA	MO
KELLMAN SIDNEY	NAVAL AIR ENGINEERING LAB	PHILADELPHIA 12	PA
KERR H B	TENNESSEE POLYTECHNIC INST	COOKEVELLE	TENN
KIEN DR GERALD	NORTHWESTERN U MEDICAL SCHOOL	CHICAGO	ILL
KLATSKY STEPHEN S	ERIE MINING COMPANY	HOYT LAKES	MINN
KOLLER E	PULP & PAPER RES INS OF CAN	MONTREAL 2	QUE
KRAFT DON	ELECTRO-MOTIVE DIV-GMC	LA GRANGE	ILL
KRUPKA R M	CONTINENTAL CAN COMPANY	CHICAGO	ILL
LANGE ROBERT G	AUTOMATIC ELECTRIC LABS	NORTHLAKE	ILL
LARCADE GEORGE A	HALLIBURTON COMPANY	DUNCAN	OKLA
LAWRENCE DEAN W	MIDWEST RESEARCH INSTITUTE	KANSAS CITY 10	MO
LEE DR J A N	QUEENS UNIVERSITY	KINGSTON	ONT
LEE E S	UNIVERSITY OF TORONTO	TORONTO 5	ONT
LEICHUS RICHARD	IBM	NYC	NY
LERICK GEORGE E	ERIE MINING COMPANY	HOYT LAKES	MINN
LERRY DAVID P	IBM CORPORATION	NEW YORK CITY	N Y
LITTELL DR ARTHUR S	WESTERN RESERVE UNIVERSITY	CLEVELAND	OHIO
LOGAN S WM	P R MALLORY & CO INC	INDIANAPOLIS	IND
LOGUE W E	AMERICAN WELDING & MFG CO	WARREN	OHIO
LOHREY JOHN	GIDDINGS & LEWIS MACH TOOL CO	FOND DU LAC	WISC
LONG JULIA	ABBOTT LABORATORIES	NORTH CHICAGO	ILL
LONG PHIL	IBM	KANSAS CITY	MO
LONGMAN JACK	WESTERN MICHIGAN UNIVERSITY	KALAMAZOO	MICH
LYNCH MARY LIZ	IBM	CLEVELAND	OHIO
MAAS ALBERT C	GREEN GIANT COMPANY	LE SUEUR	MINN
MACDONNELL L M	ROYAL MILITARY COLLEGE OF CAN	KINGSTON	ONT
MAGEE ROLAND H	THE MAGNAVOX COMPANY	FORT WAYNE	IND
MANIOTES DR JOHN	PURDUE UNIVERSITY	HAMMOND	IND
MARINELLI RICHARD	STATE UNIVERSITY OF NEW YORK	BUFFALO	N Y
MARKOVICH ERNEST	UPJOHN COMPANY	KALAMAZOO	MICH
MARQUARDT ROBERT W	MONSANTO RESEARCH CORP	MIAMISBURG	OHIO
MARTIN KATHLEEN	UNIVERSITY OF CINCINNATI	CINCINNATI 21	OHIO

MASKIELL FRANK H	PENNSYLVANIA TRANSFORMER DIV	CANONSBURG	PA
MATHEWS GERALD	BELOIT CORPORATION	BELOIT	WISC
MATTHEISS PAUL K	SUN OIL COMPANY	MARCUS HOOK	PA
MATTMUELLER DONALD	ARMY MAP SERVICE	WASHINGTON	DC
MAUDLIN CHARLES E JR	UNIVERSITY OF OKLAHOMA	NORMAN	OKLA
MAURER ROBERT	WESTERN MICHIGAN UNIVERSITY	KALAMAZOO	MICH
MC DONALD DAVID R	FEDERAL RESERVE BANK	MINNEAPOLIS	MINN
MC KEE LOWRY L	HOFSTRA UNIVERSITY	HEMPSTEAD	N Y
MC MANIGAL DAVID F	IBM CORPORATION	POUGHKEEPSIE	N Y
MCGRAW WILLIAM J	HIBBING AREA TECH SCHOOL	HIBBING	MINN
MEAGHER PROF JACK R	WESTERN MICHIGAN UNIVERSITY	KALAMAZOO	MICH
MEIDL RANDOLPH A	JOSEPH SCHLITZ BREWING CO	MILWAUKEE	WISC
MELLO LEONARD	GEOPHYSICS CORP OF AMERICA	BEDFORD	MASS
MERGEN FRANK	BRADLEY UNIVERSITY	PEORIA	ILL
MILLER EDWARD	H R B - SINGER INC	STATE COLLEGE	PA
MINNE DAVE	IBM	CHICAGO	ILL
MORRISSEY J	IBM CORPORATION	NEW YORK	N Y
MOSCHETTI JOHN	ELLIOTT COMPANY	GREENSBURG	PA
MYERS CLARENCE E	ELI LILLY AND COMPANY	INDIANAPOLIS 6	IND
MYLIUS WM G JR	RUST ENGINEERING CO	BIRMINGHAM	ALA
NAIKELIS U STANLEY	UNIVERSITY OF ILLINOIS	CHICAGO	ILL
NAYLOR R W	SPENCER CHEMICAL COMPANY	KANSAS CITY	MO
NELSON MARVIN L	DAWES LABORATORIES INC	CHICAGO 32	ILL
NEUHAUSER VINCENT E	LUKENS STEEL COMPANY	COATESVILLE	PA
NEWBERY DR A C R	UNIVERSITY OF ALBERTA	CALGARY	ALB
NEWTON LAWRENCE E	M D ANDERSON HOS&TUM INST	HOUSTON 25	TEX
NICELEY JOHN B	RICHLAND TECH EDUC CENTER	COLUMBIA	S C
NOBLE HENRY J	RCA SERVICE CO	COCOA BEACH	FLA
NOONAN BERNARD	UNIVERSITY OF MANITOBA	WINNIPEG 19	MAN
NORBY RALPH	LINK-BELT COMPANY	CHICAGO 9	ILL
NORTHAM JADE I	UPJOHN COMPANY	KALAMAZOO	MICH
ONEIL R T	AMERICAN OIL COMPANY	WHITING	IND
ORLOFF MILTON J	GENERAL MOTORS INSTITUTE	FLINT	MICH
OTTO FREDERICK M	CLARK OIL & REFINING CO	BLUE ISLAND	ILL
OWEN DAVID G	MIAMI-DADE JUNIOR COLLEGE	MIAMI	FLA
PACHON HEBERTO	AUTOMATIC ELECTRIC LABS	NORTHLAKE	ILL
PALMER P G	POLYMER CORPORATION LIMITED	SARNIA	ONT
PARKER CHARLES D	GENERAL MOTORS PROVING GROUND	MILFORD	MICH
PARKER S THOMAS	KANSAS STATE UNIV	MANHATTAN	KAN
PARKER ROBERT	NORTHERN ELECTRIC	OTTAWA	ONT
PATERSON A R	IBM	MONTREAL	QUE
PAUL RONALD	STANDARD OIL COMPANY	CLEVELAND	OHIO
PERCOCO G	IBM CORPORATION	WHITE PLAINS	N Y
PETRABORG JARROLD	LINK-BELT COMPANY	CHICAGO 9	ILL
PLUM GEORGE	MC DOWELL-WELLMAN ENG CO	CLEVELAND 14	OHIO
POORE JESSE H JR	LOUISIANA POLYTECHNIC INST	RUSTON	LA
PORTMAN CAROL	NUMERICAL CONT&COMP SERVICES	CLEVELAND	OHIO
PRATT RICHARD L	AIR FORCE INSTITUTE OF TECH	WRIGHT-PAT AFB	OHIO
PRITZ HOWARD B	TRI-STATE COLLEGE	ANGOLA	IND
PURCELL PHILIP	LEONARD REFINERIES INC	ALMA	MICH
PURCELL ALAN V	UNIVERSITY OF WISCONSIN	MADISON 6	WISC
RAAB PAUL V	ALLEN-BRADLEY COMPANY	MILWAUKEE	WISC
RAFAELIAN LEON A	CONT AVIATION & ENG CORP	DETROIT	MICH
RECTOR ROBERT R	LOUISIANA POLYTECHNIC INST	RUSTON	LA
REIMAN DAVID H	NATIONAL LIFE & ACC INS CO	NASHVILLE	TENN
REITER NED	SEWS REG PLAN COMM	WAUKESHA	WIS
RESTA EDWARD V	IBM	SAN JOSE	CAL
RICHMOND EUGENE L	JOSEPH SCHLITZ BREWING CO	MILWAUKEE	WISC
ROBINSON D G	DUPONT OF CANADA LTD	KINGSTON	ONT

ROBINSON ROBERT J	MARQUETTE UNIV	MILWAUKEE	WIS
ROEDER LT GEORGE L	FIRST AEROSPACE CONTROL SQUAD	COLO SPRINGS	COLO
ROSS RICHARD D	UNIVERSITY OF MISSISSIPPI	UNIVERSITY	MISS
RUGH J PALMER	E I DUPONT DE NEMOURS	FLINT	MICH
SANDERS PAUL G	ABBOTT LABORATORIES	NORTH CHICAGO	ILL
SANTILLI A J	NATIONAL STEEL CORPORATION	WEIRTON	WVA
SCARLETT JOHN C	SURFACE COMBUSTION DIVISION	TOLEDO 1	OHIO
SCHATZ NATHAN	NAVAL AIR ENG LAB	PHILA	PA
SCHAUSS CHARLES E	U S WEATHER BUREAU	WASHINGTON	D C
SCHERER MATHIAS E	PIONEER SERVICE & ENGINEERING	CHICAGO	ILL
SCHETTLER RICHARD C	SOUTHERN ILLINOIS UNIVERSITY	CARBONDALE	ILL
SCHROEDER ROBERT L	H F CAMPBELL COMPANY	DETROIT 10	MICH
SCOTTI CARMIN J	IBM CORPORATION	CHICAGO	ILL
SCOTT EDWARD E	LUKENS STEEL COMPANY	COATESVILLE	PA
SENN JOHN	IBM CORPORATION	MILWAUKEE	WISC
SHANAHAN GREGORY J	CECO STEEL PRODUCTS	CICERO	ILL
SHEEDY PAUL J	CLARK OIL & REFINING CO	BLUE ISLAND	ILL
SHIOURA NORIO	UNIVERSITY OF ILLINOIS	CHICAGO	ILL
SIMMONS HAROLD W	CLARK OIL & REFINING CORP	BLUE ISLAND	ILL
SMALTZ HUBERT J	DE PAUW UNIVERSITY	GREENCASTLE	IND
SMITH WELBORN H	AIRBORNE INSTR LAB	MELVILLE	NY
SMITH NOEL T	INDIANA STATE COLLEGE	TERRE HAUTE	IND
SMITH BRYAN	NORTHERN ELECTRIC	OTTAWA	ONT
SMITH V G	UNIVERSITY OF TORONTO	TORONTO 5	ONT
SOCKS B J	IBM CORPORATION	CHICAGO	ILL
SOLOMON LOUIS	NATIONAL LEAGUE FOR NURSING	NEW YORK	N Y
SPRADLING GARY A	UNIVERSITY OF OKLAHOMA	NORMAN	OKLA
STEBER GEORGE R	UNIVERSITY OF WISCONSIN	MILWAUKEE	WISC
STEELE LAURA B	GENERAL MOTORS INSTITUTE	FLINT	MICH
STEINHARDT NICHOLAS	INDIANAPOLIS POWER & LIGHT CO	INDIANAPOLIS	IND
STEPHENUTCH J R	ROCKFORD BD OF EDUCATION	ROCKFORD	ILL
STONE J	IBM CORPORATION	WHITE PLAINS	N Y
STORRY J O	SOUTH DAKOTA STATE COLLEGE	BROOKINGS	SDAK
STRANGE C CLINTON JR	BELOIT CORPORATION	BELOIT	WIS
STROUSE E I	COLUMBIA GAS SYSTEM	COLUMBUS 12	OHIO
STYLES JIMMIE C	JUNIOR COLLEGE OF BROWARD CY	FT LAUDERDALE	FLA
SULLIVAN RONALD A	CANADA DEPT OF AGRICULTURE	OTTAWA	ONT
TACK O CHARLES	U OF MD SCHOOL OF MEDICINE	BALTIMORE	MD
TARANTO DONALD	CARLETON COLLEGE	NORTHFIELD	MINN
TAYLOR DEAN JR	U S NAVAL ACADEMY	ANNAPOLIS	MD
TEPPER ALBERT	HOFSTRA UNIVERSITY	HEMPSTEAD	N Y
TERAJEWICZ GEO	AMERICAN AIRLINES	FLUSHING 71	N Y
THAYER RAYMOND J	LINE MATERIAL INDUSTRIES	ZANESVILLE	OHIO
THIEL CONNIE ANN	IBM	CLEVELAND	OHIO
THOMAS ROBERT J	DE PAUW UNIVERSITY	GREENCASTLE	IND
THOMAS RICHARD B	FEDERAL RESERVE BANK	MINNEAPOLIS	MINN
THOMPSON MAJOR IVAN B	SCHOOL OF SYSTEMS & LOGISTICS	DAYTON	OHIO
THOMSON GEORGE	ETHYL CORPORATION	FERNDALE	MICH
THURSTON A M	IBM COMPANY LIMITED	TORONTO	ONT
TRANTUM JOHN	MRD GATC	CHICAGO	ILL
TRAVER WARD B	AMERICAN OIL COMPANY	WHITING	IND
TREVINO JOSE	INSTITUTO TECH DE MONTERREY	MONTERREY	MEX
TUCK HARVEY R	IBM	DAYTON	OHIO
TUCKER LEE	SOUTH DAKOTA STATE COLLEGE	BROOKINGS	SDAK
TUNNEY JAMES L JR	J R AHART INC	DAYTON 6	OHIO
TURNER RONALD R	LANSING BOARD OF EDUCATION	LANSING	MICH
URBAN CHARLOTTE A	IBM	ST PAUL	MINN
VANSEN RICHARD J	CONT AVIATION & ENG CORP	DETROIT 15	MICH
VANSICKLE GEORGE	INDIANAPOLIS POWER & LIGHT CO	INDIANAPOLIS	IND

VERVAERT JOHN	BELOIT CORPORATION	BELOIT	WISC
VICK E	DOUGHBOY INDUSTRIES INC	NEW RICHMOND	WISC
VIDEBECK DR RICHARD	UNIVERSITY OF MISSOURI	COLUMBIA	MO
VROOM K E	PULP & PAPER RES INS OF CAN	MONTREAL 2	QUE
WALKER DONALD C	DRAVO CORPORATION	PITTSBURGH	PA
WANG JAMES C	TUSKEGEE INSTITUTE	TUSKEGEE	ALA
WAYBRIGHT GLENN E	RUST ENGINEERING COMPANY	PITTSBURGH	PA
WEBER HEINZ O	SURVEYER NENNIGER & CHENEVERT	MONTREAL 25	QUE
WEIDEMAN WILLIAM	MARQUETTE UNIV	MILWAUKEE	WIS
WESTERMAN E A	JOHN MORRELL AND COMPANY	OTTUMWA	IOWA
WHEATLEY DR J	QUEENS UNIVERSITY	KINGSTON	ONT
WHEELER H T	CARLETON UNIVERSITY	OTTAWA	ONT
WIGDAHL ALLEN B	ALLEN-BRADLEY COMPANY	MILWAUKEE	WISC
WIGHTMAN MARY G	HEWITT ASSOC	LIBERTYVILLE	ILL
WILHELM JACQUELYN	THE UNIVERSITY OF TOLEDO	TOLEDO	OHIO
WILLIAMS C R	DOW CHEMICAL COMPANY	HOUSTON	TEX
WILMHOFF KENNETH F	THE MAGNAVOX COMPANY	FORT WAYNE	IND
WILSON ROBERT	JEFFERSON CITY PUBLIC SCHOOLS	JEFFERSON CITY	MO
WINK ANNA T	INDIANA STATE COLLEGE	INDIANA	PA
WOOD F W	NATIONAL TECH CORP	WEIRTON	WVA
WOODS ARTHUR P JR	ARMCO STEEL CORP	MIDDLETOWN	OHIO
WOODS STANLEY W	STATE HGWY COMM OF WISCONSIN	MADISON	WISC
WOODWORTH J A	DOW CHEMICAL COMPANY	HOUSTON	TEX
WRIGHT DONALD L	GEORGETOWN UNIVERSITY	WASHINGTON	D C
YACU ZUHAIR A	RALSTON PURINA COMPANY	ST LOUIS 8	MO
YAMAMURA E A	DUPONT OF CANADA LTD	KINGSTON	ONT
YANAGISAWA HARUO	CONTAINER CORP OF AMERICA	CHICAGO	ILL
ZAHN JULIA	MCDONNELL AIRCRAFT CORPORATION	ST LOUIS 66	MO
ZUKE LOIS	VA HINES HOSP-BIOSTAT RES DEPT	HINES	ILL

A Relocatable Programming System

A relocating assembler for the purpose of this discussion is one which assembles a program in such a form that it may be placed anywhere in memory at load time; i.e. the program does not have to be re-assembled to change its origin. This has been accomplished in the past by manually placing flags on the digits of the op-code or by using origins greater than the machine size. The author finds that these systems have definite disadvantages and has decided on a system in which each operand (either instruction or DSA) carries with it a system assigned relocating tag which determines its relocatability status.

An assembler which produces a relocatable object deck should be of much use in programs employing subroutines (Fortran, SPS) or in systems of programs.

The R-SPS system which should be in the library by June of 1964 consists of essentially a 3 pass system: Two assembly passes and one compressing pass. The assembly passes may be batched as may the compressing pass.

The main disadvantage of the system is the increased size of the object deck.

The following pages taken from the program write-up further serve to explain the system and its use.

Michael Dorl
Engineering Computing Laboratory
University of Wisconsin
5-5-64

Program Abstract

Title: R-SPS

Author: Michael Dorl
Engineering Computing Laboratory
University of Wisconsin

Date: 1-1-64

Users Group Code: 3155

Direct Inquiries to: Prof. C. H. Davidson
Director,
Engineering Computing Laboratory
University of Wisconsin
Madison, Wisconsin 53706
Phone 608-262-3892

Description/Purpose

R-SPS provides the capability of relocating ordinary SPS programs at load time. The assembled decks which it produces are relocatable in the full sense of the word. The system provides the programmer with complete control over the relocating feature, either through manual intervention or through programming. In addition error checking has been expanded and assembly speed increased.

Specifications

Storage: 40 K or larger (self-adjusting)

Equipment:

a) Card system
b) Automatic divide
c) Indirect addressing

The automatic divide feature is used only to process one seldom used instruction (MORG). Thus the Auto-divide restriction could be easily overcome by not using that instruction.

The program cannot be used on a 20 K machine.

Program Language UW-SPS

Language Used In Write-Up English

Remarks

Although UW-SPS is not in the programming library, it is quite similar to R-SPS (see write-up). Copies are available upon request from the author.

Table of Contents

ENTRY

Program Write-up

General Data

Program Name: R-SPS

Date:

Programmer: Michael Dorl
Engineering Computing Laboratory
University of Wisconsin
Madison, Wisconsin
Phone 608-262-3892
User Code 3155

Machine Configuration Required

a) Card I/O
b) 40 K or more
c) Indirect addressing
d) Auto-divide

Program Developed On

a) 1620 MOD 1
b) Card I/O
c) Indirect addressing
d) 60 K
e) Auto-divide

Programming System Used in Development

R-SPS was assembled using an R-SPS assembled by UW-SPS. Although UW-SPS is not in the program library, it is quite similar to R-SPS. Copies of UW-SPS may be obtained from the author.

The final form of R-SPS is a dumped deck. The dumper is also available from the author.

Introduction

R-SPS is a Relocatable Symbolic Programming System. It gives the programmer much more complete and easy control over the loading and relocating operation than any of the several other systems available for the 1620. The decks which it produces can be relocated at load time under complete control of the programmer. In addition, assembly has been speeded up by a random symbol table store and recovery technique so that the assembler is reader bound during pass 1 and punch bound during pass 2 .

In the following discussion it is assumed that the reader is familiar with the use of IBM SPS and the SPS Reference Manual (IBM 1620/1710 Symbolic Programming System--C 26-5600).

Card Input Format

Two separate card input formats are provided.

1) UW-SPS format

columns 7-12	for label
columns 14-17	for instruction
columns 19-77	for operands and comments

2) IBM SPS format

columns 6-11	for label
columns 12-15	for instruction
columns 16-74	for operands and comments

The author believes the UW-SPS format to be superior in that all three fields are separated by at least one blank on the card, permitting consistent skipping to the beginning of each new field during keypunching.

Note that the IBM SPS format is modified slightly in that columns 16-74 are available for operands and comments, rather than columns 16-75.

Use of either card format is at the option of the user. The assembler recognizes both types as legal, and they may be intermixed in the same program.

Columns 1-5 are available for card identification under both card formats; however these columns will not appear on the object listing. Since column 6 is used in determining the card format, its use is prohibited for other than label field under IBM Format.

Columns 78-80 for UW-SPS Format and columns 75-80 for IBM SPS Format are also available for card identification.

Statements

Statements are of three types:

1) Processor Instructions, which give the assembler certain commands,
2) Machine Instructions, which are translated to actual Machine operations,
3) Declarative Instructions, which tell the processor to set aside certain work areas or set up actual constants to be used.

Special Characters

The @, *, and $ are used for special purposes in R-SPS.

The @ sign is used to call for a record mark to be incorporated in either a declarative operation or in the P or Q field of an instruction.

If an @ sign is used in a declarative operation it must appear as the last character in the constant. For example,

```
X  DC      3,03@
   DC      2,3@
   DC      1,@, *
   DAC     2,@@
   DACF    2,A@
```

In the DAC and DACF instruction only terminal @ signs generate records marks.

In the case where a label is attached to a DC with an @ sign, the address assigned to the label will be the address of the record mark.

The @ sign may be used to generate a record mark.in position P_6 or Q_{11} of an instruction:

```
           A = 15000
TDM        A,@        15 15000 0000‡
B          @          49 0000‡ 00000
```

The @ will be translated as a record mark in position P_6 or Q_{11} respectively; however the @ when used in this way must appear alone.

The * when used for address adjustment refers to:

1) the last assigned address when used in declarative or processor instructions.
2) the address of the instruction in which it is used.

The * is treated as a relative address.

The $ sign is used to call for a symbolic address under a given head character. For example,

A$HI

refers to HI headed by A .

Operands used in instructions, for length definition, or for assigned addresses may be of the following form.

$$\pm A \pm B \pm C \pm D$$

Up to four terms may be included and may be added or subtracted as indicated. No multiplication is allowed.

The dollar sign may be used to generate a group mark at the end of a numeric constant in the same manner as an @ sign.

The rules which determine whether an operand is relocatable (and therefore must be changed during load time) or absolute are given below:

1) The sum or difference of two absolute quantities is absolute.

2) The sum or difference of an absolute and a relative quantity is relative.

3) The difference of two relative quantities is absolute.

4) Calling for the sum of two relative quantities is illegal.

The sign of a relocatable operand is preserved at load time, for only its magnitude is increased by the relocating vector.

Processor and Declarative Operations

In declarative operations, labels or symbolic addresses may be assigned as in IBM SPS; however, the relocating of such a label is determined by its assigned address according to the following rules:

1) An integer address is absolute and makes the associated label absolute. For example, in

```
A  DS  2, 807
```

A will be taken as equivalent to an absolute 807 wherever it appears.

2) A symbolic address gives its relocatability to the label. If Q is an absolute quantity and Z is a relative quantity then in the following statements

```
A   DS   , Q
AA  DS   , Z
```

A will be absolute and AA will be relative.

3) A processor-assigned address is relative.

```
e.g. A  DS  2
```

In the Declarative Operations which follow all constant lengths must be absolute.

DC, DAC, DS, DAS, DNB

These pseudo-operations define storage in exactly the same way as described in the IBM 1620/1710 SPS Manual. The sole difference is that if comments are included without assigning an address to the constant and if the assigned address field consists of several blanks, the processor does not take the address to be zero but rather assumes it to be omitted.

Blanks must not appear in the middle of a numeric constant.

DACF (Define Alphabetic Constant Flagged)

This pseudo-operation performs the same function as does the DAC, except that alphabetic pairs appear with their high order digits flagged.

DSC, DSS, DSB

These pseudo-operations are not available in R-SPS.

DORG

This pseudo-operation is used as described in the IBM 1620/1710 Manual. Although its operand may be either absolute or relative, it is treated as relative.

DEND

This pseudo-operation is used to define the end of a R-SPS program [illegible] optionally to specify the location at which it is to begin. The beginning address may be either relative or absolute; if relative the relocating vector will be added to it at load time. Whenever this operation is used control is taken from the loader and passed either to the specified address or to a "Halt; Branch-to-zero" pair. This operation does not cause the arithmetic tables to be included in the object program.

MORG (Modify ORiGin)

This operation is used to set the next assigned address register to the next larger multiple of the operand. The operand which must appear must also be absolute, non-negative, and non-zero.

This operation is especially useful for starting tables at even multiples of 10, 100, or 1000. It should be kept in mind that the relocating vector will influence the value of the last assigned address at load time.

It is for this operation that automatic divide hardware is required.

LOAD (Punch LOADer)

The R-SPS system employs a relocatable loader which must be called for. The statement:

```
LOAD  X
```

causes the loader to be incorporated in the object deck in such a form that its first digit will be X . The operand must be absolute, or may be omitted. If the operand is omitted the beginning digit of the loader must be supplied from the console typewriter at load time. The operation also causes the add table to loaded.

This feature makes it possible to load part of the program from a loader at position X and load another part from a loader at a different position Y . It should be remembered, however, that incorporation of a second loader destroys all old values of Relocating Vector and the next Load Address. The loader occupies 1220 digits.

The instruction makes it possible to assemble a program in parts while including only one loader in the final deck.

<u>RVEC</u> (Set <u>R</u>elocating <u>VEC</u>tor)

This instruction is used for specifying the value of the relocating vector at assembly time. The operation

RVEC 10184

causes location 99 to be filled in with 10284 at load time. This instruction has no effect on the last assigned address register. The operand must be absolute. The relocating vector is initially set to 1000.

<u>TABL</u> (Punch <u>TABL</u>e)

This instruction, which requires no operand, causes the arithmetic tables to be punched out. The add table is loaded along with the loader but the multiply table is not; thus if any use is made of the multiply table this command must be given.

<u>HED</u> (<u>HE</u>a<u>D</u>)

This instruction is used as in IBM SPS. The heading character may be either alphabetic or numeric.

<u>LINK</u>

The pseudo-op

LINK A,B,C,D,E,F

is used to pass control at load time from the loader to a program which has either been loaded previously by the loader or to a program originally in core.

The A operand, which may be a relocatable symbol, is the address to which control is to pass. The B, C, D, E, and F operands serve as identification fields, and are available for use by the user's program. Upon reading the card at load time the following fields are in core.

locations 95-99	Present Relocating Vector
locations 90-94	The memory address into which the next digit of an instruction will be loaded.
locations 85-89	The address to which a branch must occur to pass control back to the loader.
location 84	A zero.
locations 26-30	A
locations 43-47	The last assigned address register at the end of pass 1 .
locations 48-62	The operands B , C , D , E , F

Upon filling in these areas, a B to 0 is executed thus transferring control to a type six card at zero. (See Output Format).

This card relocetes the A operand if necessary and transmits control to the address A via a BT A, A-1. After the user has modified any of the above constants which are required, control may be passed back to the loader via a B to -89 or BB instruction.

The single digit at location 84 is a switch which controls whether or not the program following the LINK instruction is to be loaded into core or ignored. If location 84 contains a zero the program following the LINK statement down to the next LINK statement will be loaded; if on the other hand location 84 is set to a flagged one the program following the LINK statement down to the next LINK statement will be ignored by the loader (although the cards will be sequence checked).

It should be noted that the loader does not relocate the operand's B - F .

<u>END</u>

This instruction, which also requires no operand, is to define the end of an R-SPS program without halting the loading operation. This instruction is used to end part of a fragmented program which is to be loaded as one piece.

Examples of Use of Processor Instructions

A) This example shows how LINK might be used to type a program name on the console typewriter after the first few cards have been loaded.

```
          LOAD    30000
          RVEC    0
          DORG    50000
   TYPE   RCTY
          WATY    COM1
          BB      ,,, (B -89 would also do).
          DORG    *-9
   COM1   DAC     13,PROGRAM NAME@ ,
          LINK    TYPE
                  :
```

B) This program shows how LINK could be used to read the relocating vector for a program from the typewriter.

```
          LOAD    55000
          RVEC    1000
  START   RCTY
          WATY    MESSA
          RNTY    95
          BC4     *-36
          SF      95
          CM      99,0
          BN      START
          B       -89,,,(OR  BB)
  MESSA   DAC     6,RVEC=@, *-2
                  :
```

Machine Instructions

Any of the symbolic operations in the following table may be used and are translated as shown. In addition numeric op-codes in the operation field are recognized providing that only two appear and are left justified. Four new operations,

TON - Turn ON
TOFF - Turn OFF
BON - Branch ON
BOFF - Branch OFF

are included. They are used as switches as explained below.

It is often convenient to remember a past condition by setting a two way switch to either a zero or flagged one, and then testing it at some later time with either a BNF or BD instruction. The use of the four new instructions is equivalent to the following old instructions.

```
TON       ADSW             TDM       ADSW,0
TOFF      ADSW             TDM       ADSW,-1
BON       Z,ADSW           BNF       Z,ADSW
BOFF      Z,ADSW           BD        Z,ADSW
```

A symbolic label associated with an instruction is a relative quantity.

The flag operand is used as in IBM SPS with one exception. Blanks may be included in the flag operand to set apart its various parts.

e.g. 0 1 10

However a pair of characters cannot be connected across a blank.

```
e.g.   1  1  0     is illegal
       1  10       is legal
```

TABLE 1

TABLE OF ALLOWABLE MNENONIC OPCODES AND THEIR 1620 EQUIVALENTS

A	21	XXXXX XXXXX
AM	11	XXXXX XXXXX
B	49	XXXXX XXXXX
BA	46	XXXXX 01900
BB	42	XXXXX XXXXX
BC1	46	XXXXX 00100
BC2	46	XXXXX 00200
BC3	46	XXXXX 00300
BC4	46	XXXXX 00400
BD	43	XXXXX XXXXX
BE	46	XXXXX 01200
BH	46	XXXXX 01100
BI	46	XXXXX XXXXX
BL	47	XXXXX 01300
BLC	46	XXXXX 00900
BME	46	XXXXX 01600
BMO	46	XXXXX 01700
BN	47	XXXXX 01300
BNA	47	XXXXX 01900
BNC1	47	XXXXX 00100
BNC2	47	XXXXX 00200
BNC3	47	XXXXX 00300
BNC4	47	XXXXX 00400

BNE	47	XXXXX 01200
BNF	44	XXXXX XXXXX
BNH	47	XXXXX 01100
BNI	46	XXXXX XXXXX
BNL	46	XXXXX 01300
BNLC	47	XXXXX 00900
BNME	47	XXXXX 01600
BNMO	47	XXXXX 01700
BNN	46	XXXXX 01300
BNP	47	XXXXX 01100
BNR	45	XXXXX XXXXX
BNRD	47	XXXXX 00600
BNV	47	XXXXX 01400
BNWD	47	XXXXX 00700
BNZ	47	XXXXX 01200
BOFF	43	XXXXX XXXXX
BON	44	XXXXX XXXXX
BP	46	XXXXX 01100
BRD	46	XXXXX 00600
BT	27	XXXXX XXXXX
BTM	17	XXXXX XXXXX
BV	46	XXXXX 01400
BWD	46	XXXXX 00700
BZ	46	XXXXX 01200
C	24	XXXXX XXXXX
CF	33	XXXXX XXXXX
CM	14	XXXXX XXXXX
D	29	XXXXX XXXXX
DM	19	XXXXX XXXXX
DN	35	XXXXX XXXXX
DNCD	35	XXXXX 00500
DNTY	35	XXXXX 00100
H	48	XXXXX XXXXX
K	34	XXXXX XXXXX
LD	28	XXXXX XXXXX
LDM	18	XXXXX XXXXX
M	23	XXXXX XXXXX
MF	71	XXXXX XXXXX
MM	13	XXXXX XXXXX
NOP	41	XXXXX XXXXX
RA	37	XXXXX XXXXX
RACD	37	XXXXX 00500
RATY	37	XXXXX 00100
RCTY	34	XXXXX 00102
RN	36	XXXXX XXXXX
RNCD	36	XXXXX 00500
RNTY	36	XXXXX 00100

S	22	XXXXX XXXXX	
SF	32	XXXXX XXXXX	
SM	12	XXXXX XXXXX	
SPTY	34	XXXXX 00101	
TBTY	34	XXXXX 00108	
TD	25	XXXXX XXXXX	
TDM	15	XXXXX XXXXX	
TF	26	XXXXX XXXXX	
TFM	16	XXXXX XXXXX	
TNF	73	XXXXX XXXXX	
TNS	72	XXXXX XXXXX	
TOFF	15	XXXXX 0000J	← FLAGGED ONE
TON	15	XXXXX 00000	
TR	31	XXXXX XXXXX	
WA	39	XXXXX XXXXX	
WACD	39	XXXXX 00400	
WNTY	38	XXXXX 00100	
WN	38	XXXXX XXXXX	
WNCD	38	XXXXX 00400	
WATY	39	XXXXX 00100	
SK	34	XXXXX X07X1	
RDGN	36	XXXXX X07X0	
WDGN	38	XXXXX X07X0	
CDGN	36	XXXXX X0701	
RTGN	36	XXXXX X0704	
WTGN	38	XXXXX X07X4	
CTGN	36	XXXXX X07X5	
RDN	36	XXXXX X0702	
WDN	38	XXXXX X07X2	
CDN	36	XXXXX X07X3	
RTN	36	XXXXX X07X6	

Errors

In the course of assembling a program, various error conditions can arise as illustrated in the following table.

After encountering an error condition the assembler types out an error code, a line number as referred to the last labeled statement, and a copy of the offending statement. The machine then halts.

Depending on the setting of console switches one and two, the error is treated as shown below when the start key is depressed:

Switch 2 on ----------	errors are ignored
Switch 1 on ----------	statements in which errors occur are treated as NOP's.

If the user elects to NOP an instruction in which an error occurs, the listing of the program contains a comment that the error has occurred in place of the mnenonic instruction.

If switches 1 and 2 are off the assembler expects a corrected statement to be entered from the console typewriter. The statement is read into the cleared input-area. Correct processing can not be assured if more than 80 characters are typed.

If an error in typing is made, the user can recover via console switch 4.

Additional Errors

Three additional error conditions can occur which are not treated as above.

1) If the program is loaded in a 20K machine the comment "MACHINE TOO SMALL" will be typed on the console typewriter. It is not possible to proceed.

2) If the symbol table becomes full the comment "SYMBOL TABLE FULL" will be typed. All following symbols will be checked for multiple definition, but they will not be defined. The comment is typed only once.

3) If at the end of pass 2 the contents of the last assigned address counter do not agree with the contents of that counter as of the end of pass 1, the comment "END CONFLICT" is typed. Processing proceeds after this comment.

TABLE 2

ERROR CODES

C-1	DORG operand is missing or zero
C-2	DEND or END operand is negative
C-3	DNB length is illegal
C-4	Assembler instruction contains a label
C-5	Incorrect card format
D-0	Illegal length specified in constant or illegal first operand
D-1	Illegal length specification for DS, DC, DAS, DAC, or DACF
D-2	DAC or DACF length specified will not fit on card
D-3	Illegal address specified for DS, DC, DAS, DAC, or DACF

D-4	More than one arithmetic sign occurs in a DC
D-5	More than fifty numeric characters supplied or more characters supplied than specified in a DC, DAC or DACF
D-6	Field not blank after record mark in DC
D-7	Field after alphabetic constant not blank
D-8	More than 10 terms in a DSA
D-9	DSA term includes@ or **
F-1	Non-numeric character in flag operand
F-2	Flag operand sub terms not in ascending order
I-1	Illegal identification operand in LINK
I-2	Illegal LINK address
I-3	Illegal LOAD address
I-4	Illegal RVEC operand
P-0	Field contains something in addition to ** or character
P-1	Too complicated an expression
P-2	Alphabetic symbol contains too many characters or numeric constant contains too many digits
P-3	Unmatched operator or operand
P-4	Illegal use of alphabetic symbol
P-5	Two operators in a row
P-6	Two operands in a row
P-7	Illegal character in an operand
P-8	Multiply relocatable constant
N-1	Leading numeric character followed by alphabetic character in op-code
N-2	Valid numeric op-code followed by non-blank characters
N-3	Op-code not in table
S-0	Symbol contains special character or leading numeric character
S-1	Undefined symbol
S-2	Previously defined symbol
S-3	Symbol undefined with full symbol table.

Operating Procedure

The following list of operations must be performed in order to run R-SPS:

1) Load the R-SPS deck. It is not necessary to clear memory.

2) Press start. If you desire operating procedures to be typed out turn switch 4 on and press start again. If you wish to bypass this type-out turn switch 4 off and press start.

3) The console typewriter asks: "IS SOURCE DECK TO BE STORED ON DISK, TYPE YES OR NO". If you have a disk file and wish to store the source deck on the disk for use during pass 2, type a YES. .If you do not wish to use this option type NO.

The machine then comments "SWITCH 1 ON TO CHANGE DISK MODE". This provides for change in the case of error in the above operation.

4) Place the source program in the read hopper and start the reader.

The source program will be read in and processed. If any errors are discovered, they may be treated individually as described in the ERROR section. At the end of pass one the comment "END OF PASS 1" will be typed out.

5) If you desire output on pass 2 turn switch 3 off and press start. If you desire R-SPS to suppress output turn switch 3 on and press start.

6a) If the source program has been stored on disk during pass 1, processing for pass 2 will proceed using the stored source statements.

6b) If the source program was not stored on disk during pass 1 it must be read in again during pass 2.

7) If the R-SPS program has been producing an object deck, near the end of pass 2 the program will type "SWITCH 1 ON FOR SYMBOL TABLE". The symbol table will be punched out if switch 1 is on and start is pressed. Otherwise the pass is finished, after start is depressed and a comment is typed.

Steps 3-5 may be repeated for many source programs without reloading the processor.

For those installations without a disk file steps 3 and 6a may be ignored. They may be eliminated from the processor by placing a MINUS SIGN in column 1 of the last card.

The object deck which R-SPS produces will not in general be ready to use until it is compressed. (The compressor punches out the loader).

8) Load the compressor.

9) Enter your object deck and press start.

The compressor will read the object deck and produce a compressed deck which may be loaded into the machine.

Steps 8 and 9 may be repeated for many object decks without reloading the compressor.

All decks which are processed or loaded must be in correct sequence. If an error in sequence is discovered a typewriter comment will be made. Rearrange the deck in correct order and proceed.

Output Format

There are six types of output cards as specified by the digits 1-6 in column 76 of each card. They are used as specified below:

1) Card type 1 is used to load up to five instructions per card into memory. Columns 1-60 hold the instruction while columns 61-70 hold a pair of rel-tags (digits which determine the relocatability of the various P and Q fields) for each instruction on the card. Each pair of rel-tags pertains to one instruction, the first to the P field and the second to the Q field. In all such uses of rel-tags a 2 implies a relative quantity while a 4 stands for an absolute quantity. Furthermore, the number of instructions is determined by the number of rel-tag pairs.

2) Type 2 cards are used to specify the next digit into which an instruction will be loaded. The value in columns 1-4 of the card is stepped by the relocating vector and saved in NEXDIG of the loader.

3) Type 3 cards cause the loader to transfer control to the card itself. Instructions located in columns 0-79 of the card are then executed and control is returned to the loader by a branch to 89 indirectly. These cards are used to:

a) load or reset the relocating vector,

b). change the next card check number (see patch cards).

4) Cards with a 4 in column 76 are used to load records of absolute numeric information into memory. They consist of a record of information beginning in column 1 of the card, a beginning address A in memory for the record, a memory address B to be preserved during the loading operation, and a single rel-tag which determines the relocatability of A and B. A is in columns 56-60, B in columns 61-65, and the rel-tag in column 55. This card type is similar to the one used exclusively in SPS to load information.

5) This card type is used to load DSA's into core. Up to 10 addresses appear in columns 1-50 of the card followed by one rel-tag for each address in columns 52-61. Again the number of items is determined by the number of rel-tags.

6) This final card type is used to process the LINK statement. The card loads the following data into locations 0-72:

```
A      * + 30 , 99,, THESE INSTRUCTIONS ARE
A      * + 23 , 99,, NOP S IF A ABSOLUTE
BT      A, A-1
B      -89,WIDTH
DSA     X1,X2,X3,X4,X5
```

A return address to the loader is filled in locations 85-89 and location 84 is set to a zero before control is transferred to location zero (see LINK statement).

A 4 digit consecutive card number is found in columns 77-80 of all cards.

Patch Cards

Various errors in the object deck can be corrected by means of patch cards placed in the compressed deck. These patch cards may be of the form of any of the six card types specified above.

For example to place an

```
AM        A,2,10
```

in location 1000 relative, where A is 14287 relative, the following two cards could be used.

```
                                         - 287
_
01000                                    2
1114387000-2              24             1
^                          ^            ^- 288
|____ column 1             |           /
          column 61 _______|          /
                                 column 76
```

These two cards would be incorporated into the patched deck before the last card and the whole deck renumbered.

An alternate procedure to renumbering the deck is to incorporate a type 3 card to change the next card check number located in the digit of the loader. The following 3 cards would then do the trick.

```
                                           -287
_                                          
01000                                      2-288
1114287000-2             24                1-289
                _
1100089-002216000890-2881200089-0022  /    3-290
                                     /     ^-288
                      _             /      |
                4900089 ___________/_______|
                          <column 76
```

Important Addresses

Assembler

Beginning Pass 1	00822
Beginning Pass 2	00894
Symbol Table Punch Routine	05582
Initialization	30002

Loader (located at L)

Next Card Check Number	L+22
Relocating Vector	00099
Next Instruction Address	00094

Compressor

Entry	20506

407 Board

The board wiring shown in the following diagram can be used for listing the uncompressed output of an R-SPS program. It also serves to list IBM and UW-SPS uncompressed output. The switch operations are as follows:

Switch 1	on double space off single space
Switch 2	on for IBM SPS
Switch 3	on for R-SPS

All switches are off for UW-SPS.

KINGSTON FORTRAN II

FOR THE IBM 1620 DATA PROCESSING SYSTEM

by:

J.A.A. Field,[1] D.A. Jardine,[2], E.S. Lee,[1]
J.A.N. Lee,[3] and D.G. Robinson[2]

Presented at the Joint Canadian-Midwest Region
Meeting of the 1620 Users Group, Chicago,
February 19-21, 1964

[1] Dept. of Electrical Engineering, University of Toronto, Toronto, Ontario.

[2] Research Centre, Du Pont of Canada Limited, Kingston, Ontario

[3] Computing Centre, Queen's University, Kingston, Ontario

ACKNOWLEDGEMENTS

During the early stages of developing this system, many people in the 1620 Users Group were canvassed for useful ideas on compiler and systems construction. To all those who, in any way, contributed to this venture, the authors extend their heartfelt thanks.

We would like to recognize the following people who made particularly useful contributions to the project:

J.W. Holmes[1] - for his extremely well written arithmetic and function subroutines which appear, with some modification, in this system.

F.H. Maskiell[2] - for many helpful suggestions, particularly in the coding and testing of the arithmetic and function subroutines.

C.H. Davidson[3] - 1620 Users Group representative on the A.S.A. Fortran II subcommittee, for explaining to us the structure of American Standard Fortran II, and for pointing us in the right direction for extending the language.

[1] Cooper-Bessemer Corp., Mount Vernon, Ohio.

[2] McGraw-Edison Corp., Penn. Trans. Div., Canonsburg, Pa.

[3] University of Wisconsin, Madison, Wis.

HISTORY

The writing of compilers seems to be one of the more popular pursuits of the members of the 1620 Users Group. At least six different FORTRAN compilers for the 1620 have been written by non-IBM personnel, which testifies to the enthusiasm and ability of 1620 users and to their very real desire to build the best possible mousetrap.

All previous user-written compilers have accepted variations of the FORTRAN I language, with the exception of the University of Wisconsin FORGO, a load-and-go compiler for student problems, which accepted a somewhat restricted FORTRAN II. To our knowledge, KINGSTON FORTRAN II is the first user-written FORTRAN II for the 1620. We hope that this initial effort will encourage others to tackle the problem and improve on our system in the same way that improvement followed improvement in the user-written FORTRAN I compilers.

The initial impetus for KINGSTON FORTRAN II came in about August 1963, from those of us living in Kingston, Ontario, when we started to find out how UTO FORTRAN operated, with the intention of providing a suitable FORTRAN for a 40K 1620. It soon became apparent that many useful features of FORTRAN II could be incorporated at little extra work. Messrs. Lee and Field, authors of UTO FORTRAN, were approached for ideas and suggestions, the outcome of which was a decision to join forces. After some preliminary discussion, it was found that it would be no more work to write a whole new system than to make the desired alterations in UTO FORTRAN.

The basic concepts were conceived in three rather long evening sessions during the October 1963, 1620 Users Group Meeting in Pittsburgh, Pa. By the end of this meeting the source language structure and the organization and general logic of the compiler were developed and agreed upon. The various sections were then allocated to the individuals best qualified to handle them. By the first week in January, the main sections of the compiler had been written and tested and it remained to tie the pieces together in a operating system. This was done in Kingston, Ontario, during late January, when all 5 authors worked for five days on two identical 40K 1620's (Du Pont of Canada and Queen's University).

We hope that Users with 40K 1620's will find the system useful and easy to operate. We have tried to include every useful idea from other people's efforts so that the system would be as speedy and compact as possible.

The work was divided as follows:

J.A. Field	- Input/Output statements, DO statements, input/output subroutines, FORMAT statement.
D.A. Jardine	- Arithmetic and function subroutines, write-ups and operating manuals.
E.S. Lee	- Compilation of arithmetic expressions.
J.A.N. Lee	- Compilation of everything not handled by the other authors.
D.G. Robinson	- Symbol table organization, including COMMON, DIMENSION, EQUIVALENCE, TYPE.

KINGSTON FORTRAN II

This write-up describes a FORTRAN system for the IBM 1620 equipped with automatic division, indirect addressing, additional instructions (TNS, TNF, MF), card input-output and minimum 40K memory. It is assumed that a Model E-8 or larger 407 is available for listing.

The language is that of IBM's FORTRAN II with a few modifications and a number of additions. For the purposes of this write-up it is expected that the reader is at least on speaking terms with the FORTRAN II language.

The compiler for this system batch compiles a source program in one pass, at approximately twice the speed of existing compilers for the 1620. The execution speed of the object program is also approximately twice that of IBM's FORTRAN II. Considerable effort has been made to speed up all important parts of the system; in addition, more core storage is available for the object program than existing FORTRAN II compilers allow.

SOURCE PROGRAM CARDS

These are as required for IBM FORTRAN II. Any number of continuation cards are possible, but the statement may not contain more than 300 characters (blanks not included except in Format statements).

ARITHMETIC PRECISION

Real numbers: 8 digit mantissa, 2 digit exponent.

Notation is excess 50; (i.e. 1.0 = 5110000000)

Integer numbers: 4 digits, modulo 10000

VARIABLES

These are as in IBM FORTRAN II. 1 to 6 alphabetic or numeric characters, starting with a letter, which, for integer variables, must be one of I, J, K, L, M, N, unless otherwise specified in a TYPE declaration.

SUBSCRIPTS

A variable with, at the most, two subscripts appended to it can refer to an element of a one- or two-dimensional array. Three dimensional subscripting is not permitted. A subscript may be an expression of any

desired complexity, provided only that the result of the evaluation of the expression be an integer quantity. This should be positive if you want to avoid trouble. However, a zero or a negative subscript can be used. To use this effectively, the programmer must know how data areas are laid out in memory. See the operating instructions:

Examples of Subscripts:

```
I
3
2+MU
MU+2
J*5+M
5*J
6*J-K+2-10/L+M
4*J(K+2-L+M)+K(M(N+2))/3
FIXF(A*B+3.0**SIN(X))+L/2
```

The variable in a subscript may itself be subscripted, and this process of subscripting may be carried on to any desired depth of subscripting. It can, in fact, be carried far beyond the point where the average programmer understands what he is doing.

SUBSCRIPTED VARIABLES

Only singly or doubly subscripted arrays may be defined. The size of these must be specified in a DIMENSION statement.

EXPRESSIONS

These are defined and organized exactly as in IBM FORTRAN II.

LIBRARY FUNCTIONS

Ten library (closed) functions are included in the KINGSTON FORTRAN II System. These are listed in Table I.

TABLE 1

Closed Subroutines

Function Definition	Function Name(s)	No. of Arguments	Type Of Function	Type Of Argument
Sine of the argument	SIN	1	Real	Real
Cosine of the argument	COS	1	Real	Real
Exponential (e^x) of the argument	EXP	1	Real	Real
Square Root of the argument	SQRT	1	Real	Real
Natural logarithm of the argument	LOG	1	Real	Real
Arctangent of the argument	ATAN	1	Real	Real
Arctangent of (arg_1/ arg_2)	ARCTAN	2	Real	Real
Signum of the argument; =-1.for X<0.,=0. for X,0.,=+1. for X>0.	SIGNUM	1	Real	Real
Absolute value of Arg 1 with the sign of Arg 2	SIGN	2	Real	Real
Choosing the larger value of the two arguments	MAX	2	Real	Real
Choosing the smaller value of the two arguments	MIN	2	Real	Real

Table 2 lists the open or built-in functions. These are compiled in-line every time the function is referred to.

TABLE 2

Function Definition	Function Name	No. of Arguments	Type Of Function	Type Of Argument
Absolute value of the argument	ABS	1	Real	Real
	ABS	1	Integer	Integer

Table 3 lists closed functions which are permanently stored in the machine, whether or not they are mentioned by name in a FORTRAN source program.

Function Definition	Function Name	No. of Arguments	Type Of Function	Type Of Argument
Floating an integer	FLOAT	1	Real	Integer
Truncation, largest integer in the argument, modulo 10,000, with sign of argument	FIX	1	Integer	Real

THE ARITHMETIC STATEMENT

The arithmetic statement is the same as in IBM FORTRAN II except for the extensions in complexity of evaluation of subscripts.

CONTROL STATEMENTS

The control statement flexibility in standard FORTRAN's leaves something to be desired, particularly where the program is complex and core storage is at a premium. These conditions, it might be noted, are the normal ones for almost all problems. KINGSTON FORTRAN II attempts to improve this situation by expanding the capabilities of the ASSIGN and assigned GO TO statement and by extending the ASSIGN concept to the other control statements.

ASSIGN STATEMENT

```
ASSIGN i TO n
```

In IBM FORTRAN II, the ASSIGN statement is used only in conjunction with an assigned GO TO statement. For instance,

```
ASSIGN 3 TO J

GO TO J, (3,5,9,243)
```

will cause a branch to the statement numbered 3.

The effect of the ASSIGN statement is to "equate" the non-subscripted integer variable J to statement number 3. The subsequent GO TO J, (3,5,9,243) is then interpreted as GO TO 3.

In KINGSTON FORTRAN II, this concept has been modified and expanded considerably. To describe these changes, the following definitions are used:

Statement Label - A statement label is the name attached to the memory location containing the first instruction compiled from the statement identified by the label. There are two kinds of statement labels:

Numeric Statement Label - usually known as a statement number. An unsigned integer number of from one to four digits long.

Alphabetic Statement Label - A variable which may be subscripted to any desired complexity and which by one or more ASSIGN statements has been equated to a numeric statement label (statement number).

It is most important to realize the difference between a statement label and an arithmetic variable. ASSIGN 3 TO J will place in J the address of the first instruction compiled from statement number 3. J = 3 will cause the number 0003 to be placed in J. The sequence of statements

```
ASSIGN 3 TO J
GO TO J
```

will cause a branch to statement numbered 3. However,

```
J = 3
GO TO J
```

will result in disaster. Moreover,

```
ASSIGN 3 TO J
J = J + 1
GO TO J
```

will not transfer control to the statement numbered 4. Arithmetic on assigned variables is not permitted; assigned variables are not in any way the same as arithmetic variables, except that they may be subscripted and stored in an array. They may also appear in COMMON, DIMENSION, and EQUIVALENCE statements.

It is possible in KINGSTON FORTRAN II, to equate two alphabetic statement labels by an ASSIGN statement. If the first statement label in the ASSIGN statement is alphabetic, it must be enclosed in parentheses.

The following examples illustrate the ASSIGN statement:

```
ASSIGN 3 TO N    (St. label N is equated to St. label 3)
ASSIGN (N) TO J  (St. label J is equated to St. label N)
ASSIGN 3 TO I(K) (same as the line above. K must have been
                  defined before this statement and I must be
                  dimensioned).
ASSIGN (I(K))  TO  L(3+M/4-M**3)
                 (same as above.  The alphabetic statement
                  labels can be subscripted as desired).
```

Since the primary definition of a statement identifier is its occurrence as a statement number, it is necessary that any given statement identifier must ultimately be defined (through a series of ASSIGN statements if necessary) in terms of a statement number. Failure to observe this rule will cause trouble. For example,

```
3    A = B
     —
     ASSIGN (J) TO K(L)
```

is not correct, because J has not been associated with any statement identifier when the ASSIGN statement is executed. However,

```
3    A = B
     —
     ASSIGN 3 TO J
     —
     ASSIGN (J) TO K(L)
     —
```

is correct.

Alphabetic statement labels may be used in the following control statements:

```
GO TO (both unconditional and assigned)
IF (SENSE SWITCH i)
IF (arithmetic expression)
Computed GO TO
```

Alphabetic statement labels may not be used in a DO statement.

GO TO STATEMENT

GO TO n   unconditional GO TO

GO TO n, $(n_1, n_2, --- n_m)$ assigned GO TO where n is a statement label. If n is alphabetic, then it must previously have been defined in an ASSIGN statement. The assigned GO TO statement is treated exactly like the GO TO statement. The comma and parenthesized list are optional and will be accepted but ignored by the compiler.

Computed GO TO Statement

GO TO $(n_1, n_2, n_3 \text{---} n_m), i$

where $n_1, n_2 \text{---} n_m$ are statement labels. If alphabetic they must have been previously defined by ASSIGN statements. i is a fixed point (integer) variable or expression. i may be subscripted as desired.

ARITHMETIC IF STATEMENT

IF(a)n_1, n_2, n_3

where a is an integer or real (floating point) expression of any complexity, and n_1, n_2, n_3 are statement labels. If alphabetic, n_1, n_2, n_3 must have been previously defined in ASSIGN statements.

IF (SENSE SWITCH) STATEMENT

IF (SENSE SWITCH i)n_1, n_2

where i is a one or two digit unsigned integer number or an integer expression, and n_1, n_2 are statement labels. If i is an integer expression, the low order two digits of the value of the expression are used as the value of i. The two digit numbers resulting from this are the numbers of machine indicators, not just console switches.

THE DO STATEMENT

DO n i = m_1, m_2, m_3

where n is a statement number, i is an unsigned integer variable which may be subscripted and m_1, m_2, m_3 are integer variables or integer expressions of any desired complexity, positive or negative. n may not be an alphabetic statement label, and i may not be an expression. There are no particular restrictions on m_1, m_2, m_3. In particular they may be positive or negative quantities. If $m_1 = m_2$, the DO will be executed once only. m_1, m_2, m_3 should be chosen so that the DO loop terminates. See below for an example of a never-ending DO-loop.

Example:

```
        DO 5J = K+L-5, M-I(JOB(KK)),-L
```

If m_1, m_2, m_3 are expressions, their values are the values of the expressions when the DO statement is encountered at object time, and these values are unaffected by alteration inside the DO of the values of the variables in the expressions m_1, m_2, m_3.

As a result of allowing positive or negative values for m_1, m_2, m_3, it is legal to have DO loops which count down. For example,

```
        DO 3 I = 10, 1,-1
```

will cause I to run from 10 to 1 in steps of (-1). The following is also permitted.

```
        DO 10 J = -10,5,2
```

which will cause J to assume successively the values -10, -8, -6, -4, -2, 0, 2, 4. If the DO variable assumes zero or negative values, it may be used, with caution, as a subscript. Intelligent use of negative or zero subscripts demands knowledge of the layout of data areas in memory, as described in the operating instructions.

Care should be taken to see that the DO index terminates properly. For instance,

```
        DO 20 K = -10, -1, -2
```

will increment nearly 5000 times before termination. The same is true of

```
        DO 40 K = 10, 1, 2
```

Termination in both cases occurs because integer arithmetic is performed modulo 10000.

All the restrictions on DO statements currently imposed by IBM FORTRAN II are also in force in KINGSTON FORTRAN, except as already mentioned.

CONTINUE STATEMENT

Same as IBM FORTRAN II.

PAUSE STATEMENT

PAUSE

PAUSE n, where n is a fixed point constant, variable or expression.

The typewriter types PAUSE n, together with error messages (see operating instructions) and the machine halts. If n is a variable or expression, its current value is typed. PAUSE (without n) generates an in-line halt command; there is no typing. In either case, depression START will cause resumption of program.

STOP STATEMENT

```
STOP

STOP n, where n is a fixed point constant, variable
        or expression.
```

The typewriter will type STOP, followed by the current value of n. If n is not specified, STOP 0000 will be typed. CALL EXIT is then executed (see operating instructions).

END STATEMENT

END is an instruction to the compiler that the program is complete. An END statement must be physically the last card of the main line program and of each sub-program which is associated with the job. The END statement results in CALL EXIT except in a sub-program, where it is interpreted as a RETURN statement.

FUNCTION AND SUBPROGRAM STATEMENTS

FUNCTION and SUBPROGRAM statements are the same in KINGSTON FORTRAN as in IBM 1620 FORTRAN II, and the same restrictions apply.

Because the compiler is one-pass, the subprograms are not compiled separately from the main program. See the operating instructions for further details.

INPUT/OUTPUT STATEMENTS

The INPUT/OUTPUT statements in KINGSTON FORTRAN II are similar to those of IBM FORTRAN II, except that expressions are permitted, as well as simple variables, in certain places in INPUT/OUTPUT lists. Indexed lists, array names (to handle a whole array) and all other standard FORTRAN II features are allowed. It is not necessary to specify a FORMAT statement number in an I/O statement. If no FORMAT statement number is given, the system will supply FORMAT (5N). See the description of FORMAT for an explanation of FORMAT (5N).

The permitted INPUT/OUTPUT statements are:

READ (card input), ACCEPT TAPE, ACCEPT (input on console typewriter), REREAD (re-reads last input record), PUNCH, PUNCH TAPE, TYPE (console typewriter), PRINT (on-line printer).

Indexed I/O Lists

As in IBM FORTRAN II, the statement

```
READ 10, ((A)I,J), I=1,10), J=1,10)
```

will cause 100 numbers A(1,1) to A(10,10) to be read into array A. Similarly,

```
READ 10,((A(I,J), I=K,L),  J=M,N)
```

will cause various elements of A to be read in under the control of the indices I and J.

In KINGSTON FORTRAN II, the limits on the implied DO's (I=K,L; J=M,N) may be expressions. Furthermore, the names of the input variables may be subscripted to any desired depth. For example:

```
READ 10,(((A(I(K1), J(M1)), K1=K-JOB*2,L+5-J6),M1=M*8-MM9,N-3*N18)
```

will be executed as

```
      DO   100 M1 = M*8-MM9, N-3*N18

      DO   100 K1 = K-JOB*2,L+5-J6

  100 READ 10, A(I(K1), J(M1))
```

where I and J are names of one-dimensional arrays which must previously have been defined.

KINGSTON FORTRAN II permits the same kinds of expressions in indexing as are permitted in standard DO statements. The implied DO in and I/O list may run forward or backward, and may have integer expressions of any desired complexity.

INPUT LISTS

In an input list, the variables may be only simple variables or indexed variables. Input of expressions is meaningless, and not permitted. For example:

```
READ 10, M, Q, A(I(K+4*L), M(N-5*L+4)),B
```

is permitted, provided I, K, L, N and M are previously defined.

READ 10, A+B-C(K) is not permitted.

OUTPUT LISTS

Output lists may be fully indexed lists, as described above. In addition, expressions may appear in the list as output quantities. For example:

```
PUNCH 20, C*D/(LOGF(X-Y*Z)+10.3), Y, D
```

will cause

```
C*D/LOGF(X-Y*Z)+10.3
```

to be calculated at the time the punch statement is encountered and its value to be punched, together with the values of Y and D, on a card, according to Format statement 20. The value of the expression in an output list is lost when it is output, and is not available for further calculation. The expression in an I/O list may be of any desired complexity, and may be indexed as required, either by DO statements, or by implied DO statements in the list itself. For example:

```
PUNCH 20,(((C*SQRTF(A(I,J))-M(I)),I=1,L+4,3),J=I+1,K-10,5)
```

will cause values of C*SQRTF(A(I,J))-M(I)

to be punched out for values of J from I+1 to K-10 in steps of 5 and values of I from 1 to L+4 in steps of 3.

ASSIGNED FORMAT NUMBERS

Format statement numbers may be assigned by ASSIGN statements in the same way any other statement number can. Hence, input/output statements may use alphabetic statement labels in place of Format statement numbers. For example, the following program is permitted:

```
3    FORMAT (5(I3,F10.5))
4    FORMAT (5I5)
5    FORMAT (5I7)
     ASSIGN 3 TO J
     ASSIGN 4 TO K(1)
     ASSIGN 5 TO K(2)
     READ J, (M(I),A(I), I=1,5)
     DO 10 L=1,2
10   READ K(L), (M4(I), I=1,5)
```

Note that the first statement will be executed according to Format statement 3, while the second READ statement will be executed according to Format Statement 4 when L=1, and according to Format Statement 5 when L=2.

The subscripted variables in all the above examples must previously have been mentioned in a DIMENSION statement.

ARRAY NAMES IN I/O LIST

As in IBM FORTRAN II, array names without subscripts may appear in I/O lists. Mention of an array name will cause the entire array, as specified in the DIMENSION statement to be input or output. Two dimensional arrays are handled column-wise -

```
DIMENSION A(10,10)
READ, A
```

will cause the entire 100 elements of A to be read in, in 5N notation. The elements of A must be in order A(1,1), A(2,1), A(3,1), A(4,1), A(5,1), A(6,1), etc.

FORMAT STATEMENTS

Format statements are, in general, equivalent to Format statements allowed in 7090/94 FORTRAN II. E, F, I and A conversion are permitted. Repetition of field format is allowed before E, F,I or A. Thus FORMAT (I2,3E12.4) is equivalent to

```
FORMAT (I2,E12.4,E12.4,E12.4)
```

Parenthetical expression is permitted in order to enable repetition of data fields according to certain Format specifications within a longer FORMAT statement. The number of repetitions is limited to 99. Thus,

```
FORMAT (2(F10.6,E10.2),I4)
```

The level of parenthesizing can be extended to a second level, thus:

```
FORMAT (2(I4,2(F6.2,F8.3))) is equivalent to

FORMAT (I4,F6.2,F8.3,F6.2,F8.3,I4,F6.2,F8.3,F6.2,F8.3)
```

The depth of such nesting of parentheses must not exceed 5, which appears to be more than would ever be necessary.

N-Format

Rigid format on input data is not always desirable, and in many cases makes key-punching more difficult. KINGSTON FORTRAN allows so-called "free form" input, as well as the more familiar fixed or rigid format. If the FORMAT statement specifies I, E or F format on input, then the input data record must conform to the normal rules for such format as specified in IBM manuals. However, if N format (denoting "free form") is used, the data numbers may appear anywhere on the card, and input is controlled by the input list.

N format is used like E, F or I format except that no width or decimal point location digits are required or permitted. For example,

```
READ 10, I, J, A, C, Z
10 FORMAT (5N)
```

will cause the program to read in a record of 2 integer numbers followed by 3 floating-point numbers. In N format, a number is defined as: any number of leading blanks, followed by a meaningful collection of digits, followed by 1 trailing blank. Note that the blank column immediately following the right-most digit or character of the number is considered part of the number, and serves to delineate the right-hand end of the number.

In the case of E numbers handled with N-format, blanks after the letter E are ignored, and the machine uses the next set of digits as the exponent. For example:

b1.2345678E-05b

will be interpreted as .000012345678.

The number b1.2345678Ebbbbb-05b

will be interpreted in the same way.

b1.2345678Ebbbb103

will result in an error condition (see operating instructions).

b1.2345678E bb 00005

will be interpreted as 123456.78. Leading zeros before either the mantissa or exponent are ignored.

An E- type number handled by N-format ends with the blank after the exponent digits.

A FORMAT statement may specify N, E, F, I or A format as required, thus allowing both free and rigid format on the same card. Note that, in N format, if a floating point number does not have a decimal point, it is assumed to be after the low-order digit of the number.

Some examples may help:

```
        READ 10, I, J, A, C, Z

10      FORMAT (5N)
```

The card might look like:

bb123bbbbbb12bbb16.3bbbbb1.2E6b123000bbb etc.

N Format requires only that at least 1 blank column follow the number. In this case, I, J, A, C, Z would be stored as 123, 12, 16.3, 1.2E06, 123000. resp.

```
        READ 11, I, J, A, C, Z

11      FORMAT (I3, I6, N, F10.3, N)
```

The Format requires that I, J, C follow rigid format. The card might look like:

b12bbb12bbbbbb120.↓bbbb1234567bbb16.8bbb etc.

This would give the following results:

Variable	Value
I	12
J	120
A	120.
C	1234.567
Z	16.8

Note that the F-specification for C starts on the first column after the blank following 120., (see the position of the arrow) since this blank is considered part of the value of an N-Format number.

An output, N format is equivalent to 1PE14.7,1X for floating point numbers, and I5,1X for integer numbers.

N Format allows repeated format and parenthesizing, and follows the usual rules for them.

If a number is positive, the output under E, F, I or N Format will not contain a leading plus sign. On I Format, no space is left for it, so that it is possible to construct a fully packed output record provided all numbers are positive. N Format generates a space for a + sign and a space following the number.

If a floating point number is output under Iw Format, the integer part of the floating point number is convered to Iw Format. Thus 128342.56 output with I10 Format would appear as bbbb128342.

SCALE FACTORS

To permit more general use of E and F conversion, a scale factor followed by the letter P may precede the specification. The scale factor is defined such that

$$\text{Output number} = \text{internal number} \times 10^{\text{scale factor}}$$

$$\text{Internal number} = \text{input number} \times 10^{-\text{scale factor}}$$

This operates exactly the same as in IBM FORTRAN II for the larger machines. For example

```
FORMAT (2PF10.4)
```

used on output will multiply the number by 100 before output. On input, it will divide the external number by 100 before storing it in the machine.

On E-Format output, the effect of P-scaling is to shift the decimal point in the mantissa and to adjust the exponent by the amount of the shift.

Thus, if FORMAT(E15.8), used for output, produced the number .12345678E-04, then FORMAT (3PE15.5) would produce 123.45678E-07 for the same number. Note that for E-Format output, P-scaling does not change the magnitude of the number. It shifts the decimal point, and makes a compensating change in the exponent. For F-Format, P-scaling alters the magnitude of the number on input/output.

VARIABLE FORMAT

KINGSTON FORTRAN II allows variable Format. That is, Format specifications may be read in at object time. In this way, data may be read in under control of a Format Statement which itself has been read in. Variable Format statements must be read under A-Format into an array by means of a normal Read statement.

For example:

```
      DIMENSION FMT (15)

      READ 10, (FMT(I), I=1,14)

10    FORMAT (15A5)
```

will cause 70 characters of input record (i.e. the Format Statement being read in) to be stored in array FMT. It is then possible to write:

```
      READ FMT, A, B, X, Z, (A(J),J=1,10)
```

where the input variables will be read in according to the Format Statement stored in array FMT.

It is also possible to alter array FMT by programming. This should be done with some care, otherwise the Format Statement stored in array FMT may become completely unintelligible.

The name of the variable Format specification must appear in a DIMENSION Statement, even if the Array size is only 1.

The Format read in at object time must take the same form as a source program Format Statement except that the word Format is omitted, i.e. the variable Format begins with a left parenthesis.

SPECIFICATION STATEMENTS

COMMON

Variables, including array names, appearing in COMMON statements will be assigned core storage locations beginning at the high end of memory, and will be stored at object time in descending sequence, 10 digits per variable, or per item of a dimensioned variable, as they are encountered in the COMMON statement. If a variable is a dimensioned variable, the size of the dimensioned array must appear in the COMMON statement, and the variable must not again be dimensioned in a DIMENSION statement. The COMMON statement must precede EQUIVALENCE or DIMENSION statements (if any) and must precede the first statement of the source program. For example:

```
      COMMON A,B,I,J,X(10,3),Y(5)
```

(Inclusion of dimensioning information in COMMON statements is allowed in FORTRAN IV).

DIMENSION

The DIMENSION statement is the same as IBM FORTRAN II except that variables already mentioned in COMMON may not again be dimensioned and that only 2 subscripts are allowed.

```
DIMENSION Z(10,5),V(400) is permitted

DIMENSION X(10,5,10) is not permitted
```

EQUIVALENCE

```
EQUIVALENCE (a,b,c,---), (d,e,f,--),---
```

where a,b,c,d,e,f, are variable names. KINGSTON FORTRAN imposes some restrictions on EQUIVALENCE statements which are not present in IBM FORTRAN II. These are noted below:

1. Single variables may be equivalenced only to single variables.
2. Arrays may be equivalenced to other arrays, of the same size only.
3. Single variables may not be equivalenced to individual items of arrays, nor may single items of two arrays be equivalenced. In general, no subscripts may appear in an Equivalence statement.
4. Because the compiler is single pass, it is crucial that the order in the source deck be:

   COMMON (if any), DIMENSION(if any), EQUIVALENCE (if any).

   They must precede the first executable statement of the program.

5. If arrays are to be equivalenced, the first item only in the list must have been defined previously in a COMMON, or DIMENSION declaration, and the remaining items in the list must not have been so defined. The Equivalence statement itself defines these remaining items. If single variables are to be equivalenced, and any item in the Equivalence list has been defined in a previous COMMON or TYPE statement, it must be first in the Equivalence list, and the other items must not have been defined in a COMMON or TYPE statement. For example,

   ```
   COMMON A,B(10,3),C
   DIMENSION D(50)
   EQUIVALENCE (A,F,G),(D,X)
   ```

   This puts A, array B, and C in common storage; defines array D; defines F and G as single variables in the same memory location as A; and defines X as a 50-item vector in the same location as D. The following are errors: (in the example above).

```
EQUIVALENCE (D,A)          (para.1,2)
EQUIVALENCE (B(1,1),G)     (para. 3)
EQUIVALENCE (X,D)          (para.5, X not defined)
EQUIVALENCE (G,A,F)        (para.5, G not defined,
                             A defined)
EQUIVALENCE (D(50),X(50))  (para.3)
```

6. To preserve compatibility with other FORTRAN systems, which require DIMENSION statements for all array variables in an Equivalence list, KINGSTON FORTRAN allows extra DIMENSION statements after the Equivalence statements. Such DIMENSION statements may be used to mention the equivalenced variables, but since they have already been defined in the Equivalence Statement, the compiler will ignore them. It will not, however, call them errors. For example:

   ```
   DIMENSION X(10), Y(20)
   EQUIVALENCE (X,A,B), (Y,C,G)
   DIMENSION A(10), B(10), C(20), G(20)
   ```

   is permitted. The variables A,B,C,G in the second DIMENSION statement are ignored by the compiler, because they have already been defined in the preceding EQUIVALENCE Statement.

7. It is possible to equivalence items not of the same type or mode: e.g. EQUIVALENCE (A,I) - where A is real and I is integer.

TYPE

Two TYPE declarations are permitted. These statements determine the type of variable associated with each variable name appearing in the statement. This TYPE declaration is in effect throughout the program. The two declarations are

```
INTEGER a,b,c,....

REAL a,b,c,....
```

where a,b,c, are variable names appearing within the program. Function names may not appear in TYPE declarations.

Rules:-

(1) A variable defined to be of a given type remains of that type throughout the program.

(2) INTEGER indicates that the variables listed are integer, and over-rides the alphabetic naming convention.

(3) REAL indicates that the variables listed are floating point, and over-rides the alphabetic naming convention.

The TYPE declaration must occur before the first executable statement of the program. If any of the variables mentioned in a TYPE declaration are mentioned in a COMMON or DIMENSION statement, the TYPE declaration must follow such mention.

If a TYPE declaration precedes an EQUIVALENCE statement, then it defines a variable in the sense required by the EQUIVALENCE statement, and all variables equivalenced to the one declared in the TYPE statement will be of the same type.

If a TYPE declaration follows an EQUIVALENCE statement, then only the specific variable names mentioned in the declaration will be affected.

Examples,

```
1.   INTEGER A
     EQUIVALENCE (A,B,C)

2.   EQUIVALENCE (A,B,C)
     INTEGER A

3.   EQUIVALENCE (A,B,C)
     INTEGER A,B,C

4.   INTEGER A,B,C
     EQUIVALENCE (A,B,C)
```

Examples 1 and 3 cause A,B,C, to be integer variables and occupy the same memory location.

Example 2 causes A to be integer, B,C to be real, and A,B,C to occupy the same memory location.

Example 4 is an error in KINGSTON FORTRAN (see para. 5 under EQUIVALENCE).

KINGSTON FORTRAN II

OPERATING CONCEPTS AND SUBROUTINE DECK DESCRIPTIONS

by:

J.A.A. Field,[1] D.A. Jardine,[2] E.S. Lee,[1]

J.A.N. Lee,[3] and D.G. Robinson[2]

Presented at the Joint Canadian-Midwest Region
Meeting of the 1620 Users Group, Chicago,
February 19-21, 1964

[1] Dept. of Electrical Engineering, University or Toronto, Toronto, Ontario.

[2] Research Centre, Du Pont of Canada Limited, Kingston, Ontario

[3] Computing Centre, Queen's University, Kingston, Ontario

KINGSTON FORTRAN II

OPERATING CONCEPTS

KINGSTON FORTRAN has incorporated in it the ability to recognize certain control cards both at compile and object time. The control cards recognized by the compiler are, with one exception,instructions to the compiler to execute various options such as symbol table output, compile with or without trace, etc. A list of these and their functions appears later. The control cards recognized by the object program are intended to help in the operation of programs involving blocks of data and to permit continuous flow of programs through the machine with a minimum of operator intervention. The system will allow stacking of programs in the read hopper and execution of these programs, in the order they are presented to the machine, without requiring button pushing at each program load.

COMPILER OPERATION

The compiler deck is self-loading and self-identifying. To load the compiler, push RESET, LOAD. The switch settings are:

Parity	- STOP
I/O	- Program
Sense Switches	- not used. Position immaterial.

Because the 1620 typewriter is prone to write-checks, any errors in its operation are completely ignored. Card I/O read- and write-checks are handled by programming.

The source deck is assembled with the main-line program accompanied by all subprograms in source language. The main-line and subprograms may be in any order. Because the compiler is one pass, and to avoid the complications of subprogram object loaders, the entire deck is compiled at one time. The job size must be such that the main program and all its subprograms can be accommodated in core at one time. That is, no overlay of subprograms by other subprograms is permitted. This restriction also exists in IBM 1620 FORTRAN II.

The end of the mainline program and of each subprogram must be indicated by an END statement. Thus a program may contain more than one END statement. A special control card is used to indicate the end of the entire job; a job, in this context, means the set of main-line program and all required subprograms.

The following section gives the compiler control cards and their function. All control cards must have a $̸, *, or ‡ symbol in column 1, and the identifying word in cols. 7 and following.

BEGIN TRACE

Form: Col 1 - $̸,*, or ‡
Col 7 - BEGIN TRACE

Location in Deck: anywhere

Function : Trace instructions are compiled, beginning with the next arithmetic statement. Tracing generates no additional instructions.

END TRACE

Form: Col 1 - $̸, *, or ‡
Col 7 - END TRACE

Location in Deck: anywhere

Function : If trace instructions were being compiled, this card stops compilation of trace instructions. If trace instructions were not being compiled, this card has no effect.

LIST

Form: Col 1 - $̸, *, or ‡
Col 7 - LIST

Location in Deck : anywhere

Function: The typewriter starts typing the object time location of the first machine language instruction of each source statement. The source statement itself is not typed. The typed locations can be matched with a 407 off-line listing of the source program, if desired.

UNLIST

Form: Col 1 - $, *, or ǂ
Col 7 - UNLIST

Location in Deck: anywhere

Function: If the typewriter had previously been typing locations as a result of a LIST card, the UNLIST card stops it. Otherwise, the UNLIST card has no effect.

MAP

Form: Col 1 - $, *, or ǂ
Col 7 - MAP

Location in deck: anywhere before any END statement.

Function: The symbol table for the main program or subprogram (depending on which END statement is currently being processed) is punched on cards when the END statement is encountered, provided a MAP card occurred previously in that section of the job. A separate MAP card is required for each section of the job for which a symbol table is wanted.

JOB

Form: Col 1 - $, *, or ǂ
Col 7-9 - JOB
Col 10-79- any valid information

Location in deck: The JOB card must be the first card of any source deck.

Function: The JOB card informs the compiler that what follows is a FORTRAN source program. The compiler will not recognize a source program until a JOB card is found, and will read cards indefinitely until it finds one. The JOB card is reproduced (from column 7 onwards) into the object deck so that the object deck is self-identifying when it is loaded.

END OF JOB

Form: Col 1 - $, *, or ǂ
Col 7 - EOJ

Location in Deck: The EOJ card must be the last card of any source deck, i.e. must be the last card of the job. It is in fact, the super END card.

Function: The EOJ card informs the compiler that the end of the source deck has been reached. The machine will stop, allowing removal of the object deck. Pressing start will cause the compiler to read cards searching for a JOB card or a LOAD card (q.v.).

LOAD

Form: Col 1 - $, *, or ǂ
Col 7 - LOAD

Location in Deck: following the last EOJ card of the last source deck.

Function: Because this is a batch compiler, a control card is needed to inform the compiler that what follows is not a source deck, but rather a new program to be loaded. When the compiler finds a LOAD card, it executes a 1620 load operation on the card immediately following, on the assumption that it is the first card of a self-loading program. If it is not, you will be in trouble.

PRESCAN

Form: Col 1 - $, *, or ǂ
Col 7 - PRESCAN

Location in Deck: anywhere

Function: Inhibits punching of object deck. Error cards are punched if errors are found. A PRESCAN card may be used in place of a JOB card.

SIZE

Form: Col 1 - $, *, or ‡

Col 7 - SIZE NNNN9

Location in Deck: Immediately following JOB card, i.e. 2nd card of source deck.

Function: The SIZE card specifies the highest numbered core location which the object program is to occupy. NNNN are any 4 digits, but for instance would usually be 1999 if compiling for a 20K machine on a 40K machine. It, however, can be any 4 digits whatsoever. If the assignment of this highest memory location is such that the job will not fit, an overlap message will result.

ORIGIN

Form: Col 1 - $, *, or ‡

Col 7 - ORIGIN NNNNN

Location in Deck: Immediately follows SIZE card if one exists. Otherwise it follows the JOB card.

Function: The ORIGIN card specifies the core location in which the first machine language instruction of the compiled program will be placed. NNNNN must be an even number. If not, you will have difficulties. By suitable choice of SIZE and ORIGIN, the object program can be put almost anywhere in core. In fact, it is possible to specify so little core for the object program that no source program whatsoever will fit in it.

If the origin is not specified by an ORIGIN card, the object program will start at location 5900. This is not quite as good as it looks, because, as is common with many computing systems, you may need extra bits and pieces to make things work. See the section on subroutines.

SUMMARY

To compile a program, load the compiler followed by the source deck. The source deck order is:

JOB card
Main-line program with END card)these may be
Subprogram(s) with END card(s))in any order
EOJ card.

If another source deck is to be compiled, make it up in the same way, and stack up in the reader hopper. If the next thing to be done is a self-loading program, precede it with a $LOAD card.

Symbol Table

If a MAP card occurred in the source deck, a symbol table will have been punched. Because a separate symbol table may be punched for the main program and each subprogram, it is not possible to avoid interspersing the object deck and the symbol table. For this reason, the symbol table cards are identified by a particular code on the card. The 407-E8 wiring diagram in this write-up will detect which are symbol table cards and print only those, ignoring object program cards. The symbol table is punched 4 symbols per card.

Whether or not a symbol table is punched, the compiler punches 1 blank card following completion of the job. This allows removal of the object deck without using the non-process runout feature on the 1622. The next deck to be punched will be preceded by the blank card, which must be discarded.

Error Checking

The KINGSTON FORTRAN compiler has built in provision for checking errors in the source program. Because of the expansions in the language, certain statements which are unacceptable to a normal FORTRAN compiler will, of course, be accepted by the KINGSTON FORTRAN compiler.

All errors will be punched on cards suitable for 407 listing using the panel described in this write-up. The 407 will ignore any object program cards in the deck. The error card will contain an error code followed by the line number in which it occurred. The errors and their codes are described in Table 1.

TABLE I

ERRORS AT COMPILE TIME

Error	Reason
Q1	Character after Format not (
Q2	No EOJ card
Q3	Continued Error
Q4	Argument List in Subroutine or Function Declaration not a simple variable.
Q5	Unpaired Parentheses.
Q6	No statement number in Format.
Q7	Unrecognizable.
Q8	Statement exceeds 300 characters.
Q9	Doubly defined St. No.
P1	Incorrect Go To Statement.
P2	Invalid Assign Statement.
P3	Invalid If Statement.
P4	Invalid Computed Go To Statement.
P5	EOJ Card not preceded by an End Card.
P6	Expression in Subr.
P7	Invalid Call.
WA	Illegal Operator in Expression is $ or @.
WB	Illegal sequence of operators.
WC	Mode Error.
WD	OP-VAR-OP Sequence Illegal; Syntax error in Statement.
WE	) not followed by an operator.
WF	Invalid operator in subscripting.
WH	Number of subscripts does not agree with DIMENSION Statement.
WG	Floating Subscript.
WJ	Expression Ends in Illegal Character.
WS	Invalid expression on left-hand side of Arithmetic Statement.
WP	Invalid expression on right-hand side of Arithmetic Statement.
WT	One of the tables used in compiling Arithmetic is full; i.e. Statement is too long.
WK	Syntax error in Arithmetic expression.

Table I (cont'd)

Error	Reason
R1	Incomplete DO or I/O Statement.
R2	Expression in Input List.
R3	Unpaired ( ) in Assigned Format No.
	Invalid Delimiter in I/O Statement.
	Invalid Use of ( ) in I/O Statement.
F1	Format too verbose for simple minded compiler or, (before completion of repeating format.
F2	Most likely, invalid format, DO, I/O, or Arithmetic Statement. If Format, can be: - sign that is not part of P-Type, incorrect specification of length of H type; no closing); statement not complete; non permissible character.
F3	More than 5 levels of repeating format Repeated Power Format has more than 49 repeats Field Width is missing in I, A, F, E, Specs A-Width greater than 50 D missing in EW.D or FW.D Decimal missing in EW.D or FW.D Non-permissible character. D greater than W in EW.D or FW.D (W-D) greater than 45 Field Width greater than 80. A-Type has zero field width.
ER99	Symbol is more than 6 characters.
ER98	Fixed point number has too many digits.
ER97	Floating point number too big.
ER96	Floating point number too small.
ER95	Symbol table full.
ER94	Symbol which should be a function is not.
EB93	Simple variable in Dimension Statement.
ER92	Dimension IMAX not followed by) or ,
ER91	Missing) on Dimension Variable.
ER90	No , between Dimension or Common items.
ER89	Unidentified Card.
ER88	First item in Equiv List not in Table.
ER87	Missing or , in Equivalence.
ER86	Number in Equiv Statement.
ER85	Variable Dimensioned Twice.
ER84	Arith. St. Func. Defined Twice.

OBJECT PROGRAM OPERATION

1. Introduction

The permanent subroutines package contains routines which facilitate the handling of multi-part programs and the handling of multiple data sets for the same program. The routines also have the ability to recognize certain control cards as described below. This is not by any means a resident monitor, but it uses some of the simpler concepts involved in monitor systems.

An object program will operate perfectly satisfactorily without referring to the resident supervisor program. If this kind of operation is desired, then the running of the object program is the same as for any other card 1620 FORTRAN. Load the object program, followed by the subroutine deck, followed by the data, and pray.

2. Error Messages at Object Time

No method really satisfactory to all people can be devised for handling errors at object time. Some people want every error, however trivial, brought to their attention every time it happens, either by typewriter message or by stopping the machine. Others say that any error should result in passing control to the monitor and delivering to the programmer a core dump (preferably in binary) together with a cryptic indication as to the possible source of his trouble. Still others assert that no errors should be detected at all, that the machine should run merrily on and that it is up to the programmer to figure out post facto why his answers are out by a factor of 10^{35}.

The position taken in KINGSTON FORTRAN is that a 40K 1620 is a little too expensive to permit unbridled chattering by the typewriter, but is still cheap enough to permit some stopping during the course of debugging a program.

Object time errors are collected in an 18 digit error field located in the permanent subroutines. Digits are inserted in this field to indicate various kinds of errors, and system CALL statements have been included to allow interrogation, typing, and resetting of this error field.

Most errors do not result in stopping the machine, and the error is not communicated when it occurs.

The error field is also typed out when a PAUSE, STOP or END statement is encountered. The error codes are given in Table 2. If a check digit is zero, the error in question did not occur.

The systems CALL statements for interrogating and using the error field, follow:

CALL EPRT

If the error field contains one or more non-zero digits (i.e. at least one error has occurred) the typewriter types the 18 digit field followed by CHECK. If the error field was zero throughout, only the word CHECK is printed. The error field is not reset to zero by CALL EPRT. Control is passed to the next executable statement of the program.

CALL RESET

The error field is reset to zero. It is not typed out. Control is passed to the next executable statement of the program.

CALL ERRCK(J)

The error field is interrogated. If it is non-zero (at least one error has occurred) the integer variable J is set equal to 1. If no errors have occurred, J is set equal to 2. The error field is printed out (if non-zero) and reset to zero. Control is passed to the next executable statement of the program.

The error field is also typed out by certain supervisor control cards, as described in the next section.

THE SUPERVISOR

The resident supervisor can recognize 3 kinds of control cards. One of these signals that the following card is the first card of a new job and that a load operation is called for. The other two are used to delineate blocks and files of data for a given program.

New Program Card

Form : Col 1 $
Col 2-80 any alpha numeric information

Location: first card of an object program deck.

Function: This card informs the object program that a new job is waiting to be loaded. If the current object program reads such a card under the misapprehension that it is a data card, the words END OF DATA are typed followed by the word CHECK and the error field (if non-zero). The typewriter then types out the contents of the card, and the computer simulates the load operation to read in the next job.

End of Block

Form : Col 1,2 - $$
Col 3-80- any alphanumeric information.

Location: At the end of a block of data.

Function: When a card containing $$ is read under control of a READ statement, the End of Block Indicator is turned on, and the typewriter types the contents of the End of Block card, followed by the word CHECK and the error field (if non-zero). Control is then transferred to the first executable statement of the program.

The End of Block Indicator may be interrogated by calling the End of File or Block subprogram. See below. (The End of Block Indicator is a program switch, not a hardware feature).

End of File

Form: Col 1,2,3 $$$
Col 4-80 any alphanumeric information

Location : at the end of a set of blocks of data

Function: When a card containing $$$ is read under control of a READ statement, the End of File Indicator is turned on, and the typewriter types the contents of the End of File card, followed by the word CHECK and the error field (if non-zero). Control is then transferred to the first executable statement of the program.

The End of File Indicator may be interrogated by calling the End of File or Block Subprogram. See below. (The End of File Indicator is a program switch, not a hardware feature).

The End of File or Block Subroutine Subprogram (which is built into the system) may be used to interrogate the End of File and End of Block Switches.

```
CALL EOFB(J)
```

The End of File and End of Block indicators are interrogated.

If the End of File Indicator is on, J is set equal to 1.

If the End of Block Indicator is on, J is set equal to 2. If neither is on, J is set equal to 3. Both indicators are set to the OFF position after interrogation. Control is transferred to the next executable statement of the program.

Note on the use of End-of-File, End-of-Block

In a job which is processing data in batches, it is convenient to have some way of telling the computer where the end of a data set is, and also to tell the machine which is the last set of such data.

The end of a set of data is called a "block" in our nomenclature. It may be of variable number of data points (as in many statistical problems), but at least it is the amount of data which is appropriate for the whole job or for a section of it.

Many jobs are set up to process several blocks of data in more or less the same way for each block. It is useful, however, to identify the end of the last block so that the program is informed that no more data exist. The End of File Indicator accomplishes this. In our nomenclature, a "file" is a set of one or more "blocks" of data.

Since reading an End of Block or End of File card returns control to the first executable statement of the program, it is suggested that this first statement should be

```
CALL EOFB(J)
```

followed at a suitable place by a computed GO TO using J as its index.

Two other system subroutine call statements are provided in KINGSTON FORTRAN:

```
CALL EXIT
```

When this subroutine call is encountered the object program is stopped and control is passed to the supervisor. The machine will read cards until it finds a new job card. When this is found, the number of cards read before finding the new program card is typed out as BYPASS N where N is the number of cards. The error check field is typed and the new program card is handled in the normal way.

CALL SKIP

This subroutine call causes interruption of the normal program. The machine will read cards until the next end-of-block or end-of-file ($$ or $$$) card is encountered, at which time control is transferred to the first statement of the program. If a new job card is encountered before at $$ or $$$ card, a normal exit to new program will result. In any case, the check field is typed, together with BYPASS N as explained above under CALL EXIT.

CALL SKIP will usually be employed to stop calculation on a block of data because of an abnormal situation (e.g. failure to converge on an iteration, bad data) which has occurred in the block of data. In such a case, CALL SKIP will cause that particular calculation to be abandoned, and a new set of data to be presented to the program.

CALL EXIT and CALL SKIP may also result from certain object time error conditions. See Table 2.

Certain input-output errors are also detected at object time. If one of these is encountered, the typewriter will type the words I/O ERROR, followed by a digit. A list of these errors is shown in Table III.

TABLE 2

Object Time Errors

Position in Error Field	Digit	Meaning	Action Taken (FAC = Accumulator = Result Field)
1st digit	1	Floating Point Underflow	FAC = 0000000000
2nd	2	Floating Point Overflow	FAC = ±9999999999
3rd	3	Floating Point Divide by Zero	FAC = ±9999999999
4th	4	Fixed Point Divide by Zero	FAC is unchanged, i.e. J/0 ≡ J
5th	5	Square Root of Neg. Number	Square root of absolute value of arg.
6th	6	Log of zero or Neg.Number	Log(0) ≡ -9999999999; otherwise log of abs. value of arg.
7th	7	Sin or cos, arg. > 10^8	CALL EXIT
8th	8	Exp(x) out of range	FAC = ±9999999999
9th	9	Input number too small	The number entered memory as 0000000000
10th	1	Input number too big	The number entered memory as ±9999999999
11th	2	Unused. Available for user-defined relocatable subprograms.	
12	3		
13	4		
14	5		
15	6		
16	7		
17	8		
18	9		

TABLE 3

I/O Errors at Object Time

I/O Error	Reason	Result
0	Input record from T/W or paper tape over 120 characters long	CALL EXIT
1	Non-alphabetic data on A-type output	CALL EXIT
2	Field Width too small on I, E, F, output	CALL SKIP
3	Invalid character on input data on I, E, F, or N Format	CALL SKIP
4	Read in integer with E, F, or N Format and has lost right-hand end digits	CALL SKIP
5	Input-Output list with no numeric specifications between last opening-closing parenthesis pair in Format statement	CALL EXIT
6	Format requires more than 120 characters in a record	CALL EXIT
7	Write-check occurred 3 times when attempting to punch output or trace card	CALL EXIT
1F	Error in Variable Format - similar to error F1 at compile time	CALL SKIP
2F	Ditto - similar to error F2 at compile time	CALL SKIP
3F	Ditto - similar to error F3 at compile time	CALL SKIP
READ 1	Read check on T/W	Computer halts. When start is pressed, the machine will attempt to read the record again
READ 3	" " " paper tape	
READ 5	" " " cards	

MEMORY ALLOCATION AT OBJECT TIME

All constants and variables are stored in 10 digit words. The address of the low order digit ends always in 9. Hence the address of the high order digit ends in 0.

SIMPLE VARIABLES

Real Variables - 10 digits, low order digit address ends in 9.

Integer Variables - 4 digits, low order digit address ends in 9.

Real Constants - same as real variables.

Integer Constants - In the rare cases that fixed point constants are stored in the object program, both the negative and positive value of the constant are stored. The positive value occupies the low order 4 digits of the 10 digit word; the negative value has its low-order digit address ending in 4. The other 2 cores are unused.

address

e.g. 0 1 2 3 4 5 6 7 8 9

0 $\bar{5}$ 6 7 $\bar{8}$ 0 $\bar{5}$ 6 7 8

This illustrates storage of 5678

A-Format Words - are stored in a 10 digit field, the low order address of which ends in 9. The digit at the high order address (ending in 0) is flagged.

ARRAYS

Vectors- one-dimensional arrays. Vectors are stored starting with the first element at the highest numbered address, and with succeeding elements at progressively lower numbered addresses. That is, the vector dimensioned A(10) would be stored A(1),A(2),A(3),--A(10) at successively lower memory locations. The address of element A(I) may be calculated from:

Address of A(I) = Address of A(0)-10*I

where the address of A(0) is called the base address of A.

Note that A(0) = A(1) + 10

Zero and negative subscripts will perform properly on a vector provided space is available.

Matrices - two dimensional arrays. Matrices are stored starting with the first element, B(1,1), stored at the highest numbered address. The elements are stored column-wise at progressively lower numbered addresses: B(1,1),B(2,1),B(3,1) etc. The address of B)I,J) may be calculated from:

Address of B(I,J) = Address of B(0,0)-10 (J*IMAX+I)

where the address of B(0,0) is called the base of B. IMAX is the maximum number of rows in B as specified in the DIMENSION statement.

Note that:

The address of B(0,0)=address of B(1,1)+10 (IMAX+1)

Negative or zero subscripts on a matrix will work properly on the second subscript, but not on the first subscript. For instance, if B is dimensioned B(3,3), then B(3,1) and B(0,2) will be in the same memory location, a condition which may be undesirable. However, B(2,0) will be stored in the second item before B(1,1).

An example of a memory layout may help. Suppose the program has the following COMMON statement:

COMMON X, A(4), B(2,3)

The layout of memory is:

Variable	Memory Location (low order digit)
X	39999 (Base of A)
A(1)	89
A(2)	79 (Base of B)
A(3)	69
A(4)	59
B(1,1)	49
B(2,1)	39
B(1,2)	27
B(2,2)	19
B(1,3)	09
B(2,3)	39899

STATEMENT NUMBERS

The address of the statement is stored in a 5 digit field, which is referred to indirectly. Two such 5 digit fields are contained in a 10 digit word, whose low order address ends in 9.

SUBPROGRAM ADDRESSES

FORTRAN subprograms or arithmetic statement functions require two 5 digit addresses for their entry points. These are stored in a 10 digit word whose low order address ends in 9.

TEMPORARY ACCUMULATORS

(1) 10-digit accumulators may be required during the evaluation of an arithmetic expression. These are treated exactly like storage for simple variables.

(2) 5-digit accumulators are used in subscripting calculations. Two of these are stored per 10 digit word.

THE SUBROUTINE DECK

No provision is made for reproducing the subroutine deck into the object program. It is the opinion of the writers of KINGSTON FORTRAN II that the 1620 should not be used to reproduce subroutine decks indiscriminately. For that reason it is required that the subroutine deck be placed behind the object deck when loading. If a condensed program is desired, use a suitable core-dump-and-reload program.

The subroutine deck consists of 3 parts, which are, in order: (a) the relocator, which handles the loading of the relocatable subprogram (as requested by the object program) into core storage; (b) the relocatable subprograms which consist of the library function subprograms(sin, cos, exp, etc.), parts of the input-output subroutines and any subroutine subprograms which the user may wish to write; (c) the permanent subroutines containing the programmed floating-point arithmetic routines, the fixed point routines, the supervisor, the trace routine, and a hard cord of the input-output routines.

These three sections are essentially independent. The relocator is a completely separate program which uses information contributed by the object deck (memory size, subprograms desired, where empty space is available for those subprograms, etc.) to select and load into core storage the relocatable subprograms needed for the job.

In a small machine, like the 1620, it is essential to conserve memory space. For this reason, the input-output routines have been broken up into pieces, and are treated like any other relocatable routines. For instance, if the object program does not use A-specification, the routine to handle this will not be loaded into core. Thus, only the input-output routines needed by the object program will be loaded.

The question arises, how far should this be carried. About 2000 cores of input-output routine are used by all other input-output routines, and these are part of the permanent subroutines package. The routines for exponentiation, floating and integer, and for integer division were made relocatable. All others are part of the permanent package. Some consideration was given to making the trace routine (300 cores) relocatable. This was rejected on the grounds that any program which fits into the machine should also be traceable. The only way this can be assured is to have the trace routine permanently in place.

The permanent subroutines are a self-contained deck independent of the relocator. This deck is so programmed that after loading, control is transferred to the core location containing the first machine language instruction of the object program.

A list of the relocatable subprograms is given in Table II.

RELOCATABLE SUBPROGRAMS

User-defined-relocatable-subprograms (abbreviated UDRS) may be of several types, depending on the coding generated by the compiler when the subprogram name is encountered in a FORTRAN source statement. The generated coding is controlled by the makeup of cards placed at the end of the compiler deck. These trailer cards inform the compiler what subprograms are available and also supply auxiliary information about them.

Each entry on a trailer card consists of a two digit subprogram number, followed by the subprogram name (six characters maximum) followed by a 3 digit code number enclosed in parentheses. A typical entry has the form:

NNXXXXXX($n_1n_2n_3$)

where NN is a two digit number unflagged).

XXXXXX is a 1 to 6 character name

$n_1n_2n_3$ is a 3 digit code which describes the subprogram properties to the compiler

NN is any two digit number between 01 and 66.

XXXXXX is any name beginning with a letter. It does not have to end in F and its starting letter is independent of the function mode; e.g. integer functions do not have to begin with I, J, K, L, M, or N.

$n_1n_2n_3$ is made up as follows: n_1n_2 form a two digit number controlling the coding generated by the compiler; n_3 describes the function properties. Table I describes this.

SVECT is a location in a subprogram transfer vector located in the permanent subroutines. This vector contains the relocated address of the subprogram, as explained below.

TABLE I

Digits n_1n_2	Coding Generated		Use and Notes
10	TF BTM	BØFAC,ARG -SVECT,*+12	Single arg. UDRS which may internally branch and transmit
20	BT BT : BT	-SVECT,ARG_k -SVECT,ARG_{k-1} : : -SVECT,ARG1	Multi-argument, single entry UDRS. If one of the arguments is already in BØFAC when the UDRS is called, it will not be transmitted by a BT-SVECT,ARG. If one of the arguments is not already in BØFAC, then ARG_k is placed in BØFAC and ARG_{k-1} to ARG_1 are transmitted through SVECT. This type of entry is designed for functions like MAX and MIN.
25	BT	-SVECT,ARG	Single argument UDRS which may not internally branch and transmit
30	TF BT : BT BT	BØFAC, ARG_k -SVECT,ARG_{k-1} : : -SVECT,ARG_2 -SVECT,ARG_1	Similar to n_1n_2=20 above, where ARG_k is forced into BØFAC
35	BTM : BTM BTM	-SVECT,ARG_k : : -SVECT,ARG_1 -SVECT+5,*+12	A UDRS which may have any number of arguments (including no arguments at all) and which may branch and transmit internally, and which does not have any of its arguments in BØFAC when entered. This entry is required if the UDRS is to be used as a subroutine subprogram.

If n_1 is flagged, the UDRS is expected to produce a floating point result. If n_1 is not flagged, the UDRS is expected to produce an integer result.

n_3 can have 3 possible values:

$\bar{0}$ denotes an even function, i.e. f(x)=f(-x); used only for single arg. functions

$\bar{1}$ denotes an odd function, i.e. f(x)=-f(-x); used only for single arg. functions.

0 denotes a function which is neither odd nor even.

If in doubt, set n_3=0, which will never cause trouble. (Certain economies at object time are possible if the compiler knows whether the function is odd or even).

A UDRS may be used as an arithmetic function or as a subroutine subprogram. If a UDRS is called as the result of the appearance of its name in FORTRAN arithmetic statement or expression, it will be compiled as if it were a function; that is, it must have a single number for a result, and this result must be left in BØFAC on exit from the UDRS.

However, if the UDRS is used as a subroutine subprogram its name must appear in a FORTRAN CALL statement. In this case, more than one result can be generated, and these are transmitted back to the calling program by the formal parameter list or through COMMON storage. The only permissible n_1n_2 for this case is 35.

The same subprogram may be given several names. All that is necessary is to construct several entries in the trailer card using the same subprogram number and code digits, but different names. Entries on the trailer cards must be packed with no blanks. After the last entry on each card, a single record mark (0-2-8) is placed. After the last entry on the last trailer card, two record marks are placed.

For example, a trailer card might look like:

```
05BLAP(1ŌŌ)05BLAPF(1ŌŌ)10GURK(200)18A(15Ī)10RUNCH(200)
10FLAPF(200)‡
```

The code digits have been described above. Subprogram 05 is known by the names BLAP and BLAPF, subprogram 10 by GURK,RUNCH and FLAPF, and subprogram 18 as A.

As many trailer cards as necessary are constructed and placed after the compiler deck. They should be inserted between the second and third card from the end. The last two cards contain the names of the system relocatables; the last card contains two record marks at the end of its entries (see above).

PROGRAMMING RELOCATABLE SUBPROGRAMS

UDRS are to be coded and assembled using either IBM 1620/1710 SPS II or AFIT SPS. To aid in understanding the process of adding subprograms, a short description of the subprogram relocator behaviour follows.

The relocator will relocate all addresses of 80000 or more. Thus subprograms are preceded with a DORG 80000 instruction. Any address below 80000 will not be relocated. The relocator assumes that all instructions are to be relocated; however a constant (defined by DC, DSC, DAC) or a DSA will be relocated only if its location is at or above 80000. Furthermore, the constant assembled from a DSA will have both its value and its location relocated if they lie at or above 80000 and provided also that it occurs before the first executable instruction of the subprogram. This will become clearer later.

The compiler constructs entry commands to the subprogram as described previously. Note that the user can force compilation of instructions (in the object program) which culminate with BTM -SVECT,*+12. Since in this case, the UDRS has a real 5-digit return address stored away and control is passed back to the main program by branching indirectly to this 5 digit address. This allows the user to employ BT and BTM instructions in his own subprogram. This feature permits direct access to all the floating arithmetic routines and also to other relocatable subprograms. It is also possible for a UDRS to call other relocatables, even though the FORTRAN source program does not require them directly.

Linkage to the relocated subprogram is provided by a transfer vector located in the permanent subroutine package. It is defined by a DSB in the permanent package source deck, a copy of which is included with the systems decks. There is thus no reason for the user to concern himself with absolute addresses, since each UDRS will be compiled with the aid of the permanent subroutines source deck.

The user must specify certain information, in SPS, before the coding of his UDRS. This information becomes the header cards used by the relocator to select and relocate the subprogram properly.

For functions entered by BT -SVECT, ARG, the coding must look like:

```
DORG    0

DC      2,NN
DC      2,II
DC      2,JJ
DC      2,KK
-       --
-       --
DORG    BØSVECT-9+5*NN
DSA     START, 99999, NXX-80001

DORG    80000
DS      10
```

```
START function coding
      -
      -
      -
NXX   DAS    1
      DEND
```

The list of DC's at the beginning define the subprogram number NN, followed by the numbers II, JJ, KK, etc. of the relocatable subprograms required by this subprogram. These constants must be preceded by a DORG 0 (zero); the relocator identifies them as subprogram numbers by this fact. A subprogram may call a limit of 29 other subprograms, i.e. there may be a maximum of 30 DC's in this list.

The next item is a DORG to the proper place in the transfer vector BØSVECT, followed by a DSA list. The first item in this list is the address of the first executed instruction of the subprogram. The next item in the DSA list must be 99999. The next item is constructed such that it will assemble to a 5 digit number which is the size of the subprogram. Note that NXX (or other suitable label) is a DAS 1 which must appear just before the DEND statement of the subprogram. Obviously NXX-80001 will be an even number and will be the number of digits occupied by the subprogram. The relocator assumes that the DSA immediately following a DSA 99999 is the subroutine size.

The coding of the subprogram itself is preceded by DORG 80000.

In the case of multiple argument subprograms, two entries in the transfer vector are necessary. The subprogram arguments are transmitted through vector location BØSVECT +5*MM, where NN is the subprogram number; the return address is transmitted through BØSVECT-9+5*NN. The programmer must provide coding to handle the arguments and return address as they are transmitted. For such a subprogram, two DSA's for transfer vector entries must be programmed. For example,

```
        DORG    0
        DC      2,MM
        DORG    BØSVECT- 9 +5*MM
        DSA     START,SOAK,99999,NY-80001
        DORG    80000

SOAK    -
        -
        -
START   -
        -
        -
NY      DAS     1
```

In such a case, SOAK is the entry for coding to handle argument addresses, and START is the beginning of the multi-argument subprogram.

MM is the 2-digit subprogram number. Note that the two entries in the transfer vector must be contiguous, and that the first one may not be used for any other subprogram. Thus, if MM is the number of this subprogram, MM+1 may not be used as the number of any other subprogram, since the transfer vector location which it needs has already been used by multiple argument subprogram MM.

For example, let us program a subprogram to calculate the hyperbolic sine of a floating point argument, by the well known formula

$$SINH(X) = \frac{1}{2}(e^{X} - e^{-X})$$

To do this we will need the exponential routine, subprogram number 69 and we will use the floating subtract and multiply routines.

```
       HEAD    K
       HYPERBOLIC SINE OF X, SUBPROGRAM NO. 12
       DORG    0
       DC      2,12
       DC      2,69

       DORG    BØSVECT-9+5*12
       DSA     SINH,99999,N12-80001

       DORG    80000
       DS      5
SINH   TF      BIN2,BØFAC
       BT      BØSVECT-5+5*69,BØFAC,6
       TF      BIN1,BØFAC
       TF      BØFAC,BIN2
       BT      BØRVSGN,BØRVSGN-1
       BT      BØSVECT-5+5*69,BØFAC,6
       BT      BØFSBR,BIN2
       BT      BØFMP,FLHAF
       B       SINH-1,,6
       DORG    *-3
FLHAF  DC      10,5050000000
BIN1   DS      10
BIN2   DS      10
N12    DAS     1
       DEND
```

Several points should be noted

(1) This is an example only. Much better methods for SINH(X) exist.

(2) A UDRS must be headed. DO NOT USE HEADING CHARACTERS B NOR S. These are already used in the permanent subprograms.

(3) The first DC defines this subprogram as number 12. The second DS causes the relocator to load in subprogram number 69, the exponential routine (see Table II for a list of systems relocatables).

(4) The next DORG and DSA define the transfer vector entry and the subprogram length.

(5) Since this subprogram calls other subprograms, it will be entered by having the argument in B$FAC, and the compiler will generate BTM B$SVECT-5+5*12,*+12,6. The compiler trailer card entry would be:

12SINH($\bar{I}$0$\bar{0}$)

(6) The coding for the subprogram follows directly. The first instruction saves the argument x, the second calculates e^{x}, the third instruction stores this result away. We then reverse sign of x in the next two instructions, and calculate e^{-x}.

The two exponentials are then subtracted, and multiplied by 0.5. The result remains in B$FAC, and the subprogram branches indirectly to the return address carried into the subprogram.

A special method must be used to handle DSA's which are used internally in a UDRS. A DSA used internally must have both its value and itslocation adjusted by the relocator. A true constant, defined by a DSC, DC or DAC, must have its location adjusted but its value left unchanged. Unfortunately, in a condensed deck prepared by IBM 1620/1710 SPS II, a DSA and a constant are indistinguishable. For this reason, the following rules must be observed.

RULE 1: Any DSA which is local to the subprogram and which is to have both its location and its value adjusted by the relocator, must be defined after the DORG 80000 statement and before the first instruction ofthe subprogram.

RULE 2: Any constant which is local to the subprogram and which is to have only its location adjusted by the relocator, must be defined after the first instruction of the subprogram. For example,

```
*      EXAMPLE SUBPROGRAM NO. 38
       DORG    0
       DC      2,38
       DORG    B$SVECT-9+5*38
       DSA     GLOP,99999,N38-80001

       DORG    80000
       DSA     A1,A2,A3
       DS      20
GLOP   (an instruction)
       -
       -
       -
A1     -
A2     -
A3     -
       -
       -
GORP   DC      25,0
GORP1  DC      35,1
N38    DAS     1
```

The DSA A1,A2,A3, will be adjusted as required, because it occurs before the first instruction which in this case is labelled GLOP. The two DC's, GORP and GORP1 will have their location adjusted, but their value unchanged, because they occur after the first instruction.

ASSEMBLY OF A UDRS

Program the subprogram as described above. Place it behind the source deck for the permanent subprograms. Using IBM 1620/1710 SPS II, or AFIT SPS, put this combined source deck through Pass 1 of the assembly in the normal way. For PASS II of the assembly, read in only the source deck for the UDRS; it is not necessary to read in the source deck of the permanent subprograms for Pass I. Get a condensed object deck for the UDRS. Throw away the first two and last seven cards of this deck. What remains is the subprogram coding itself, preceded by its built-in headers.

The UDRS condensed deck is to be inserted in the deck after the relocator, which ends with card No. ZZZ. The UDRS must be located physically in front of any subprogram which it calls. If this is not adhered to, an error message will result when loading an object deck (see operating instructions). Obviously the relocator cannot load a subprogram which it has already bypassed before the calling subprogram appeared.

TABLE II

Subprogram Number	Length of Subprogram	Sub.called by this Sub.	Entry to Subprogram[1]	Purpose of Subprogram
67	0	68	BT -SVECT,ARG	Trigonometric COSINE of argument
68	694		BT -SVECT,ARG	Trigonometric Sine of argument
96	132	69,70	BT -B$EXPØ3,A BT -B$EXPØ4,B	Reverse Float-Float Exponentiation A**FAC→FAC Float-Float Exponentiation FAC**B→FAC
69	528		BT -SVECT,ARG	Exponential function of argument,Exp(arg)
70	578		BT -SVECT,ARG	Natural logarithm of argument
71	308		BT - SVECT,ARG	Square Root of argument
72	866		BT -SVECT,*+12	Arctangent of argument; arg. in B$FAC
76	54		BT -SVECT,ARG	Signum of Arg; Argument: >0 Result: +1. =0 0. <0 -1.
74	304		BT -SVECT,ARG	Random number generator; see spec.description
77				Larger of (arg$_1$,arg$_2$)
78				Smaller of (arg$_1$,arg$_2$)
73		72		Arctangent of (arg$_1$/arg$_2$)
75				Sign of arg$_1$,arg$_2$. Magnitude of arg. with the sign of arg$_2$.
79	1048		Special	Input of E,F,I, or N-type numbers
80	1056		Special	Output of E,F,I, or N type numbers
81	124		Special	Routine to handle Hollerith Fields
82	100		Special	Routine to handle I/O implied DO's
83	104	84	Special	Routine to handle I/O of arrays (formal)
84	68			Routine to handle I/O of arrays
85	156		Special	I/O subscripting routine for A(I,J)
86	122	92	Special	Accept
87	74		Special	Type
88	436		Special	Print (on-line)
89	122	92	Special	Accept tape
90	130		Special	Punch tape
91	268		Special	Reread
92	64		Special	Snip (part of Accept and Accept Tape)
93	1794	94,95	Special	Variable Format
94	180		Special	Routine to handle repeated,parenthesized,Format
95	176		Special	Routine to handle A-type numbers
97	188		BT -B$EXPØ1,A BT -BSEXPØ2,I	Reverse Float-Fixed Exponentiation,A**FAC→FAC Float-Fixed Exponentiation,FAC**I→FAC
98	226		BT -BSEXPØ5,I BT -BSEXPØ6,J	Reverse Fixed-Fixed Exponentiation,A**FAC→FAC Fixed-Fixed Exponentiation FAC**I→FAC
99	122		BT -B$D1,I BT -B$D2,J	Reverse integer division I/FAC→FAC Integer division FAC/J→FAC

[1] SVECT is the location in the transfer vector which contains the link address for the subprogram. Its exact position is B$SVECT-5+5*NN, where NN is the subprogram number

TABLE III

Permanent Subroutines

Routine	Purpose	Symbolic Address	Entry to Subroutine
Floating Add	FAC+A→FAC	B$FAD	BT B$FAD,A
Floating Subtract	FAC-A→FAC	B$FSB	BT B$FSB,A
Reverse Floating Subtract	A-FAC→FAC	B$FSBR	BT B$FSBR,A
Floating Multiply	FAC*A→FAC	B$FMP	BT B$FMP,A
Floating Divide	FAC/A→FAC	B$FDV	BT B$FDV,A
Reverse Floating Divide	A/FAC→FAC	B$FDVR	BT B$FDVR,A
Reverse Fixed Subtract	I-FAC→FAC	B$FXSR	BT B$FXSR,I
Fixed Multiply	FAC*I→FAC	B$FXM	BT B$FXM,I
Reverse Sign	-FAC→FAC	B$RVSGN	BT B$RVSGN,B$RVSGN-1
Float	(A=I)→FAC	B$FLOAT	BT B$FLOAT,I
Fix	(I=A)→FAC	B$FIX	BT B$FIX,A
Zero Accumulator	Floating Zero→FAC	B$ZERFC	BT B$ZERFC,B$ZERFC-1
Floating Overflow (sets error code)	± Floating nines →FAC	B$ER9	BT B$ER9,B$ER9-1(sign of answer must be in location 00099)
Floating underflow	Floating zero→FAC	B$ERØ	BT B$ERØ,B$ERØ-1
STOP N	See general specifications	B$STØP	BT B$STØP,N
PAUSE N	" "	B$PZUSE	BT B$PAUSE,N
CALL ERRCK(N)	Check error field	B$ERRCK	BTM B$ERRCK,N
CALL EPRT	Print out error field	B$EPRT	BT B$EPRT,B$EPRT-1
CALL RESET	Reset error field to zeros	B$RESET	BT B$RESET,B$RESET-1
CALL SKIP	Find next block or file card	B$EXIT	BTM B$EXIT,0,10
CALL EXIT	Find next program	B$EXIT	BTM B$EXIT,1,10
CALL EOFB(N)	Interrogate block and file indicators	B$EOFBR	BTM B$EOFBR,N
Return typewriter carriage and type a message	Obvious	B$TWSR	BTM B$TWSR,LOC,,where LOC is the address of the record to be typed.

TABLE IV

Useful Constants and Their Addresses

Address	Constant
BØFLONE	Field Address of $\bar{5}110000000$
BØNINES	Field Address of $\bar{9}999999999999$
BØFNINE	Field Address of $\bar{9}999999999$
BØFZERØ	Field Address of $\bar{0}000000000$
BØFZERC	Defined as DC 21,@
BØONE	Defined as DC 14,10000000000000
BØERRF	Defined as DSC 18,0 This is the error field.

HINTS AND NOTES

(1) All object time subprograms should assume that the arithmetic overflow light is ON. Fixed add and subtract are done in line, not by subroutine, so there is lots of opportunity for it to get turned on. All the routines of Table III assume the overflow is ON. Conversely, if your subprogram turns on the overflow light, the other routines could not care less. They turn it off themselves if they need it.

(2) The console area (locations 00000 to 00099) is available for work area. Routines of all sorts use it for temporary storage. Watch out for possible complications if your subprogram calls other subprograms which also use the console area as a scratch pad. The accumulator is in BØFAC, loc 50-59. Put the result of a function subprogram there.

(3) The error field has blank spaces in it for user-defined error codes. The high order digit of the error field is in location BØERRF. Use a TDM instruction to put a suitable digit in the right place. Do not used flagged digits in the error field. It will foul up the compare instruction used to find out whether the field is all zero or not.

The following papers were presented at the joint Canadian and Mid-western Regional meeting of the IBM 1620 Users Group in Chicago, February 19-21, 1964.

These programs will be submitted to the 1620 Users Group Program Library in the near future.

CARLETON COLLEGE COMPILER by Donald H. Taranto, Carleton College, Northfield, Minn.

The Carleton College Compiler is a load-and-go algebraic compiler designed especially for the 20K, automatic floating-point card system with indirect addressing and additional instructions. Compilation and execution are fast, and batch operation is handled quickly.

The language includes the usual 4-digit fixed-point, 10-digit mantissa floating-point, and elementary function arithmetic. In addition, there are boolean, maximum-minimum, and remainder operations. Subroutine calls are allowed and flexible alphameric typed output is available.

The entire compiler occupies 9-11K of core (depending upon what function subroutines are included) and is practically fool-proof. Source statements are thoroughly checked for legality during compilation.

This fast, versatile compiler is bought at the price of severe (but not serious) restrictions on variable and statement formats.

CARLETON BINARY SIMULATOR by William R. Gage, Carleton College, Northfield, Minn.

The Carleton Binary Simulator is an interpreter which turns the 1620 into a fixed-word-length, single address, binary computer. There are 4096 words of 16 bits (15 bits and sign) each, a 32-bit accumulator-remainder unit, and a 12-bit index register. There is a generous supply of arithmetic, boolean, control, and input-output instructions.

This versatile and unusual interpreter is bought at the price of rather slow execution of a source program. Machine requirements are 20K Card System, Automatic Divide, Indirect Addressing, and Additional Instructions.

Modifying Monitor I to Include other Programming Systems

by

Alan V. Purcell
Engineering Computing Laboratory
University of Wisconsin
Madison, Wisconsin

Modifying Monitor I
to Include other Programming Systems

Modifying Monitor I
to Include other Programming Systems

I Introduction

The purpose of this paper is to show how it is possible to include other programming systems in the Monitor I package, even if the systems to be added require different types of control cards and occupy the same memory locations as the Monitor I Supervisor. To warrant such a modification, the system would have to be in use frequently enough to justify including it on a level with FORTRAN II-D, the Disk Utility Program, and SPS II-D and it would presumably be undesireable to re-assemble it under SPS II-D and use it under the XEQ option of Monitor I.

The resulting modified monitor system would have complete compatibility with Monitor I, i.e. all Monitor I functions would be performed as before. In addition, the modified Supervisor would recognize control cards for the additional system and transfer control to it when such an option is specified. The system to be added would require some modification to require it to return control to the Supervisor routine when a Monitor I control card is specified.

In this paper the necessary changes to Monitor I are explained. As an example of a typical system to be added, the FORGO Fortran compiler is used throughout. By way of explanation and background, FORGO is a compiler which is uniquely suited for educational use because:

(a) FORGO is a load-and-go FORTRAN compiler. Since it resides in memory at all times, it eliminates processor reloading and object deck handling.

(b) It has exremenly complete diagnostics, both at compile time and run time. Even at run time, all comments are referred back to the user's source language program.

(c) FORMAT is optional, permitting the postponing of this single most complicated FORTRAN statement until after running experience has been gained.

It should also be noted that the version of FORGO used was the two pass system, in which the compiler section is overlayed in memory by the subroutines at program execution time. The compiler section is known as FOR-TO-GO A and the subroutine section is known as FOR-TO-GO B.

Under this scheme, if a FORGO control card is recognized by the Monitor I Supervisor, FOR-TO-GO A is called into core. If the program is accepted, FOR-TO-GO B is called in and the program executed. If a Monitor I Control Record is then read in, control returns to the Supervisor, and the appropriate system is called in. Thus mixing types of jobs (i.e. FORTRAN II, SPS, FORGO, previously assembled or compiled object program, etc.) is perfectly allowable and requires no operator intervention to load decks.

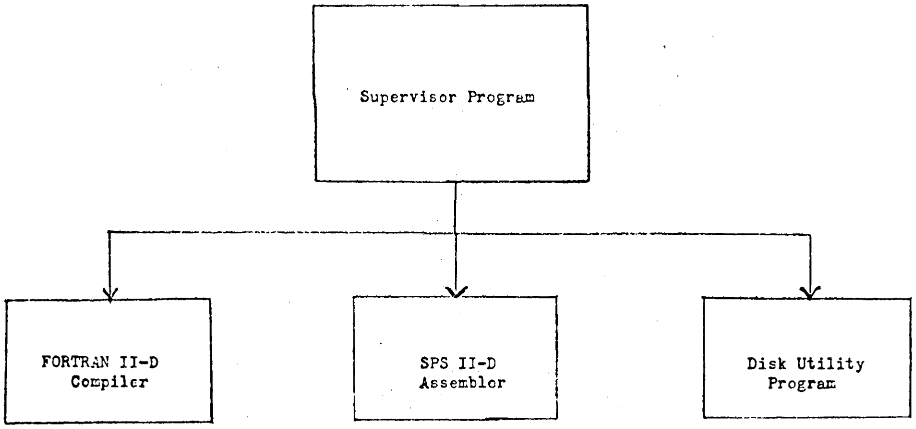


Figure 1 - IBM 1620 Monitor I System

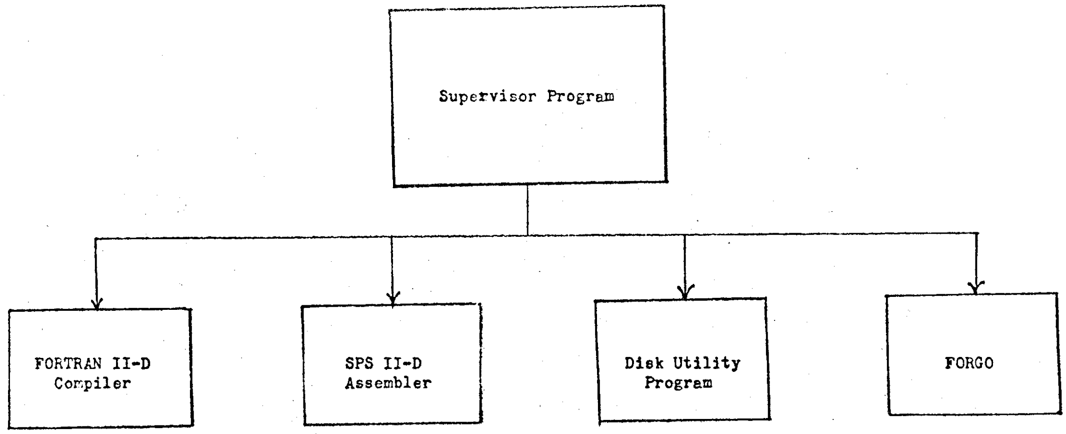


Figure 2 - Educational Monitor System

II Integration of the systems

A. Modifications to Monitor I

The basic modification to the Monitor I system is, of course, the inclusion of the additional system to be added. FORGO is used here as an example.

This modification is made more difficult by the fact that the supervisory routine of Monitor I requires all core locations below location 2402, as does the FORGO compiler. This means that FORGO will have to replace the Supervisor in memory, yet be called into memory under control of the Supervisor. Also, patches to Monitor I must go above the area used by the Supervisor (i.e. above 13160) and must be replaced every time they are destroyed. The specific patches to accomplish this are found in the Appendix; it suffices to outline them generally here.

The patches to Monitor consist of two main parts; one is the routine which reads in the patch area of the Supervisor every time it is destroyed, and the other is the routine which scans the incoming cards for FORGO control cards--as well as for Monitor I control cards. This first patch area, beginning at location 2914 in the Supervisor, reads the Monitor I Supervisor patch area into location 13162 and branches to it to execute the instructions displaced by the first patch area. The choice of location 2914 to begin these read instructions was not arbitrary. The instructions in this area are executed every time the Supervisor is read into core, thus assuring that the second patch is in core also. It is a location that makes certain that several disk indicators are reset, so that reading in the second patch area does not cause erratic disk operations.

The second patch area forms the linkage between the Monitor I System and the FORGO compiler. Upon recognition of a FORGO control card, the FORGO compiler is read in and supervisory control passes to it. Thus, in this patch area is the routine which scans for FORGO control cards.

B. Modifications to FORGO--A typical System to be added

The patches to FORGO are chiefly those required to link FORGO and the Monitor I Supervisor (see flow chart, Fig. 3). Using the modified system, if a FORGO control card is recognized by the Supervisor, FOR-TO-GO A (the compiler) is called into core from disk. If the program is acceptable, FOR-TO-GO B (the subroutines) is called in and the program is executed. Control is then returned to the Supervisor, and the process repeated. Every card read under the supervisory control of FORGO is checked to see if it is a Monitor I control card; if it is, FORGO operation is terminated with an error comment if appropriate, the Supervisor is read from disk, the card is set-up in the Supervisor input area in memory, and supervisory control is relinquished to the Supervisor.

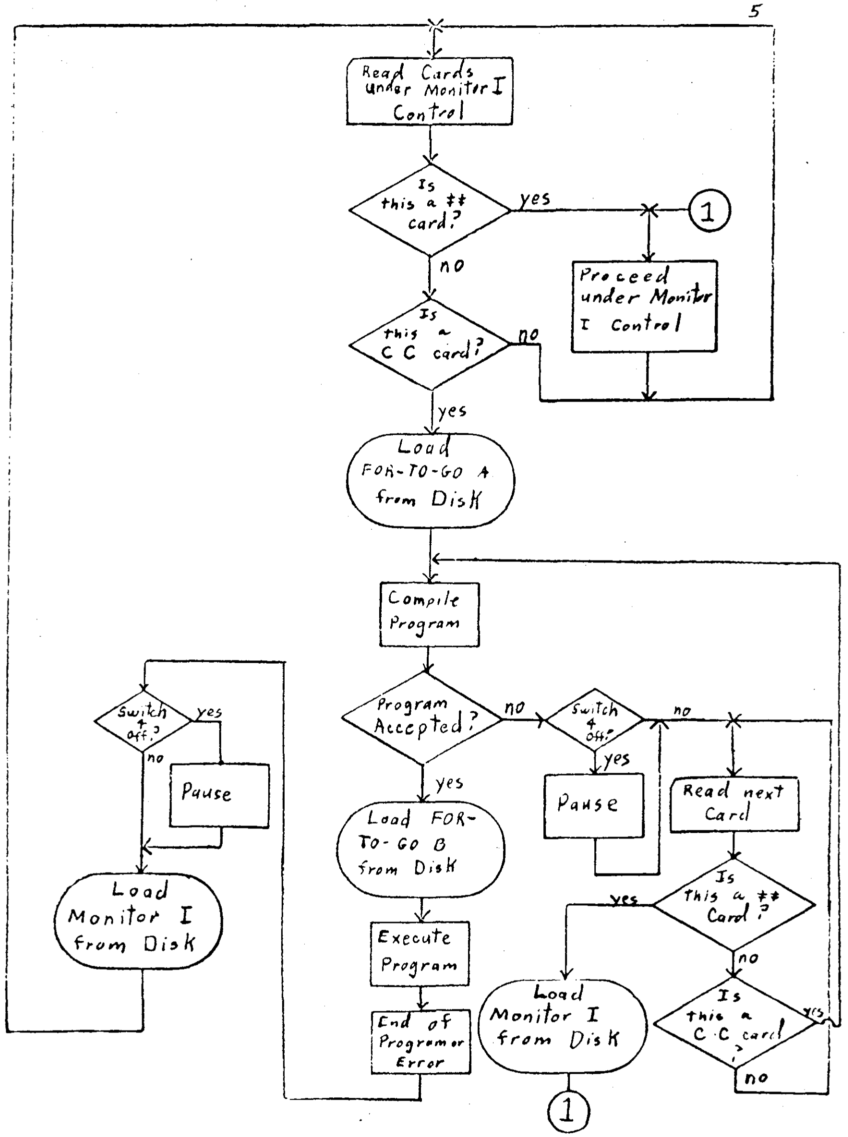


Figure 3 - Flow-chart of Modified System

C. Creation of the Resulting System

The procedure used in creating the working version of the resulting system is given here. The pertinent listings and typewriter sheets are given in the appendix of this paper.

The same basic procedure is used in adding the main patch area to Monitor I and in adding the additional system. It is necessary that the actual changes to the Supervisor, which are done using the D. U. P. routine DALTR, be done last.

The basic procedure was to first load the Monitor I system on disk as described in the Monitor I Systems Reference Manual. After this has been accomplished the system to be added, for example, FOR-TO-GO A, is loaded into core. The Disk Write Program is used to transfer it to the work cylinders, and the Monitor I D. U. P. routine DLOAD is used to move the information from the work cylinders to the desired disk cylinders-in this case cylinders twenty-six and twenty-seven were used to contain FORGO and the Supervisor patch area. Exactly the same procedure is followed for adding the chief Supervisor patch area (which starts in core at 13162).

To make the patches within the Monitor I Supervisor itself, the Disk Utility routine DALTR was used, and the desired changes were typed in (see typewriter sheets). Now the entire system in on disk in the form required for operation.

To get decks which will load under control of the Monitor I System Loader and to eliminate the need to do all of the preceding steps every time it is desired to reload the system on to disk, the DUP routine DDUMP was used. The system tables, the modified Supervisor, FOR-TO-GO A, FOR-TO-GO B, and the Monitor patches were dumped on cards. It was then necessary only to add the System Loader control card to each deck, the format of which is described in the Monitor I Systems Reference Manual. The systems tables deck replaces deck two of the original system, the Supervisor deck replaces deck seven, and the other decks are added at the end of the other Monitor I decks when it is desired to load the system on to disk.

Concerning ordinary operation of the modified system, it is the same as the Monitor I system. Cold start procedures are exactly the same. Disk cylinders twenty-six and twenty-seven are not available for use, however. These cylinders are protected by the Monitor I system tables.

III Some Suggested Changes to Monitor I

B. Improvements to Monitor I

Other systems could be added to Monitor I, using much the same techniques as were used in adding FORGO to Monitor I. Possible additional systems could be ALGOL, UW-SPS, COGO, etc. Suitable control cards could be designed, and the routine which scans for FORGO control cards could easily be expanded to include a scan for the other control card types.

A very important modification which should be made to the Monitor I system loader,whether FORGO has been added to the Monitor I system as described in this paper or not, would be one which puts read-only flags on the sector addresses of the Monitor I system routines as they are loaded onto the disk. The purpose of these flags is to file-protect the information contained on the flagged sectors; information contained thereon may then be read bút cannot be written over and destroyed. Although parts of the system must be left capable of being changed (i.e. the system tables), the unchanging parts could be file-protected by the loader--perhaps signaled to do so by a punch in a certain column in the heading control card of the decks to be loaded. Not file-protecting the system routines is a serious error on the part of the creators of the system, and as it stands, any SPS program could, through use of disk-write instructions, destroy the system routines on disk and necessitate reloading the entire system from cards.

```
                                        *****************************************************************
                                        *****************************************************************
                                        *****************************************************************
                                        *
                                        *       THESE ARE THE PATCHES TO THE IBM MONITOR I SYSTEM TO MAKE
                                        *       IT RECOGNIZE FORGO CONTROL CARDS AND TO CALL THE
                                        *       FOR-TO-GO A COMPILER FROM DISK.   FORGO OPERATION WILL BE
                                        *       EXACTLY AS ON A COMPUTER WITHOUT MONITOR, BUT COMPLETE
                                        *       MONITOR OPERATION IS MAINTAINED.
                                        *
                                        *                          * * *
                                        *
                                        *       ALAN V. PURCELL
                                        *       ENGINEERING COMPUTING LABORATORY
                                        *       THE UNIVERSITY OF WISCONSIN
                                        *       MADISON, WISCONSIN 53706
                                        *
                                        *****************************************************************
                                        *****************************************************************
                                        *****************************************************************
02914                                         DORG 2914,,,             IN SECTOR 19664
02914   49 02938 00000                        B    *+24
02926   48 00000 00000              HALT      H
02938   34 04088 00701                        K    DPTH,701
02950   36 04088 00702                        RN   DPTH,702,,          READ IN PATCH AREA FROM DISK
02962   46 02926 01900                        BA   HALT
02974   49 13162 00000                        B    PATCH
04080                                         DORG 4080,,,             IN SECTOR 19675
04080   49 13242 00000                        B    FORCD
04088                                         DORG *-3
04088             1                 DPTH      DC   1,1,,               THIS IS DDA FOR MONITOR PATCH AREA
04093             5                           DC   5,5387
04096             3                           DC   3,5
04101             5 J3162                     DSA  PATCH
04340                                         DORG 4340,,,             IN SECTOR 19678
04340   45 13310 13001                        BNR  CARD,13001
13162                                         DORG 13162
13162   25 09794 02878              PATCH     TD   9794,2878,,         REPLACE INSTRUCTIONS
13174   31 02110 04046                        TR   2110,4046
13186   26 02103 02857                        TF   2103,2857
```

```
13198   25 00440 02857                        TD   440,2857
13210   32 00456 00000                        SF   456,,,              REPLACE INSTRUCTION
13222   15 01967 00009                        TDM  1967,9
13234   49 02986 00000                        B    2986
13242                                         DORG *-3
13242   25 13265 02855              FORCD     TD   *+23,2855,,         REPLACE INSTRUCTIONS
13254   26 04941 10700                        TF   4941,10700
13266   44 04104 13160                        BNF  4104,13160,,        CHECK IF CARD READ BY FORGO
13278   15 09828 00001                        TDM  9828,1,,            INHIBIT READING A CONTROL CARD
13290   33 13160 00000                        CF   13160
13302   49 04104 00000                        B    4104
13310                                         DORG *-3
                                    *
                                    *         CHECK FOR FORGO CONTROL CARDS
                                    *
13310   33 13001 00000              CARD      CF   INAR
13322   32 13000 00000                        SF   INAR-1
13334   14 13001 000M3                        CM   INAR,43,10
13346   47 04056 01200                        BNE  BACK
13358   32 13002 00000                        SF   INAR+1,,,           PUT FLAGS ON EVEN NUMBERED
13370   33 13003 00000                        CF   INAR+2,,,           POSITIONS FOR FIRST FOUR COLUMNS
13382   14 13003 00000                        CM   INAR+2,0,10
13394   47 04056 01200                        BNE  BACK
13406   32 13004 00000                        SF   INAR+3
13418   33 13005 00000                        CF   INAR+4
13430   14 13005 00000                        CM   INAR+4,0,10
13442   47 04056 01200                        BNE  BACK
13454   32 13006 00000                        SF   INAR+5
13466   33 13007 00000                        CF   INAR+6
13478   14 13007 000M3                        CM   INAR+6,43,10
13490   47 04056 01200                        BNE  BACK
13502   34 13602 00701                        K    FORA,701,,          A FORGO CONTROL CARD HAS BEEN FOUND
13514   39 13001 00400                        WACD INAR
13526   46 13538 00900                        BLC  *+12
13538   31 18600 13558                        TR   18600,*+20
13550   49 18612 00000                        B    18612
13558                                         DORG *-3
13558   48 00000 00000              HT        H
13570   36 13602 00702                        RN   FORA,702,,          READ IN FOR-TO-GO A
13582   46 13558 01900                        BA   HT
                                    *
```

```
                                                 *     NOTE---BEFORE  DUMPING FOR-TO-GO A FOR DISK, AT LEAST ONE       00167
                                                 *         CARD MUST HAVE BEEN READ SO THAT THERE ARE FLAGS IN THE     00169
                                                 *         CORRECT POSITIONS IN THE I/O AREAS.                         00171
                                                 *                                                                     00173
13594  49 01342 00000                                 B     1342                                                       00175
13601         1                                       DC    1,@,*-4                                                    00177
13602                                                 DORG  *-3                                                        00180
13001         0                               INAR    DS    ,13001                                                     00182
04056         0                               BACK    DS    ,4056                                                      00184
13602         1                               FORA    DC    1,1,,                     THIS IS DDA FOR FOR-TO-GO A      00186
13607         5                                       DC    5,5200                                                     00189
13610         3                                       DC    3,186                                                      00192
13615         5                                       DC    5,0                                                        00195
13162                                                 DEND  13162                                                      00198
```

97

98

```
                          101     *******************************************************************
                          102     *******************************************************************
                          103     *******************************************************************
                          104     *
                          105     *           THESE ARE THE PATCHES TO THE 3-62 FOR-TO-GO A DECK
                          106     *           TO OPERATE  UNDER CONTROL OF THE IBM MONITOR
                          107     *           OPERATING SYSTEM.  TO ALL APPEARANCES, OPERATION IS
                          108     *           JUST LIKE REGULAR FORGO ON A NON-MONITOR SYSTEM.
                          109     *
                          110     *                                     * * *
                          111     *
                          112     *           ALAN V. PURCELL
                          113     *           ENGINEERING COMPUTING LABORATORY
                          114     *           THE UNIVERSITY OF WISCONSIN
                          115     *           MADISON 6, WISCONSIN
                          116     *           NOVEMBER, 1963
                          117     *
                          118     *******************************************************************
                          119     *******************************************************************
                          120     *******************************************************************
00904                     121             DORG 904
00908          5 J8540    122             DSA  FIRDIG
01070                     123             DORG 1070
01070  47 09404 00300     124             BNC3 9404
01234                     125             DORG 1234
01234  49 18080 00000     126             B    RECA
02144                     127             DORG 2144
02144  49 18328 00000     128             B    NSCAN
09404                     129             DORG 9404
09404  34 18440 00701     130             K    FORB,701,,            READ B DECK IN FROM DISK
09416  49 18490 00000     131             B    READB
09668                     132             DORG 9668
09668  39 09491 00100     133             WATY 9491
09680  43 18420 18195     134             BD   LC3,PNA,,             TEST IF LC-3 ERROR
09692  46 01114 00400     135             BC4  1114
18080                     136             DORG 18080
18080  45 18100 00423     137     RECA    BNR  *+20,423,,            CHECK FOR RM ON C  C CARD
18092  49 18140 00000     138             B    RECMK-48
18100                     139             DORG *-3
18100  45 18120 00425     140             BNR  *+20,425
```

```
18112  49 18140 00000   141          B    RECMK-48
18120                   142          DORG *-3
18120  24 01257 00423   143          C    1257,423,,           REPLACE INSTRUCTION
18132  49 01246 00000   144          B    1246
18140                   145          DORG *-3
18140  16 18295 -0581   146          TFM  INEND,581
18152  16 18463 -0422   147          TFM  INAR,422
18164  49 18188 00000   148          B    *+24
18176  48 00000 00000   149   HALT   H
18188  34 18464 00701   150   RECMK  K    MON,701,,            CALL IN MONITOR AND
18195            0      151   PNA    DS   ,*-4
18200  16 18458 J3000   152          TFM  INMON,13000,7,       PUT CARD IN INPUT AREA
18212  15 00031 00005   153          TDM  31,5
18224  36 18464 00702   154          RN   MON,702
18236  46 18176 01900   155          BA   HALT
18248  25 1845Q 1846L   156   TD     TD   -INMON,-INAR
18260  11 18458 000-1   157          AM   INMON,1,10
18272  11 18463 000-1   158          AM   INAR,1,10
18284  14 18463 -0581   159          CM   INAR,581
18295            0      160   INEND  DS   ,*
18296  47 18248 01300   161          BN   TD
18308  32 13160 00000   162          SF   13160,,,             INDICATE CARD ALREADY READ
18320  49 02402 00000   163          B    2402
18328                   164          DORG *-3
18328  16 02162 00P37   165   NSCAN  TFM  2162,737,9,          REPLACE INSTRUCTION
18340  45 18360 00595   166          BNR  *+20,INPUT2
18352  49 18372 00000   167          B    *+20
18360                   168          DORG *-3
18360  45 02156 00597   169          BNR  2156,INPUT2+2
18372  16 18463 -0595   170          TFM  INAR,INPUT2
18384  16 18295 -0653   171          TFM  INEND,653
18396  15 18195 00001   172          TDM  PNA,1
18408  17 09892 N343K   173          BTM  9892,-53432,7,       ERROR LC-2 IF MON C CARD
18420  15 18195 00000   174   LC3    TDM  PNA,0,,              TURN OFF INDICATOR
18432  49 18188 00000   175          B    RECMK
18440                   176          DORG *-3
18440            1      177   FORB   DC   1,1,,                   THIS IS A DDA TO CALL B DECK
18445            5      178          DC   5,5400
18448            3      179          DC   3,175
18453            5      180          DC   5,958
18458            5      181   INMON  DS   5
```

```
18463            5      182   INAR   DS   5
00595            0      183   INPUT2 DS   ,595
18464            1      184   MON    DC   1,1,,                   THIS IS A DDA TO CALL MONITOR
18469            5      185          DC   5,19636
18472            3      186          DC   3,113
18477            5      187          DC   5,102
18478  48 00000 00000   188   HT     H
18490  36 18440 00702   189   READB  RN   FORB,702,,           READ IN B DECK FROM DISK
18502  46 18478 01900   190          BA   HT
18514  46 15918 01400   191          BV   15918,,,             TURN OFF OVERFLOW INDICATOR
18526  49 15918 00000   192          B    15918,,,             AND GO,GO,GO
18534                   193          DORG *-3
18540                   194          MORG 10
18540            1      195   FIRDIG DS   1
01070                   196          DEND 1070
```

```
                                101   ******************************************************************
                                102   ******************************************************************
                                103   ******************************************************************
                                104   *
                                105   *          THESE ARE THE PATCHES TO THE 3-62 FOR-TO-GO B DECK
                                106   *          TO OPERATE UNDER CONTROL OF THE IBM MONITOR
                                107   *          OPERATING SYSTEM.  TO ALL APPEARANCES, OPERATION IS
                                108   *          IDENITAL TO REGULAR FORGO ON A NON-MONITOR SYSTEM.
                                109   *
                                110   *                               * * *
                                111   *
                                112   *          ALAN V. PURCELL
                                113   *          ENGINEERING COMPUTING LABORATORY
                                114   *          THE UNIVERSITY OF WISCONSIN
                                115   *          MADISON, WISCONSIN 53706
                                116   *          NOVEMBER, 1963
                                117   *
                                118   ******************************************************************
                                119   ******************************************************************
                                120   ******************************************************************
```

101

```
01082                           121              DORG 1082
01082    49 18264 00000         122              B    READM,,,              READ IN MONITOR
17648                           123              DORG 17648
17648    49 18320 00000         124              B    END-12
04276                           125              DORG 4276
04276    49 18080 00000         126              B    BREC,,,               CHECK FOR RECORD MARKS IN DATA
18080                           127              DORG 18080
18080    16 03708 -0423         128   BREC       TFM  3708,423,7,           REPLACE INSTRUCTION
18092    45 18112 00423         129              BNR  *+20,423,,            TEST FOR MONITOR CONTROL CARD
18104    49 18124 00000         130              B    *+20
18112                           131              DORG *-3
18112    45 04288 00425         132              BNR  4288,425
18124    16 18368 00M22         133              TFM  INAR,422,9,           PUT CARD IN MONITOR INPUT AREA
18136    16 18368 -0422         134              TFM  INAR,422
18148    16 18373 J3000         135              TFM  INMON,13000
18160    25 1837L 1836Q         136   TD         TD   -INMON,-INAR
18172    11 18368 000-1         137              AM   INAR,1,10
18184    11 18373 000-1         138              AM   INMON,1,10
18196    14 18368 -0583         139              CM   INAR,583
18208    47 18160 01300         140              BN   TD



18220    32 13160 00000         141             SF   13160,,,              INDICATE MON. C. RECORD ALREADY
18232    17 17104 N3432         142             BTM  17104,53432,7,        READ.  ERROR LC-2
18244    49 18264 00000         143             B    *+20
18252                           144             DORG *-3
18252    48 00000 00000         145   HALT      H
18264    34 18374 00701         146   READM     K    MON,701
18276    15 00031 00005         147             TDM  31,5
18288    36 18374 00702         148             RN   MON,702,,             READ IN MONITOR
18300    46 18252 01900         149             BA   HALT
18312    49 02402 00000         150             B    2402
18320                           151             DORG *-3
18320    39 00595 00400         152             WACD 595,,,                REPLACE INSTRUCTION
18332    46 18356 00400         153   END       BC4  *+24
18344    48 00000 00000         154             H
18356    49 18264 00000         155             B    READM
18364                           156             DORG *-3
18368          5                157   INAR      DS   5
18373          5                158   INMON     DS   5
18374          1                159   MON       DC   1,1,,                    THIS IS A DDA TO CALL MONITOR
18379          5                160             DC   5,19636
18382          3                161             DC   3,113
18387          5                162             DC   5,102
01070                           163             DEND 1070
```

102

```
                                          ******************************************************************** 00002
                                          ******************************************************************** 00004
                                          *                                                                    00006
                                          *         THIS IS A PROGRAM TO WRITE A PROGRAM IN CORE ONTO  THE     00008
                                          *         DISK WORK CYLINDERS.  CHECKING IS DONE FOR CORRECT TRANSFER 00010
                                          *         OF DATA.                                                   00012
                                          *                                                                    00014
                                          *                              * * *                                 00016
                                          *                                                                    00018
                                          *         ALAN V. PURCELL                                            00020
                                          *         ENGINEERING COMPUTING LABORATORY                           00022
                                          *         THE UNIVERSITY OF WISCONSIN                                00024
                                          *         MADISON, WISCONSIN 53706                                   00026
                                          *                                                                    00028
                                          ******************************************************************** 00030
                                          ******************************************************************** 00032
19000                                           DORG 19000                                                     00034
19000   34 00000 00102              START       RCTY                                                           00036
19012   39 19263 00100                          WATY MES1                                                      00038
19024   34 00000 00102                          RCTY                                                           00040
19036   36 19254 00100                          RNTY IN-2                                                      00042
19048   34 00000 00102                          RCTY                                                           00044
19060   39 19321 00100                          WATY MES2                                                      00046
19072   34 00000 00102                          RCTY                                                           00048
19084   36 19257 00100                          RNTY BEGIN-4                                                   00050
19096   31 00000 19228                          TR   ,FORGO,,             TO PUT CORRECT FORGO INSTRUCTIONS    00052
19108   34 00000 00102              DISK        RCTY                                                           00054
19120   34 19248 00701                          K    WK,701                                                    00056
19132   38 19248 00702                          WN   WK,702                                                    00058
19144   36 19248 00703                          RN   WK,703                                                    00060
19156   47 19204 01900                          BNA  *+48                                                      00062
19168   39 19397 00100                          WATY MES3                                                      00064
19180   48 00000 00000                          H                                                              00066
19192   49 19108 00000                          B    DISK                                                      00068
19204   39 19487 00100                          WATY MES4                                                      00070
19216   48 00000 00000                          H                                                              00072
19228   41 00000 00100              FORGO       NOP  ,100                                                      00074
19240   49 01070 00000                          B    1070                                                      00076
19247                                           DORG *-4                                                       00078
19247            1                              DC   1,@                                                       00080


19248            1                  WK          DC   1,1,,                THIS IS THE DISK CONTROL FIELD       00083
19253            5                              DC   5,4000                                                    00086
19256            3                  IN          DC   3,0                                                       00089
19261            5                  BEGIN       DC   5,0                                                       00092
19263           29                  MES1        DAC  29,TYPE A 3 DIGIT SECTOR COUNT @,                         00095
19321           38                  MES2        DAC  38,TYPE A 5 DIGIT CORE STARTING ADDRESS @,                00099
19397           45                  MES3        DAC  45,INCORRECT DATA TRANSFER, PUSH START TO RETRY@,         00103
19487           26                  MES4        DAC  26,DISK OPERATION SUCCESSFUL@,                            00107
19000                                           DEND START                                                     00111
```

```
160000800000RS
EQUTAB LOADED FROM 10̄5000 TO 10̄5079
DIMFOR LOADED FROM 10̄4800 TO 10̄4999
SEQ PL LOADED FROM 01̄9801 TO 01̄9880
DUP A  LOADED FROM 11̄8139 TO 11̄8599
DUP B  LOADED FROM 11̄9300 TO 11̄9399
DUP C  LOADED FROM 11̄7084 TO 11̄7127
SUBSUP LOADED FROM 01̄7024 TO 01̄7074
ALLSUB LOADED FROM 01̄6400 TO 01̄6799
SPSIID LOADED FROM 01̄8600 TO 01̄9291
SUPERI LOADED FROM 11̄9600 TO 11̄9799
PH 1-A LOADED FROM 01̄7200 TO 01̄7339
PH 1+2 LOADED FROM 01̄7400 TO 01̄8138
LOAD 1 LOADED FROM 01̄6000 TO 01̄6199
LOAD 2 LOADED FROM 01̄6940 TO 01̄6964
SET 1  LOADED FROM 01̄9400 TO 01̄9599
SET 2  LOADED FROM 01̄6800 TO 01̄6939
DIM FS LOADED FROM 00̄4802 TO 00̄4807
FLN FS LOADED FROM 01̄6200 TO 01̄6211
FEXPFS LOADED FROM 01̄6212 TO 01̄6227
SUB FS LOADED FROM 01̄6228 TO 01̄6234
DKIOFS LOADED FROM 01̄6235 TO 01̄6267
S+C FS LOADED FROM 01̄6268 TO 01̄6280
FATNFS LOADED FROM 01̄6281 TO 01̄6298
SQRTFS LOADED FROM 01̄6299 TO 01̄6306
ABS FS LOADED FROM 01̄6307 TO 01̄6308


‡‡JOB 5

‡‡DUP 5

*DFINE                                             5


‡‡PAUS
END OF JOB
```

```
3-62A FOR-TO-GO
TYPE A 3 DIGIT SECTOR COUNT
186RS
TYPE A 5 DIGIT CORE STARTING ADDRESS
00000RS
DISK OPERATION SUCCESSFUL

‡‡JOB 5                             WRITE FROM WORK CYLINDERS

‡‡DUP

*DLOADFORGOA     02001040001041850052000000001070DIP026027

DUP* TURN ON  WRITE ADDRESS KEY, START
DUP* TURN OFF WRITE ADDRESS KEY, START
DK LOADED FORGOA 0̄200 00̄52001̄860̄24020̄1070‡
END OF JOB




3-62B. FOR-TO-GO
TYPE A 3 DIGIT SECTOR COUNT
175RS
TYPE A 5 DIGIT CORE STARTING ADDRESS
00958RS
DISK OPERATION SUCCESSFUL

‡‡JOB 5                             WRITE FROM WORK CYLINDERS

‡‡DUP

*DLOADFORGOB     02011040001041740054000095801070DIP027028

DUP* TURN ON  WRITE ADDRESS KEY, START
DUP* TURN OFF WRITE ADDRESS KEY, START
DK LOADED FORGOB 0̄201 00̄54001̄750̄09580̄1070‡
END OF JOB


TYPE A 3 DIGIT SECTOR COUNT
005RS
TYPE A 5 DIGIT CORE STARTING ADDRESS
13162RS
DISK OPERATION SUCCESSFUL

‡‡JOB 5                             WRITE FROM WORK CYLINDERS

‡‡DUP

*DLOADMONPAT     0202104000104004005387131620240 2DIP026027

DUP* TURN ON  WRITE ADDRESS KEY, START
DUP* TURN OFF WRITE ADDRESS KEY, START
DK LOADED MONPAT 0̄202 00̄53870̄051̄31620̄2402‡
END OF JOB
```

```
‡‡JOB 5                          ALTER MONITOR SECTORS

‡‡DUP

*DALTR

SECTOR
119664RS

1ST.HALF 3400000001 0225097940 2878310211 0040462602 1030285725  ORIGINAL
2ND.HALF 0044002857 3200456000 0015019670 0009430316 2004781509  ORIGINAL

SECTION
02RS          9
0225097940  TYPE CHANGE
XX4902838000004800000000003404088007013604088007024602926019004913162RS
1ST.HALF 3400000001 0225097940 2878310211 0040462602 1030285725  ORIGINAL
1ST.HALF 3400000001 0249028380 0000480000 0000003404 0880070136  CORRECTED

2ND.HALF 0044002857 3200456000 0015019670 0009430316 2004781509  ORIGINAL
2ND.HALF 0408800702 4602926019 0049131620 0009430316 2004781509  CORRECTED

SECTION
‡RS
DISK SECTOR 119664 CORRECTED
SECTOR
119675RS

1ST.HALF 5660̄403322 0̄4018‡11̄96 63001̄02802 ‡22̄0̄9732‡0 000‡0̄1̄0̄1̄0̄1̄  ORIGINAL
2ND.HALF 0̄1̄‡0150982 8000002504 1270285525 0410302855 2604941107  ORIGINAL

SECTION
08RS
1270285525  TYPE CHANGE          Note typing error
XXXXXXXXJ.UOIJIRS ←
1ST.HALF 5660̄403322 0̄4018‡11̄96 63001̄02802 ‡22̄0̄9732‡0 000‡0̄1̄0̄1̄0̄1̄  ORIGINAL
1ST.HALF 5660̄403322 0̄4018‡11̄96 63001̄02802 ‡22̄0̄9732‡0 000‡0̄1̄0̄1̄0̄1̄  CORRECTED

2ND.HALF 0̄1̄‡0150982 8000002504 1270285525 0410302855 2604941107  ORIGINAL
2ND.HALF 0̄1̄‡0150982 8000002504 12702855̄1̄3 4̄691̄902855 2604941107  CORRECTED

SECTION
08RS
12702855̄1̄3  TYPE CHANGE                  Correction of error
XXXXXXXX4913242010538700513162RS ←
1ST.HALF 5660̄403322 0̄4018‡11̄96 63001̄02802 ‡22̄0̄9732‡0 000‡0̄1̄0̄1̄0̄1̄  ORIGINAL
1ST.HALF 5660̄403322 0̄4018‡11̄96 63001̄02802 ‡22̄0̄9732‡0 000‡0̄1̄0̄1̄0̄1̄  CORRECTED

2ND.HALF 0̄1̄‡0150982 8000002504 12702855̄1̄3 4̄691̄902855 2604941107  ORICINAL
2ND.HALF 0̄1̄‡0150982 8000002504 1270285549 1324201053 8700513162  CORRECTED

SECTION
‡RS
DISK SECTOR 119675 CORRECTED
SECTOR
119678
```

```
       RS

1ST.HALF 6044043280 4127̄340000 0001021509 8280000145 0405613001  ORIGINAL
2ND.HALF 4504056130 03160970 40̄ 0004170803 81̄30054504 4440973045  ORIGINAL

SECTION
05[illegible]5613001  TYPE CHANGE

UOOURS ←——————— typing error
1ST.HALF 6044043280 4127̄340000 0001021509 8280000145 0405613001  ORIGINAL
1ST.HALF 6044043280 4127̄340000 0001021509 8280000145 4̄664̄613001  CORRECTED

2ND.HALF 4504056130 03160970 40̄ 0004170803 81̄30054504 4440973045  ORIGINAL
2ND.HALF 4504056130 031609704 0̄ 0004170803 81̄30054504 4440973045  CORRECTED

SECTION
05RS
4̄664̄613001  TYPE CHANGE
13310RS ←——————— correction of error
1ST.HALF 6044043280 4127̄340000 0001021509 8280000145 4̄664̄613001  ORIGINAL
1ST.HALF 6044043280 4127̄340000 0001021509 8280000145 1331013001  CORRECTED

2ND.HALF 4504056130 031609704 0̄ 0004170803 81̄30054504 4440973045  ORIGINAL
2ND.HALF 4504056130 031609704 0̄ 0004170803 81̄30054504 4440973045  CORRECTED

SECTION
‡RS
DISK SECTOR 119678 CORRECTED
SECTOR
‡RS
```

```
‡‡DUP
*DDUMP        CSO
END OF JOB
‡‡JOB 5
‡‡DUP
*DDUMP        CI
END OF JOB
‡‡JOB 5
‡‡DUP
*DDUMP        CE
END OF JOB
‡‡JOB 5
‡‡DUP
*DDUMP        CL  105200105385
END OF JOB
‡‡JOB 5
‡‡DUP
*DDUMP        CL  105387105391
END OF JOB
‡‡JOB 5
‡‡DUP
*DDUMP        CL  105400105574
END OF JOB
```

Bibliography

1. International Business Machines Corporation, *IBM 7090/7094 Programming Systems Manual, FORTRAN IV Language*, Form C 28-62 74-1, White Plains, New York, 1963.
2. __________, *IBM 1620 Data Processing System*, Form A 26-4500-2, White Plains, New York, 1961.
3. __________, *IBM 1620/1710 Symbolic Programming System*, Form C 26-5600-1, White Plains, New York, 1962.
4. __________, *IBM 1620 Monitor I Systems Reference Manual*, Form C 26-5739-1, White Plains, New York, 1963.
5. __________, *IBM 1620 Monitor I (Supervisor Listings)*, Form 1620-PR-026, White Plains, New York, 1963.
6. __________, *IBM 1311 Disk Storage Drive Model 3*, Form A 26-5650-1, White Plains, New York, 1963.
7. Leeds, Herbert D., and Weinberg, Gerald M., *Computer Programming Fundamentals*, New York, McGraw-Hill Book Company, Inc., 1961.
8. McClure, Charles W., *FORGO and FOR-TO-GO Manual*, White Plains, New York, 1620 General Program Library, IBM Corporation, 1961.

A New Course in Computer Appreciation

Charles H. Davidson
Engineering Computing Laboratory
University of Wisconsin

Computer Education is becoming a recognized necessity for the technical student in college. At Wisconsin it has been incorporated in the required experience of all engineers for some time, and is being made increasingly available to interested students with various backgrounds and degrees of preparation, as is indicated in Figure I, which lists the courses available in the Numerical Analysis Department.

The first entry in this table, however, represents an innovation in the teaching philosophy. Here for the first time is a course deliberately aimed at the non-technically trained student. As is pointed out in Figure II, the catalog description of the course, the only prerequisite is intermediate level high school algebra, equivalent to about two and one half units of high school mathematics; it is estimated that about 3/4 of our University freshmen are eligible to take this course.

"Introduction to Computing Machines" is intended to be more of a cultural than a professional course. Many of the students who take it may indeed never use a computer again, but they will all hear about computers every week of their lives. Whenever they receive a paycheck, register for a class, pay an insurance premium, make an airline reservation, or watch a rocket launching or an election return, it is almost certainly an IBM card or a computer-produced document they will be dealing with.

As the course is taught, the first two weeks are devoted to acquiring enough of a rudimentary knowledge of FORTRAN to be able to present simple problems to the computer. The rest of the course consists of examining some of the areas of significant application of the computer, classified as far as possible according to the particular advantages or capabilities of the computer. In each case, the students actually do simplified, watered down example problems illustrating its use, and extrapolations hopefully point out and make meaningful the true role of the computer in these areas. During the first semester it was offered, the students each did about twelve problems on the computer, including problems in:

1) finding roots of polynomials
2) class scheduling
3) sorting and table look-up
4) inventory control
5) missle tracking
6) library information retrieval

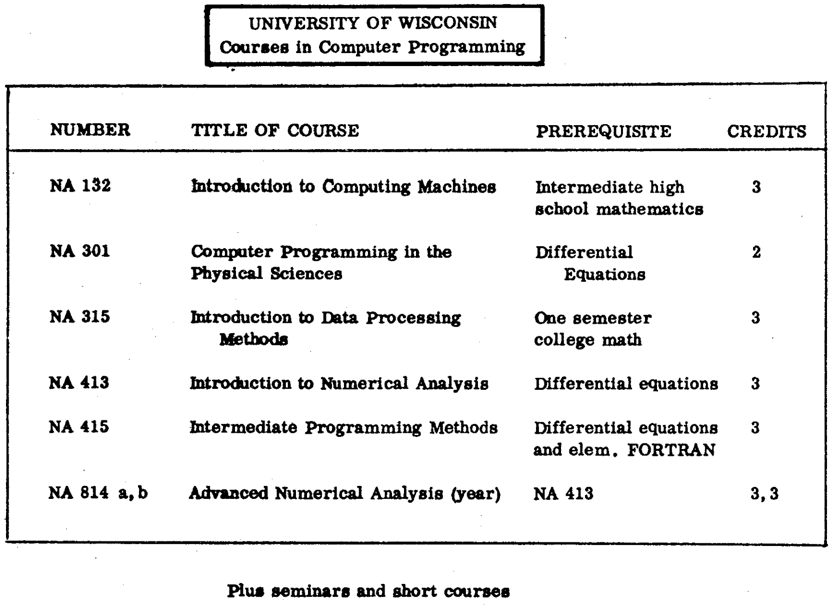
and several others.

All of this laboratory work has been done in the Engineering Computing Laboratory using the FORGO system, which is ideally suited for this type of teaching, with the exception of one problem near the end of the semester in which the use of the CDC 1604 and monitor system operation were demonstrated.

Figure III presents a condensed outline of the course, indicating some of the topics discussed and their sequence. Perhaps, however, one of the best pictures of the scope of the course can be obtained from the list of Review Questions shown in Figure IV, which was distributed shortly before final exam time, and represents material which they might be expected to have learned.

Since there is obviously no text-book existing which treats such a range of material in this fashion, we are preparing all of our own notes for the course, which will be published as a textbook. A preliminary version of the notes will be printed for use with the third offering of the course in the fall of 1964, and the official published version of the book is expected to be out in the late spring of 1965.

Some people are referring to the course in a colloquial fashion as "Computer Appreciation". This we accept as an apt description, provided it is modified to read "Introduction to Computing Machines--a do-it-yourself course in computer appreciation".

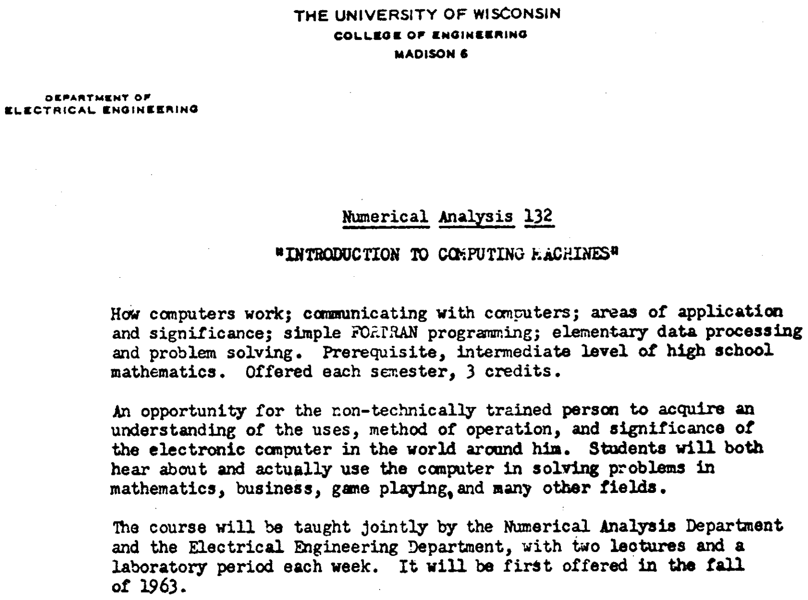


UNIVERSITY OF WISCONSIN
Courses in Computer Programming

NUMBER	TITLE OF COURSE	PREREQUISITE	CREDITS
NA 132	Introduction to Computing Machines	Intermediate high school mathematics	3
NA 301	Computer Programming in the Physical Sciences	Differential Equations	2
NA 315	Introduction to Data Processing Methods	One semester college math	3
NA 413	Introduction to Numerical Analysis	Differential equations	3
NA 415	Intermediate Programming Methods	Differential equations and elem. FORTRAN	3
NA 814 a, b	Advanced Numerical Analysis (year)	NA 413	3, 3

Plus seminars and short courses

Figure I



THE UNIVERSITY OF WISCONSIN
COLLEGE OF ENGINEERING
MADISON 6

DEPARTMENT OF
ELECTRICAL ENGINEERING

Numerical Analysis 132

"INTRODUCTION TO COMPUTING MACHINES"

How computers work; communicating with computers; areas of application and significance; simple FORTRAN programming; elementary data processing and problem solving. Prerequisite, intermediate level of high school mathematics. Offered each semester, 3 credits.

An opportunity for the non-technically trained person to acquire an understanding of the uses, method of operation, and significance of the electronic computer in the world around him. Students will both hear about and actually use the computer in solving problems in mathematics, business, game playing, and many other fields.

The course will be taught jointly by the Numerical Analysis Department and the Electrical Engineering Department, with two lectures and a laboratory period each week. It will be first offered in the fall of 1963.

Figure II

Numerical Analysis 132

"INTRODUCTION TO COMPUTING MACHINES"

Chapter 1	The Information Machine	
Chapter 2	The FORTRAN Language	
Chapter 3	Things Worth Computing:	(A) Routine Repetition
Chapter 4	The Second Industrial Revolution	
Chapter 5	Things Worth Computing:	(B) Try and Try Again
Chapter 6	Further FORTRAN	
Chapter 7	Things Worth Computing:	(C) When Mistakes are Costly
Chapter 8	What Goes on Inside Computers	
Chapter 9	Things Worth Computing:	(D) Hurry, Hurry, Hurry
Chapter 10	The Language of the Machine	
Chapter 11	Things Worth Computing:	(E) Simulating the Real Thing
Chapter 12	Other Kinds of Computers	
Chapter 13	Things Worth Computing:	(F) Memory Like and Elephant
Chapter 14	Things Worth Computing:	(G) Just Plain Curiosity
Chapter 15	Computers and the Future	

Figure III

NA 132

REVIEW QUESTIONS

1. Actual FORGO operating procedure on the 1620.
2. Distinguish between: a) machine language b) symbolic language c) algebraic language
3. What is the difference between "compile" time and "execute" time? How does FORGO mark the transition?
4. Name two (data handling) processes at which the human is more efficient than the computer.
5. Below are listed five characteristics of computers. List three important areas of application that take advantage of each (not necessarily mutually exclusive).
   a) high speed
   b) accuracy (freedom from mistakes)
   c) repetetive ability
   d) large non-forgetting memory
   e) logical ability
6. What are "pseudo-random" numbers? What good are they?
7. How are computers used in inventory control?
8. What trends are observed in business uses of computers?
9. How are computers used in product design? How does this compare and contrast with automation?
10. What is a "critical path"?
11. When is a computer operating in "real" time? Illustrate.
12. Elementary binary arithmetic.
13. Why is binary arithmetic used?
14. What are the five main functional units in a general purpose digital computer? Diagram then, showing the principal paths of data flow and control.
15. What is meant by a single address computer? two address? one-plus-one address? three address? Give an example of each type of instruction.
16. Compare analog and digital computers.
17. Name the three main logical components of an analog computer. Which is used to get distance from velocity? How?
18. How do you program an analog computer?
19. Why are libraries concerned with "Information Retrieval"?
20. What are some of the problems in attempting to upgrade the intelligence of a computer?

Figure IV

COMPUTER CENTER

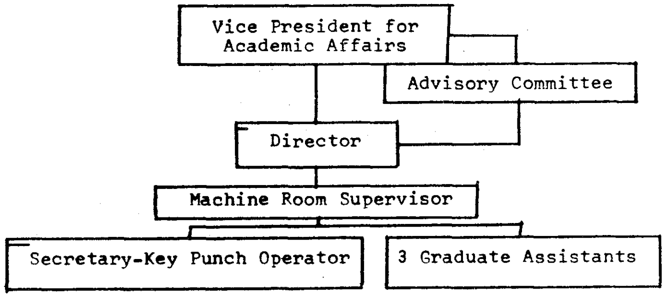
Western Michigan University

The Computer Center was established in Room 372 of Wood Hall in August 1962. Professor Jack R. Meagher was appointed Director.

The Computer Center is organized as a University-wide service, like the University Library, to provide research, training and service facilities for faculty, staff and students. A basic policy of the Center is to encourage widespread interest and use of all its equipment. Highly technical knowledge is not required. Information concerning the use of the Center's equipment is being prepared and will soon be distributed.

An Advisory Committee, consisting of the Director of the Computer Center, the Dean of the School of Graduate Studies, and nine other faculty member's have been appointed by the Vice President for Academic Affairs. This committee will (1) be representative of the whole University, (2) present the Computer Center's operations to the University, and (3) establish broad, general policy for the Center.

The following diagram portrays the Computer Center organization:



FORTRAN WORKSHOP

Jack R. Meagher

MARCH, 1964

The following equipment has been installed in the Computer Center at Western Michigan University:

IBM 1620 Central Processing Unit - (20,000 positions of core storage, a console panel, and an input-output typewriter.)
Automatic Divide
Indirect Addressing
Table Protection
Additional Instructions
Floating Point Arithmetic

1622 Card-Read Punch

1623 Storage Unit (additional 20,000 positions of core storage)

In addition to the IBM 1620 Data Processing System, which is an electronic computer system for scientific and technological application, the Computer Center has the following auxiliary machines:

2 Key Punch: (026) A basic machine for transferring data to punched cards. It also can print the punched data on other cards.

Sorter: (082)- with Counting Unit) This machine sorts cards into a number of pre-selected categories. The counting device tabulates the number of cards in each category.

Collator: (077) The basic function of the collator is "filing". It is capable of making comparisons between decks of cards; and then merging, selecting, or checking the sequences of the cards.

Reproducer: (514) This machine can produce cards that have been previously punched. It can duplicate the original pattern or select and/or rearrange the punched pattern.

Interpreter: (552) This machine prints on the face of cards the data that is punched in them. It facilitates reading and editing the cards.

Accounting Machine: (407) The Computer Center will use this machine primarily to print the input and output data of the computer. This machine has many other capabilities.

A. **Mathematics Department:**

Fortran Workshop

A short, intensive, no-credit course for 20 clock hours. The workshop is non-technical in nature, and has no pre-requisite. The purpose of the workshop is to teach the fundamentals of Fortran programming. This workshop is offered each semester and each summer session.

Programming for Computers - 506

Organization of, problem preparation for, and general use of, high-speed computing machines from the point of view of scientific and engineering computations. Flow charts and programs will be prepared for problems such as: social security, square root, quadratic equation exponential, multiplication of matrices, solution of polynomials and correlation. Problem will be done in machine language followed by the use of a compiler (Fortran). Boolean algebra. Integration of one ordinary differential equation numerically. Pre-requisite: Calculus. This is offered every semester.

Numerical Analysis - 507

Numerical methods as applied to matrix inversion, sets of linear equations, linear programming problems, eigen-values and eigen vectors. Integration of ordinary differential equations and integration of partial differential equations will be presented. Pre-requisite: Math 530 (Vectors and Matrices)

B. **Business Administration:**

Integrated Data Processing - 359

A survey of mechanical and electronic data processing methods with particular emphasis on the application of the electronic system and with special reference to administrative problems experienced in introducing computer systems.

Introduction to Management Science - 554

Modern scientific techniques used in business and industry for controlling operations, maximizing profits and minimizing costs. Allocation of men, money and machines among alternative uses. Other strategies and control methods applicable to management, marketing and finance. Preq.--a course in Statistics.

Electronic Data Processing Seminar - 555

Examination of current literature in electronic data processing with special emphasis on systems analysis, applications of computers to business problems, and feasibility studies. Pre-Requisite: A Computer Course or Consent.

Exercises Assigned in Math 506 Programming for Computers

1.	Volume of Right Circular Cylinder	Machine Language
2.	Social Security Problem	Machine Language
3.	Square Root by Newton's Method	Machine Language
4.	Ouadiatic Equation	Machine Language
5.	Volume of Right Circular Cylinder, Use Sense Switch to Compute $V=\pi r r h$; $V=\pi r^2 h$; $V=\pi r^2 . h$;	Fortran
6.	Ouadratic Equation (Use Hollerith Statement)	Fortran
7.	Given the coordinates of two line segments, find coordinates of the points of intersection.	Fortran
8.	Evalcate e^x	Fortran
9.	Multiply two matrices	Fortran
10.	Find the Inverse of Matrix	Fortran
11.	Solve a cubic equation	Fortran
12.	Correlation Coefficient of X,Y,X	Fortran
13.	Solve a system of simultaneous equations	Fortran
14.	Correlation Coefficient up to and including 20 variables.	Fortran
15.	a. Calculate an integral by Trapezoidal Rule; Calculate an integral by Simpson Rule b. Solve a differential equation by the Runge Kutta Method.	Fortran

The following is a resume of my talk presented to the joint meeting of the Canadian and Midwestern Regions of the 1620 Users Group in Chicago, February 21, 1964:

Miami University, having no engineering school, has concentrated its computer education courses in the new Department of Systems Analysis. This department functions as both a degree granting department and a service department for other University departments.

In its role of a service department, a course in 1620 FORTRAN (2 credit hours) is offered each semester and during the summer term. In addition, students from other departments are free to take any Systems Analysis courses offered provided they have the necessary prerequisites (proper mathematics background in most cases).

For majors in Systems Analysis, two alternatives are offered, business or scientific. In either case, the first two years are devoted to programming, computer analysis, and an introduction to systems analysis. The programming progresses from machine language, to assemblers, and then to the various compilers. The third and fourth years are devoted to the tools of analysis where all examples are worked on the computer.

The Systems Analysis courses offered are:

Introduction to Systems Analysis I and II
Computer Analysis I and II
Systems Design and Selection
Linear Programming
Analog and Hybrid Systems
Operations Research I and II
Simulation and Model Building
Dynamic Programming
Advanced Data Processing Applications I and II
Management Science

Commercial majors are required to take some business and accounting courses as well as 20 or more credit hours of mathematics. Scientific majors are required to take some physics as well as 30 or more credit hours of mathematics.

LAWRENCE J. PRINCE

KANSAS STATE UNIVERSITY

MANHATTAN, KANSAS
66504

Department of Mathematics

Physical Science Building

February 27, 1964

The following is a brief outline of my talk at the 1620 Users' Group (Panel on Education), February 21, 1964, at Chicago.

Kansas State University, a Land Grant school, has approximately 9000 students in a wide variety of curricula. Our basic computer course, 2 hours credit, offered every semester, has an enrollment of 100 to 140 every time. Enrollment will be larger since the Engineering School is making the course mandatory. We teach peripheral equipment, then 1620 and 1401-1410 series. We stress flow charting, then some machine language, some S.P.S., and then Fortran. We use McCracken's book as text, with IBM manuals, and we recommend Germain's book as well.

The Business College uses Schmidt and Meyers as a text.

We teach Scientific Computing Techniques, requiring differential equations and the basic course as prerequisites. Our text is Ralston and Wilf. In addition we have a number of computer-oriented courses such as our Numerical Analysis I, II and III and certain courses in Network Logic, Components, etc., taught in Engineering.

Our staff consists of four regular faculty members, with half-time computing center appointments, a number of graduate assistants each one-quarter time, 2 1/2 key-punch operators, and a machine operator.

Yours very truly,

S. Thomas Parker, Director
Computing Center

STP:nv

Resume of talk delivered at 1620 Users Group February 19-20, 1964

by Roy F. Keller, University of Missouri

The Computer Research Center of the University of Missouri is responsible for both educational and research computing. The computing facilities utilized are three (3) 1620 computers with one disk drive each and a 1410 system. The 1410 is used primarily for business and hospital administrative activities and assembling of medical records. One 1620 is set up to serve most of the educational activities.

At present the Wisconsin FORGO system is primarily being used for student problems. We hope to put FORGO into the monitor system. I understand Wisconsin is doing so, if possible, we will use their system.

Courses presently being taught are:
in Mathematics Department

Math 104 - Fundamentals of Programming Digital Computers
Math 323 - Numerical Analysis
Math 423 - Advanced Numerical Analysis in engineering
Fortran Programming
Eng. 304 - Engineering Problems

and in the Business School

Accounting 101 - Commercial Programming (COBOL) (1410 is used).

In addition to the above formal courses we have a series of lectures and workshops to acquaint faculty and graduate students with computing techniques. Both programming and use of library routines are discussed in these lectures and workshops. One of the most important requirements for a good educational program is to acquaint the faculty with computing.

A SURVEY OF THE BEGINNING PROGRAMMING COURSE

Clarence B. Germain
College of St. Thomas
February 20, 1964

Last Fall, a questionaire was sent to the 280 schools which are members of the USERS Group. 175 schools responded. The results are tabulated on the following pages.

1. No allowance has been made for non-respondents. This does bias the results.
2. Since the survey covers only schools having 1620's, the figures for the end of 1964 do not reflect the influence of schools which will acquire their first 1620 during the year.
3. A suprising number of respondents gave incoaistent answers; e.g., they indicated floating-point hardware, but not divie hardware, or they indicated that 35% of their students run their own SPS programs, while they taught SPS only to 20% of their students.
4. Figures for index registers, binary capabilities, and the 1627 plotter may not be indicative since the questionaire was circulated too soon after announcement of these features.
5. Average enrollment in the beginning programming courses in 170 students per school per year.
6. Many of the Model II 1620's will supplement existing Model I's, not replace them.
7. Relatively few schools indicated any plans to obtain the 1443 printer.
8. The disk units will more than double in popularity during 1964 with 1/3 of all schools having at least one disk unit by the end of the year.
9. While 3% of the schools offered no course involving Fortran, 35% of the students were taught more than one version of Fortran.
10. At the end of 1963, 51% of the schools had the hardware necessary to run Fortran II; by the end of 1964, this figure will rise to 59%.
11. 85% of the students get "hands on" experience in running their own programs on the computer. This percentage is about the same regardless of what programming systems (SPS, GOTRAN, etc.) are taught.
12. Jim Moore's Multi-Trace, 1.4.003, was the most commonly mentioned trace program taught to students. However, 85% of the schools indicated that they used no trace program in their courses.
13. The figures for textbooks are for use in at least one course. Many schools use more than one text in a course. 31% of the schools use only IBM publications as texts. While a wide variety of texts, many unrelated to either Fortran or the 1620, are in use, only four commercial texts and a half-dozen IBM publications are used with any frequency. Of the non-programming type texts, numerical analysis books, particularly Stanton's, were most often mentioned.
14. The textbook percentages in no way indicate sales of books; these figures are quite different from the percentages shown here and were not a part of this study.

RESPONSES OF 175 SCHOOLS TO A SEPTEMBER 1963 QUESTIONAIRE

Results are given as a percentage of the number of schools replying to the questionaire. Probable errors do not exceed ±3% except for items marked with an asterisk (*) where the probable error is less than ±8%. Results are given for the end of 1963 and for the end of 1964. Changes for 1964 are only for equipment now on order. Slight discrepancies in the percentages are due to rounding.

1620 Model:	1963	1964
I	98%	89%
II	2	11
Special Features, Model I		
AFP, Div, IDA, Edit	31	31
AFP, Div, IDA	3	3
AFP, Div, Edit	0	0
AFP, Div	1	1
Div, IDA, Edit	31	31
Div, IDA	14	14
Div, Edit	1	1
Div	3	3
IDA, Edit	1	1
IDA	3	3
Edit	1	1
No special features	13	13
Summary:		
Automatic Floating-Point	34	35
Automatic Divide	82	82
Indirect Addressing	82	82
Additional (Edit) Instructions	64	64
Storage:		
20K core, no disk	48	38
40K core, no disk	21	18
60K core, no disk	17	13
20K core, disk	5	12
40K core, disk	4	9
60K core, disk	5	9
Input-Output:		
Paper Tape only	4	4
Paper Tape and Cards	10	10
Cards only	86	86
Magnetic Tape	4	4
Paper Tape	13	14
Cards 1622-1	83	81
Cards, 1622-2	13	16
Cards, RPQ to read 800 cpm	3	3
1443 Printer		8
Disk, one or more	14	32
1627 Plotter	4	4
1710	2	3

Number of 1620's in the school:	
One	95%
Two	5
Special Features, Model II (1964)	
Automatic Floating-Point	65*
Index Registers	0*
Binary Capabilities	5*
Installations with Printer (1964)	
No disk	23*
1 disk	15*
2 disks	54*
3 disks	0*
4 disks	8*
Type of Courses Offered:	
Both credit and non-credit	5
Non-credit courses only	36
Credit courses only	13
No answer or no courses	47
Departments which offer courses:	
Engineering	40
Education	1
Mathematics	45
Business	31
Other	40
Subjects Taught:	
Machine Language	32
Operation of the Computer	66
SPS	29
GOTRAN	17
FORTRAN with FORMAT	47
FORTRAN II or II-D	33
FORGO, etc.	35
Use of some library trace	13
Block Diagramming	63
Monitor I	9

Disks:		
No disk	86	68
1 disk	8	20
2 disks	5	11
3 disks	0	0
4 disks	1	1
Hardware necessary to run:		
Fortran II only	37	29
Fortran II and II-D	9	19
Fortran II-D only	5	11

Required or recommended texts:

IBM Publications	
1620 Reference Manual	74
1710 Reference Manual	4
SPS Reference Manual	49
GOTRAN Reference Manual	22
1620 FORTRAN Reference Manual	61
1620 FORTRAN II Bulletin	38
FORTRAN General Information Manual	23
1620 Program Writing and Testing Bulletin	12
Introduction to IBM Data Processing Systems	15
Programming and Block Diagramming Techniques	12

Commercial Publications	
Germain—Programming the IBM 1620	27
Leeson-Dimitry—Basic Programming Concepts and the IBM 1620 Computer	39
Gruenberger-McCracken—Introduction to Electronic Computers	6
McCracken—A Guide to FORTRAN Programming	38
Organick—A FORTRAN Primer	38
Colman-Smallwood—Computer Language	6
Smith-Johnson—FORTRAN Autotester	3

Students are expected to write and run their own programs using:	
SPS II	25
GOTRAN	15
FORTRAN with FORMAT	43
FORTRAN Pre-Compiler	28
FORTRAN II	27

DATA PROCESSING TECHNICIAN TRAINING
The Need; The Attempts; The Remaining Void

Computers: Are the nucleus of an extremely large field of operations called Electronic Data Processing. All handling of information within present day large organizations must be done with the electronic computer in mind. The rate of growth of that organization, coupled with the rate of development of compact, fast, efficient, economical computers demands that any and all internal operations be designed with the computer as a possible, if not central, theme. Many more smaller organizations enter the field of computer operations each week, to say nothing of the acquisition of electronic data processing equipment on the part of larger corporations. Some of these companies decide to make use of computers because of profit motives. Others make the switch to computers defenisvely because their competition down the block has installed a computer. These basic reasons and many more subtle factors are at work in forcing businesses toward the world of Electronic Data Processing.

Agility, ability, capability and speed of the electronic computers cause complete revamping of internal operating systems. Even the word "Systems" takes on a different significance when used in conjunction with "Electronic Computers." Information processing now has to be looked upon on the "Total Systems Concept," that is, the processing, handling, generation, and analysis of data through a single means; the electronic computer. Small centers of activity in the line of information travel, and for that matter entire lines of travel have been eliminated by the computer. The Total Systems Concept dictates a complete review and analysis of information requirements, methods of handling data, necessity for various reporting modes, lines of data movement and necessity for generation of information. The computers capability of retention of data in its original format in many cases completely eliminates the need for regeneration centers along the lines of information travel. The speed of operation and mathematical as well as storage ability oftentimes promoted an entirely new system of operation. What before had to be done because of pure necessity could now be accomplished "on line." Even "Empire Building" took a serious setback. These things were

accomplished only if a concerted effort was made to get them accomplished. Only the foolish attempted to incorporate a computer as a single item part of an overall system now operating for the single purpose of getting a single job done. Foolish or not, this has often been done.

Integrated computer operations demanded integrated personnel operations. People, or groups of people within an organization, that before never had any interrelationships now found themselves stumbling over one another wondering what next to be done. The Tabulating Department people on the one hand found that the Systems Department people on the other hand seemed to be working at odd ends with them. The computers promise of the so far intangible brought about a coordinated effort on the part of these two groups. In fact, some data processing people were even caught taking courses in Systems Analysis and vice versa. Most major computer installations today are made up of two categories of personnel, Data Processing and Systems Analysis, working together quite harmoniously. Their accomplishments have been fantastic to say the least. But something is definitely lacking; a something which could and would provide so much more; a something which could and would reduce computer operational cost quite significantly. That something is a single brain thinking and working an integrated Data Processing, Systems Analysis approach.

Simple economics demands that we now find personnel with both the Data Processing and Systems Analysis training and experience. The shortcomings of single field training when considering overall computer management are becoming more and more significant every day. There are several reasons for this, first, single field training limits an individual's approach, definition, and solution selection to a problem. Secondly, reduced cost of actual equipment negates the advisability of employing two to do the job of one. Thirdly, because of cost of operations, vision as well as practicality is required in the selection of fields of computer applications. Fourth, the smaller organizations cannot maintain high computer overhead cost. Fifth, from a profit standpoint, efficiency must be maintained in direct proportion to capability of the equipment involved.

The demand for proper, effective training, therefore, is fantastically greater today than it has ever been in the past and it continues to grow more and more with each computer installation. Far too often we in industry find that our present staff is not adequately capable to provide our managements with the desired level of results from a computer operation. All too soon, we realize that the answer lies in integrated training, formally applied, and practical experience. All too soon, we realize that there is no source of supply for this category of individual. Training, therefore, becomes a major problem to us.

Proper adequate training is the focal point of the single major problem of industry in the field of Electronic Data Processing. Training in the practical way of doing something has always been accomplished by a given organization within its own environment. Different companies within the same industry had much different ways of accomplishing the same thing, and, therefore, each organization trained its own personnel in its own way. Naturally, industry relied, and still does rely, upon formal education to provide the basic general concepts as well as related principles. Industry solved its training problem for the most part by setting up In-house, On-the-Job training programs designed to get the most for the least cost. Many times these programs were extremely limited in scope simply because of the fact that industry chose to use personnel with extremely limited backgrounds. Much of the reason for this was a lack of willingness to pay higher wages for lesser scale jobs.

Profit motive restricted training to a MUST level and added none of the frills of peripheral, or related areas of training. The objective of "Adequate Button Pushers" seemed to be the most efficient level of in-house training. In reality, this is all that industry should be responsible for providing. In all categories of jobs, this in-house training was enough to allow profitable operation. Most jobs did not require a great deal of knowledge, if any, about the last operation or the next operation in the line of process. It was only necessary for the individual to know his or her own special function.

Until the computer came along this was a fairly satisfactory method of training. The O-J-T approach was profitable from two standpoints, first, few dollars invested, second, appreciably high production after training. Without having to be cognizant of the previous operation or the next operation to come, an operator could attain maximum production efficiently and quickly. This was true also in the Data Processing field. For the most part, training of personnel was accomplished in relation to operation of a specific machine or group of machines. Key punch operators learned to simply key punch and verify. Tab operators learned the operation of a series of machines, and in some cases also learned to wire control panels for these machines. From a training standpoint, these data processing jobs were just as any other job in any company, i.e., learn a specific process for a specific operation without consideration of any related areas.

Computers brought about the necessity for training even more categories of personnel. Systems analysts, programmers, coders, program librarians, information librarians, console operators, became new and imposing personnel categories. New, because they came in the front door just ahead of the computer. Imposing, because for the first time a job category came along that demanded an acute interest in what has happened before "this operation" and what will happen after "this operation." No longer could an "operator" be trained to simply accomplish a single operation. Some system of training had to be developed which would allow these new categories to be trained in a reasonable period of time and at reasonable cost. The on-the-job training approach again was utilized by industry in conjunction with short term courses put on by the manufacturers. Initially these two media proved to be just adequate. The only reason that they were adequate was that the personnel originally selected to enter the training programs, and then to handle the computers, were personnel with long experience with the company involved. The fact that they did have the company experience, and, therefore, knew the internal aspects of the organization quite intimately, allowed them to solve most all major problems without too much loss of time or money.

At the same time, these personnel were garnering the necessary hands-on experience on the computer involved. This period of time has been a substantial one. Some of the early computers delivered to the industries came along in the early 1950's. The personnel selected for the computer operations did through trial and error, and OJT methods, finally attain a degree of proficiency in the handling of both the computers and the information involved. However, this took several years of hard work and a severe compromise of original goals and target dates.

Normal attrition for various reasons dictated that an adequate attempt be made in-house to train replacements. The major problem here was one of time. Industry could not afford to invest three to five years of training for each of the replacements. Job attrition rate was far beyond this replacement rate. Some fast means of training was absolutely essential. Here the question was raised, "Do we do this in-house again, or do we go outside for our supply of people?" Obviously, the in-house training cost made another source of supply more desirable. What then was the source to be? Would the apparent returns of employment draw very many capable people into private training operations? Would, in turn, the private training operations provide adequate levels of reliable training for reasonable costs? Did any training facility, private or public have the necessary instructors available? What program would the industries have to undertake to attain good levels of education in the public school systems? Most major industries gathered together for the purpose of determining some of the answers to most of the questions.

Many answers were found, and, as a result, many proposals by industry were made to education. All of them were rebuffed for various reasons. In some cases, education administration personnel were willing to take on the responsibilities of these new requirements but could not find faculty to staff such an endeavor. More often than not, a negative attitude on the part of administration personnel in public education resulted in industry going back to costly in-house methods. People in universities and colleges considered the field to be a vocational one and not an academic one, and, therefore, it could not be touched

with a "ten foot pole." The people in vocational schools were all to willing to accept the challenge, but completely negated the requirements of industry by telling industry that their desires for anything beyond the operator status were ridiculous, and, thereby, vocational education fell down on the job. Industry then racted in the negative quite violently. All proposals to education were withdrawn and in-house, expensive, time-consuming training was put into effect. In some of the more comprehensive of these programs, it was not at all uncommon to find the cost of training at the $30,000.00 to $40,000.00 mark. Time stretched out from the normal one or two month period to somewhere in the area of three to five years, with all its built-in ramifications.

Public education finally entered the field of Data Processing on a late, meager, but welcome basis. First of all, the educators had to be educated. Those administrators and faculty personnel of various public school systems who desired entry into the field of Data Processing had to leave their posts full time or at least part time to get a bit of education themselves. The problems involved here were many and varied. The original estimates on the part of both manufacturers and users that at least a college degree in mathematics was an essential prerequisite to computer operations was impressive and many university and college personnel took this to heart. Leaves of absence were granted to some very few PHD Mathematics type people willing to expose themselves to the rigors of the industrial world. Others went to the manufacturers' short courses in specific operations to acquaint themselves with some of the computer requirements. This training soon became pretty much of a bandwagon effort. If the education people were at all interested, which they probably weren't, they tried to get on the bandwagon. Somehow they got themselves exposed to a course or two, and to a lot of conversation so that they at least knew the terminology of the field. Because of the influence of the manufacturers initially, almost all first comers from education to the field of Data Processing were in the mathematics area. To converse with these people on their own level, an attempt was made to train them in the finer arts of computer utility by means of mathematical problem solution.

This, indeed, served its purpose, but it also had some very undesirable results. The first and most important of these was to indefinitely postpone industry's desire of Data Processing training. The reason was quite obvious, of course. Mathematicians would naturally set up mathematics type courses first, and perhaps other courses later.

Educators from other fields gradually came to either industry or the manufacturers to gain some insight into the world of computers. Some went to the manufacturers for their short courses in specific operations and specific machines. Others left the field of education completely and entered industry bent on learning all they could about the entire field of Data Processing, and then returned after several years to education to set up courses of their own in their own professional fields. Those that did return to education found themselves beset with obstacles sometimes insurmountable. Of these, many became disillusioned quickly and again left education. Of the few that were left, only a handful persisted in the efforts to establish the desired courses. The remaining group soon diverted their attentions from setting up courses desired by industry to setting up the courses which their particular administration happened to think fit well into the scheme of every day living. For the most part, these turned out to be personal research project type courses, unproductive and invaluable to the students that made the mistake of taking them. Education became quite wrapped up in the business of trying to get some decent courses established. Intermingled with this was the personal desire of the particular individual instructor, and an unending avalanche of propaganda from any and all sources.

Gradually, industry has lessened its requirements in the selection of personnel. This is due in most part initially to the fact that people with training were simply non-existent. As industry accepted lower scale personnel, it became apparent that perhaps a Doctorate Degree in mathematics wasn't quite the most essential single pre-requisite. This feeling has some how permeated into the education field until today some of us feel that maybe even a lowly college freshman just might have a chance of understanding computers, providing, of course, that

he first attain 20 years of experience and four college degrees.

There is still a basic contradiction in the approach to the problem by educational personnel. The zeal on the part of many educators to accomplish personal goals, as concerns computers, has caused many of these people to lose sight of their original purpose. Industry requests well-defined, practically oriented courses of training. Education has come up with a hodge-podge of one-quarter courses which are, for the most part, unrelated and unguided. Some universities offer only one section of one course each year, and that course is usually nothing more than an introductory type of course. FORTRAN is often taught as the means of solving business problems. Systems courses can usually be defined as courses in machine capability rather than in their true light.

This has been due in great part to the shortcomings of computer education of the educators. For purposes of quick exposure, each learned about computers in his or her own field. The mathematician learned how to solve math problems and never talked to the business department. The business people learned how to solve the accounting type problems and never talked to the engineering department. The engineering department people learned to solve the standard stress problem and never talked to the science department. The science department people learned to do some of their work on the computers and forgot about the rest of the campus crowd completely. Many and varied requests for equipment came to the administrators from all of these groups. Noan of these were coordinated nor even exposed to the scrutiny of any other departments. People on the administrative staffs were rather prone to allowing each and every department to function independently as in every other facet of their operations. In many cases, it was a first-come, first-served type of operation in the acquisition of a computer. The results were, of course, chaos in the selection, ordering and installation of both the "hunk of junk" involved, and also the courses decided upon by the powers to be to be offered. As is usual, a tremendous amount was left to be desired in setting up Data Processing education in our public school systems.

The evolution of computer and data processing courses followed an almost identical path in every case. First came the "hunk of junk." After some semblance of study within a particular department, a computer was ordered and installed. In general, there was very, very little coordination between the various university departments relative to how the computer could be used. In general, there was always the stipulation on the part of the administration that it was to be used also for administration purposes. I am tempted to wonder if this is also true of the microscopies of the biology department, and the typewriters of the secretarial department and the football shoes of the athletic department.

Next came the exciting question, "What am I going to do with this crazy adding machine?" The individuals involved in the overwhelming task of convincing the administration to acquire a computer had spent all their time in just that, and no time at all in the practical development of courses to be installed after the computer had been installed. The short space of time left in between final order date and installation date of the computer was not at all adequate for developing a good course of instruction on the machine. Result: an unrealistic approach of "Getting something together before the administration finds out." For the most part, this turned out to be a hurredly put together FORTRAN course built on the notes from the instructors own attendance at a manufacturer's short course. In some cases, these even proved to be enough to get through a full semester course. The main problem with this was the fact that this method was actually being used. Under the pretext of being too busy with other aspects of educational life, usually because the individual himself was still quite in the dark about what actually a computer was all about, the students were thrown the bone of sample problems from the manufacturer's course, while the Prof. went on his merry way trying to find out for sure just what did happen when a multiply command was given. Generally, being unfamiliar with a computer language, led to the development of many courses being put together on nothing more than machine language. Machine language and FORTRAN became the ever present by-laws and by-words. Many, in fact most, courses began and ended on this level.

Of particularly significant interest to industry was the method in which the computer itself was handled within the school operation. There seemed to exist therein a fervent desire to restrain the student from ever having any contact whatsoever, except by reference during a lecture, with the computer. Perhaps the administration was fearful of possible repair costs; perhaps the Profs. involved were afraid that the students would find out which were the right buttons to push before the Profs. did; at any rate, the actual operation of the computer center was built around a selected staff of graduate students who did all the actual operations, processed any and all student programs, and laughed rather hideously when a logic error appeared in a student's work. The said part of all this was that the students never did get to find out what the computer really did look like, nor what it did while in operation. On-line diagnostics and debugging techniques were never even mentioned to the students for fear that some questions might be asked. Sadly enough, some of this was justified simply because of time. It would be impossible to cram into a single quarter course any more than an exposure to a language such as FORTRAN and expect the student to get as far as writing a single program. In a semester course, he might be expected to write two short programs, punch them into cards or tape and just maybe get them into the computing center. After all, this was more than the Prof. had accomplished at the manufacturer's short course in slightly less time. It is possible to accomplish only so much in a 60-hour quarter or 90-hour semester, and this was the only interest originally--the one quarter or one semester course.

Unfortunately, this is still true of almost every college and university computer center in the country. Today's offerings in courses actually amount to nothing more than a conglomeration of odds and ends which, for the most part, reflect only the lack of knowledge on the part of those setting the course up. The lack of truly diligent effort is a thorn in the side of Education.

Just as in industry, the computer on campus became a status symbol. Many schools began to wear it as a badge of some sort. Other schools without a computer soon found themselves in a race to the wire in acquiring a "hunk of junk" and getting it into operation. It became very fashionable to be setting up "Computing" centers; computing centers, not computer centers. The terminology was indeed indicative of understanding on the part of both faculty and administration people as regards the computer's place in industry. It was just another machine for the solution of a particular problem, preferably a mathematical problem. The true implication of the computer in the modern business and scientific world was realized by only a very few across the country. These fee comprehended the computer's ramifications, but they made the significant mistake of placing the "hunk of junk" on the "graduate school level only," and, thereby, effectively eliminated almost all students. The result could only be that of ineffective application insofar as the general situation was concerned. Those most in need of exposure to computer education could not be exposed simply because they never got into graduate school.

Again, we have the situation of a Machine language or a FORTRAN course for under-graduates, wherein a program was limited to being written, passed through the keyhole to the graduate student and the garbage results received back through the keyhole with a note telling the poor freshman to do it over again. In most cases, even the graduate student on the inside never got to run any diagnostics of debugging on the program simply because he had never been taught to. However, they did get to help the Prof. while the Prof. was going about getting his own computer education, so they were able to garner some extra tidbits of knowledge regarding computers.

This is where we stand in academic education today. The requests of industry have been forgotten or relegated to the area of unimportance, and most on-campus computers have been gobbled up by Profs. doing personal research for doctorate degrees while the only effective courses are on the graduate school level. In other words, almost total ineffectiveness.

The attempts of vocational schools and area technical schools have also been ineffective. The traditional approach of the vocational school in training for a specific skill has resulted in just another series of "operator" type courses which industry has been able to supply for many years. Vocational schools have been successful to some degree in supplying industry with adequate input trainee personnel for specific job categories in the operator jobs. These personnel still had to complete in-house, on-the-job training after being hired by a company. True, because of their training in school, the in-house OJT training programs could be significantly reduced in length.

Again a failure to comprehend the true ramifications of computers within a business caused the vocational people to continue on their merry way setting up operator type courses in the various job categories of a computer center. All of these were specific in nature, that is, the Key Punch operator was concerned only with punching cards; the Tabulator operator was concerned only with processing the cards in a tabulator or sorter; the Computer Console operator was taught to push buttons and mount tape reels; and the Programmer was just another individual job concerned with writing instructions for the computer. None of these categories received any additional training or even exposure to adjacent operational areas. Even the previous or next operations were left out of the training courses.

The stress on individual job categories by the vocational school people has relegated the programmer's job to that of just another operator in the cycle of business events. Again, this was due to almost the same factors as those arising in the academic world. Too little education on the part of faculty and administration led to impractical courses being set up. An unwillingness to admit that there just might be a need for education beyond just the "operator" level left all related course training out. In these most advanced schools, there sometimes appeared a course in mathematics to about the level of beginning algebra. Never did we find the area of systems being covered, for this was felt to be unrealistic, and, therefore, academic in scope.

Those courses in "Computer Programming" which were set up followed almost exactly the efforts within the academic world in doing the same thing. Short courses in the use of a computer language such as Absolute and FORTRAN made their appearance, and were handled in virtually the same way. After an exposure to a language, the student was expected to write a program and subject it to the processor by way of either the Prof. of his assistant. Almost invariably, what little hands-on experience that was provided was channeled only into the area of what buttons to push "to get the darned thing to run." All of these courses were set up on the "lab" basis, with little or no solid lecture content. For the most part, the instructor was a converted Business Education type person with a specialty of office machines or typing who had been exposed to a short course at a manufacturer's school somewhere. The results again were very similar to that in the colleges and universities. The principal problem here was a reluctance to admit that any job could have implications or ramifications beyond its own seemingly immediate scope. The administration and faculty people refused to even attempt to deviate from their set ways of education and set up some semblance of an adequate course in computers.

This satisfied the requirements for "operator" type people but left the original problem completely unsatisfied. The so-called computer programmer was nothing more than a language coding clerk, lacking any and all knowledge of how to really put a computer to effective use within an organization. The Systems Analyst dealt only with the abstract it seemed, and, therefore, had no place in vocational education. Because of this, so-called programmer courses could give the student nothing more than a language background. The companies hiring these graduates found themselves right back where they started with their own in-house OJT programs. The vocational schools did effectively provide input trainees for operator jobs, but left much to be desired when it came to effective use of equipment within a Data Processing center.

Computers and their effective use demand an integrated training program. In general, every business computer installation depends upon a staff of personnel which has had a good academic background in a particular field such as Accounting, Engineering, Mathematics, Science, etc., and also a vocational skill training in machine use and operation. Industry has found that it wasn't enough to just be an accountant or engineer, a scientist or mathematician, just a systems analyst or programmer. It wasn't enough to have just a high school education followed by quicky type operator courses. It wasn't enough to have an exposure to a quarter course in a computer language. It wasn't enough to approach a computer with simply problem solution in mind. Somehow the benefits of college or university academic training had to be molded together with hands-on vocational training into a useful level of competency and judgment. The far-reaching aspects of every computer application made it very desirable to provide the type of training which would allow the individual to make the utmost use of a particular "hunk of junk."

During the last two to three years, it has become more and more apparent that more than one academic field had to be included as well as more than one type of computer training in any effective computer course. In addition to accounting, some mathematics, English, and statistics were very necessary. In addition to an exposure to computer programming via learning a computer language, a background in problem analysis and solution, in Symbolic language as well as FORTRAN, in hands-on debugging methods as well as desk checking were all vitally necessary.

Computer operations training was the immediate concern, and primary courses had to be directed toward this end. However, the larger aspects of computer ability had to be made the goal of all training. Somehow the Methods Analyst, Systems Engineer, Computer Programmer, Data Processing Manager, Controller, Accountant, Sales Manager, Production Superintendent, Factory Foreman, Inventory Clerk, Grounds Keeper, and Garbage Collector had to be rolled up into a single individual via an effective training program.

From a practical standpoint, this conglomeration of far apart fields had to be rolled into one, but this isn't quite possible within the short periods of time available for training. Still it is apparent that quite a different emphasis in training is required. Rather than being just another tool or method, the computer has become the center of operations affecting all other departments of an organization. No single operation escaped the inevitable scrutiny of the computer. Training, therefore, had to include all possible fields of endeavor, so as to provide the computer personnel with as broad a scope as possible to enable them to make sound decisions. Exposures to other fields had to be realistic and sound as well as effective, but these exposures had to be made within realistic time allocations.

The pressure of telescoped time has not allowed industry the luxury of retraining older staff members. Though these personnel had extremely good knowledge of the organization, it often required more time than we had available to bring them to a point of competency in computer utility. Additional personnel with "Wide angle" computer backgrounds had to be discovered and acquired. Most of these came from the fields of either Data Processing or Systems Analysis, but even these did not have the proper backgrounds of mutliple field. The greatest deterent to these in aclimatizing themselves to computer operations was professional prejudice. Until computers, these two fields stood at odds with one another. This factor became a serious restriction to further development within the computer field.

The two year Data Processing Technician Course seems to offer some indication of an adequate answer to the problem. The integration of all the required facets into a single applied course can, of course, be the only true solution to the problem of acquisition of trained, capable personnel. This is not going to take place very soon because of some of the reasons already mentioned. In the meantime, some adequate substitute must be found. To ask industry to continue their own extremely expensive training programs is not realistic. In the first place, this is far too expensive a mode of training. Secondly, it is unrealistic in the consumption of time.

Developments in the field of computer utility arrive so fast and in such quantities as to render ineffective any long range in-house, on-the-job training program. Industry training programs are usually based on a set way of doing

things which is expected to continue as the way of doing things for many years to come. In computer operations, this is not at all true. Yesterday's method was ancient history as soon as it was used. Today's method will be outmoded before the job is done tonight. Tomorrow's method is already gathering pale, green mold of disuse.

As an intermediate solution to the problem of providing industry with adequate input personnel, the two-year technician program has been suggested and put into use. These, too, have fallen into the same traps for the same reasons as the academic courses and the vocational courses. The reason for this is obvious. The very same faculty and administrative people are making the very same unintelligent decisions about this course curriculum and setup.

During the last two to three years, there has taken place a great rush to "get on the Data Processing bandwagon." Unfortunately, the depth of sincerity here seems to go only that far. Again we find unacquainted administrative personnel and untrained instructor personnel attempting to set up and teach a totally unfamiliar curriculum. For the most part, these attempts have been made in junior colleges and vocational schools on a post high school basis. In almost every instance a basic two year curriculum, suggested by several of the manufacturers and approved by the U. S. Department of Health, Education and Welfare, has been the sum and substance of these attempts. These suggested curriculums had been set up on a purely theoretical basis, and had not been put into practical operation before being foisted upon an unsuspecting education group. Perhaps in awe of the suggesting body, perhaps for lack of personal knowledge, perhaps for desire to be "one of the bunch," education accepted these curriculums and attempted to do their best at training people in Data Processing, without first doing any extensive exploration of the curriculum.

This is the point at which the most significant of failures occurs. Scrutiny of the suggested curriculums by those in Data Processing management positions in industry would perhaps have pointed out some of the most glaring inadequacies. Some of these are: first, a purely theoretical approach to subject matter without adequate experience emphasis; second, basic courses being taught last in sequence, with advanced courses coming first; third, emphasis on language communications with the machine rather than a solid foundation in machine operation characteristics; fourth, relagation of the Data Processing courses to a status of just another course rather than a status of core course.

As an example of this, let us examine the most commonly found curriculum. It is set up as follows:

First Year	First Semester Credit Hours	Second Semester Credit Hours
Data Processing Mathematics I and II	2	4
Accounting I and II	4	4
Communications I and II	3	3
Basic Computing Machines	3	
Unit Record Machines	5	
Data Processing Applications		3
Introduction to Programming Systems		4

Second Year	Third Semester Credit Hours	Fourth Semester Credit Hours
Computer Programming I and II	5	5
Social Science	3	
Statistics	3	
Business Organization	3	
Cost Accounting	3	
Systems Development and Design		3
Advanced Computing & Programming Systems		3
Data Processing Field Project		6

An examination of the content of each of these Data Processing courses generally reveals the following course descriptions:

BASIC COMPUTING MACHINES. This is a survey course of common fundamental concepts of data processing systems. It describes the evolution of computer systems--from manual to stored program methods.

UNIT RECORD MACHINES. This course is a survey of unit record equipment. It illustrates the need for machine-processable solutions to accounting and recordkeeping problems. Laboratory problems include wiring of control panels.

DATA PROCESSING APPLICATIONS. This course is designed to acquaint the student with actual business data processing applications. The student learns through lecture and case studies to apply the data processing equipment previously studied to

various applications. Through this study, the student gains an understanding of how machines and systems are combined, and the advantages of mechanization.

INTRODUCTION TO PROGRAMMING SYSTEMS. The basic concepts of programming systems are taught in this course. During this class, the student becomes aware that programming systems are as important as the machine hardware. This course familiarizes the student with the purpose and function of the various types of programming systems or languages.

COMPUTER PROGRAMMING. The Basic Computing Machines course in the first semester provides the theoretical basis for detailed study of data processing machines in this course. Programming drills, case studies and exercises serve to bridge the gap from the theoretical to the practical use of data processing. The two hour per week laboratory sessions provide further reinforcement of basic principles by providing "hands-on" training. The FORTRAN language is the main media of training, while the student is introduced to Machine Language and Symbolic Programming Systems.

SYSTEMS DEVELOPMENT AND DESIGN. A survey course discussing the effective use of data processing equipment in meeting the information needs of business. The course is designed to guide the student through the various stages of system development.

ADVANCED COMPUTING AND PROGRAMMING SYSTEMS. The objective of this course is to provide the student with sufficient knowledge of programming language concepts so that he may easily master any specific system with a minimum of instruction. Actual programming languages are not taught. However, individual phases of certain selected language systems are treated in detail in order that the student may learn advanced programming language techniques contained in sophisticated systems.

Let's examine these courses a little more minutely from the standpoint of "what is being taught when." In the first semester we have two parallel courses, Basic Computing Machines and Unit Record Machines. The Basic Computing Machines course is almost always a pure theory course examining the various styles and types of machines. Little or no effort is made to acquaint the student with the methods of problem solution required by each of these types of machines. Since this is usually a three hour per week course, there is really no time to go beyond a term definition stage of learning. The Unit Record Machines course is a little more definitive in that the student does get some "hands-on" experience in control panel wiring. This, too, is very limited since the five hours per week during one semester must be spread over several types of machines and all the punched card processing philosophy involved in each of them. The student receives a total of about 20 hours per machine type. Certainly not enough even to complete the acquaintanceship status of knowledge.

The simple fact that these two courses are so drastically limited in scope negates their desirability as foundations for further data processing courses. The Basic Computing Machines course should have as its prime objective teaching the student the various methods of problem solution used in the various types of computers. Since problem solution is really the prime consideration of all subsequent courses, and all activities involving computers, it would be far more desirable to teach this topic rather than mechanical configuration and operation in the initial course.

During the second semester, the student is exposed to something called Data Processing Applications. During this course, he "learns through lecture and case study to apply the data processing equipment previously studied" to business applications. This is all well and good, but since the Basic Computing Machines course did nothing more than acquaint him with mechanical configuration, this course will still leave him asking "how, what, where?" This course would be infinitely more appropriate were it taught following some exposure to practical programming experience. At least, then the student would have some means by which he could relate the machines to the applications. Far too many students and graduates come to industry knowing all about computers or language systems, but are unable to put this knowledge to work on a practical application.

Also, during this econd semester, the student is subjected to a course called "Introduction to Programming Systems." This is basically a course which examines the various types of programming languages. It is an exploration of the

"software" wherein the student "becomes aware that programming systems are as important as hardware." Again, this course is pretty much a theoretical one with little or no practical programming experience. For the most part, the languages generally discussed are not a part of the software package of the computer which is installed. Here again, the student is left wondering "what to do with it" as concerns all of these languages.

So far, two full semesters have gone by and the student still hasn't "soloed" on the computer." He has still to write his first practical problem and run it on the computer. It would seem to be an awful waste of time.

Finally, in the third semester, a course called Computer Programming I shows up. Welcome at long last, even though it falls far short of its rightful goal. During this course, "programming drills, case studies and exercises serve to bridge the gap from the theoretical to the practical." This objective is perhaps a sound one, but since the FORTRAN language is generally used, with only an exposure to Symbolic and Machine Language, it is doubtful that it can ever be reached. The FORTRAN language still gives the student no insight into data manipulations within the machine. FORTRAN does not allow on-line debugging and diagnostics. FORTRAN does not even allow the student to develop his own programming logic techniques. FORTRAN limits him entirely to a rigid set of rules evolved from someone else's logic. In those courses where Machine language is taught as the basic media, the student is severely limited by the clerical activity involved, and also by the fact that only relatively simple problems can be attempted. We find, therefore, that in the third semester when the student should be at an advanced level, he is in reality just beginning. An examination of the number of computers installed and in use reveals that approximately 95+% are used in business type operations. Why then use a mathematical language such as FORTRAN, with all its inherent restrictions and limitations, as a training media? Why not give the student the type of training which is going to equip him for 95% of the job market?

In the fourth semester, another course is given in Programming Systems.

This usually is a survey type course designed as an extension of the introduction course given in the second semester. These two courses could well have been combined into a single course during this semester. By the time the fourth semester has begun, the student is just barely capable of making effective use of the high-powered languages usually discussed in this course. The philosophy of operation involved in COBAL, ALGO and some of the other more sophisticated languages are beyond practical comprehension until the student has had an adequate exposure to manipulation of data through the use of some lessor language system. The third semester course in computer programming will accomplish this to a certain degree, but places the second semester course, "Introduction to Programming Systems," in a wasted status.

The "Systems Development and Design" course in the fourth semester is a practical one, though it comes a little late. Far more emphasis needs to be placed on "the effective use of Data Processing equipment in business needs." This theme must effectively be built into each and every segment of each and every data processing course beginning with the very FIRST day of instruction. Without this, the student in his theoretical surroundings loses sight of the practical application of what he is studying.

Because of the inadequacies of suggested and adopted courses, a new approach to course setup and content had to be taken. At Hibbing, we first tried to determine what type of individual was really in demand by industry. It was all well and good for us to listen to the claims of almost everyone that the field of Data Processing was one of unlimited opportunity and a never-ending sponge which would gather up any and all graduates of any and all types of courses. Having spent a few years in the business world, I found this quite hard to believe. We spent a great deal of time and effort finding out what we believed in from the first--that not just any old one-quarter course approach would suffice; that somehow, we would have to begin at the level of a good workable computer language from both the practical and educational point of view and go on from there.

Realizing that we had opened the door to a vast area of extremely hard

work, we went about the task of developing our core courses. After much serious investigation as to where the computers were being used, we decided to place the emphasis of our core courses on business applications. We, therefore, asked our college instructors in Accounting, Mathematics, Economics and English to put together a group of integrated courses sufficient to support an intensive level of training in Data Processing.

All courses developed were to be integrated with the Data Processing courses. The problems discussed in the Accounting and Mathematics courses were to be discussed from an integrated computer operations point of view. The Communications English courses were to emphasize public speaking and technical report writing, with Data Processing applications as topics. These courses were accelerated versions of the same basic courses taught in our Junior College, by the same instructors, but with a much different objective in sight. The level of training, therefore, was intensive, but it was also very well directed along the Data Processing lines.

The Data Processing courses, built around the IBM 1620 computer and IBM 407 accounting machine emphasized a total integration of Computer Programming and Systems Analysis. No problem was to be discussed in the Data Processing course grouping without a thorough investigation of all the systems ramifications of the problem. In this manner, we hoped to be able to acquaint the student to a good degree with all factors involved with a particular type of problem, and not just with "a typical programming"approach. The significance of the several means and methods of problem solution was also brought to the student in the discussion of each problem.

Early examination of the field of data processing showed that a particular type of person was in great demand. This was a person with more than just an exposure to machines and computer programming. This was a person with sound fundamental training in Business courses and more than just an acquaintance with Systems Analysis and Procedures. This was a person capable of evaluating the relative costs of several methods of problem solution and data handling and able to make a sound, profitable judgment and decision. This was a person capable of more than just console operation, or debugging through a list of error codes. This was a person who looked at every data processing problem through two sets of eyes, those of the Systems Analyst and those of the Data Processing Technician.

To achieve satisfactorily the goal of training this person, we put together the following curriculum.

	Hours per week		Semester	Total Semester
Course Title	Lecture	Lab.	Credits	Class Hours
First Semester:				
Principles of Accounting I	4	4	4	144
Data Processing Mathematics	2	2	2	72
Introduction to Computers	3	2	3	90
Unit Record Equipment	4	4	4	144
Communications English I	3	2	4	90
	16	14	17	540
Second Semester:				
Principles of Accounting II	4	4	4	144
Data Processing Mathematics II	4		3	72
Communications English II	3	2	4	90
Computer Programming	8	5	8	234
	19	11	19	540
Third Semester:				
Cost Accounting	4	1	3	90
Human Relations	4	1	3	90
Business Organization	4	1	3	90
Data Processing Applications	4	4	4	144
Introduction to Systems	4	3	4	126
	20	10	17	540
Fourth Semester:				
Systems Development	5	5	5	180
Advanced Programming	5	5	5	180
Data Processing Field Project		10	5	180
	10	20	15	540
Totals	1170	990	68	2160

Beginning in the first semester, our individual Data Processing course content is as follows.

UNIT RECORD EQUIPMENT. This course examines the use of most unit record punched card machines including the key punch, verifier, sorter, reporducer, collator and accounting machines. Heavy emphasis is placed on proficiency on the IBM key punch and verifier, and on complex board wiring for both the 085 collator and the 407 accounting machine. Forms design for both card formats and printed output is also thoroughly covered. This course forms the basis for all future laboratory problems in that the student must wire all necessary control panels for each machine

used in any laboratory problem solution.

INTRODUCTION TO COMPUTERS. This course is the key to success or failure in computer programming. This course is divided into three major segments each of which covers a single basic type of computer. These are Disk-Drum type machine, the Core-card processing machine and the Core-Magnetic Tape machine. The logical approach to problem solution for each of these basic machine types is stressed in this course. Flow charting and detail block diagramming of problem solution becomes the student's central concern. The laboratory portion of this course consists of setting up the IBM 1620 computer for problem processing and of console operations using 1620 user-group library programs and problems for these programs.

COMPUTER PROGRAMMING. This second semester course in IBM 1620 computer programming forms the solid foundation on which all future programming courses are built. This course has the two-fold objective of teaching the student computer programming techniques, and of acquainting the student with how a computer is used in business through carefully developed laboratory problems. The first three weeks of this course are devoted to the Absolute Machine Language of the 1620 computer. Basic techniques of on-line debugging are also stressed during this period. The remaining 15 weeks of this course are given over to the use of the 1620 Symbolic Programming System Language. Business problems of increasing complexity are handled as the course progresses. With each problem, the significance of the computer relative to overall systems and procedures is stressed to the students. The laboratory problems are designed to extend and reinforce the basic computer logic ideas covered in the first semester as well as to develop proficiency in 1620 computer programming.

DATA PROCESSING APPLICATIONS AND INTRODUCTION TO SYSTEMS. These two parallel third semester courses bring to a practical working level all the principles of Computer Programming and Systems and Procedures learned in the second semester. The student continues his Symbolic Programming with each laboratory problem, but he now expands his operations by developing the source and handling of all data for a particular problem. For each laboratory problem, the student develops a complete system of information handling culminated by a complex computer program necessary to arrive at desired results. Complex applications such as incentive payrolls, complete accounts receivable processing including aging, inventory processing and item use and scrap analysis, general ledger and other integrated systems of information handling. The student also designs and develops any and all necessary documentation of the system including both manual and machine operations. The stress in these two courses is on polishing computer programming techniques and acquiring practical system and procedures knowledge.

SYSTEMS DEVELOPMENT given as a fourth semester course continues the third semester Introduction to Systems course with heavy emphasis on the total systems concept and the development of systems and procedures throughout an organization which will support the necessary profitable operation of a computer. Discussions and laboratory problems include practical exposure to PERT and Critcal Path techniques both on and off the computer, in addition to standard systems of Information Retreival.

ADVANCED PROGRAMMING in the fourth semester consists of IBM 1401 computer Symbolic Programming training. Most basic laboratory problems from the second and third semester Programming and Applications courses are reviewed and 1401 programs are written and tested solving these problems. FORTRAN is also taught during this course.

DATA PROCESSING FIELD PROJECT during the fourth semester consists of the student actively engaging in shop operation of a local area data processing department of local industry. During this period, the student selects an information system which he completely documents and programs for the computer. His solution must be one which is better than the system in operation which he selects to study.

The objective of training Data Processing technicians dictated the attainment of a thorough knowledge of computer operations. To achieve our desired end, we decided early in our investigations that we would utilize the Symbolic Programming Language as our major computer instructional tool. Our foundation was to be thorough understanding of internal data handling by use of Absolute Machine

Language. The relative advantages and disadvantages were thoroughly explored and the decision for Symbolic Language was based upon the primary requirement of solid understanding of the internal capabilities of the equipment involved. All laboratory problems were designed with hands-on assembly, testing, debugging, and processing as a significant part of the problem. A requirement for thirty clock hours of solo time on the 1620 computer by the end of the second semester became a mandatory objective. "Console Confidence" could be attained only through personal contact with the machines. Mastery of the FORTRAN Language was also included in the curriculum, but as a secondary rather than primary language. FORTRAN programming, as well as 1401 Symbolic programming was included in the fourth semester area of the course. A course in Systems Analysis was designed which emphasized methods of data handling outside of, but orientated to computers. The how to do it approach of integrated data flow operations was tied in directly to a computer program to solve a given problem. In all cases, the discussions of various problems included and revolved around the ramifications of a computer in the systems area. Topics in the Data Processing group of courses ranged from simple payroll type problems to integrated general ledgers; from production control to sales analysis; from historical records to operational research; from information retrieval to business simulation. All of these topics were arranged with two things in mind; first, the how and why of computer solution; and, second, the method of programming the problem within the computer.

In all computer applications, each student was required to do his own on-line debugging and diagnostics. Each laboratory problem required the submission of a written report on the problems which the student had encountered while solving the problem. This technical report had to include a complete explanation of the programming method used, the assembly and processing techniques, and the Diagnostic and Debugging procedures utilized by the student. The original objective of the overall outlook and integrated problem solution techniques were repeatedly brought to bear in each of the courses. Various systems and procedures of approach were explored with each laboratory problem, and students were each encouraged to attempt different methods of arriving at a solution to the particular problem. No two like solutions were to be accepted in the computer programming phases and systems phases of the course. Individual thinking and exploration was thus encouraged to the utmost throughout the course structure. Failure on the part of the faculty to foster and nurture this aspect of student development would in reality be a failure of the entire course structure. The two year program is not the answer to the problem, and will probably soon be outmoded. The two year program came about as a stop-gap measure necessitated by the inadequate fumblings of colleges, universities and vocational schools. To date, only meager attempts have been made to provide integrated training. Most of these have been of the too little, too late variety. Most courses of any value are on the graduate school level, where most students never appear for training. Courses on the undergraduate level are ineffective because of too narrow scope and FORTRAN type approaches. Much hard work remains to be done in course development before university, college and vocational school programs offer the type of training necessary to provide industry with adequate input personnel. Without this course development work, all such programs will be nothing more than a waste of time and money. Universities, colleges and vocational schools must bring themselves to recognize the fact that computers have become a factual way of life for both organizations <u>and</u> personnel. Without this, Education can never hope to catch up to the world of reality. Where does your program stand?

INTRODUCTION TO MATRICES

Charles E. Maudlin, jr.
University of Oklahoma

Intended for users with no knowledge of matrices and little background in mathematics. What a matrix is, matrix operations, singular matrices, how errors arise in matrix operations. Example of matrix inversion by Gaussian elimination.

In working with simultaneous linear algebraic equations, it seems reasonable to work with the coefficients only. For example, in the system

$$\begin{array}{rcrl} X & + & 3Y & = 4 \\ 2X & - & 9Y & = -7 \end{array}$$

We would expect to find the solution values by some set of operations on the numbers

$$\begin{array}{rrr} 1 & 3 & 4 \\ 2 & -9 & -7 \end{array}$$

Indeed we would expect to find the same solution values if the equations were

$$\begin{array}{rcrl} a & - & 3b & = 4 \\ 2a & - & 9b & = -7 \end{array}$$

It now seems reasonable to make the following definition:

A rectangular array of numbers is called a <u>matrix</u>.

Ex.: $\begin{bmatrix} 1 & 3 & 4 \\ 2 & -9 & -7 \end{bmatrix}$

The size of a matrix is characterized by the number of rows and the number of columns.

$$2 \times 3$$

A matrix consisting of exactly one row or exactly one column is called a <u>vector</u>.

$$\begin{bmatrix} 4 \\ -7 \end{bmatrix}$$

a_{ij} denotes the element in the i-th row and j-th column of a matrix.

$$a_{21} = 2$$

a_i denotes the i-th element of a vector.

$$a_2 = -7$$

$$\begin{bmatrix} 1 & 3 & 4 \\ 2 & -9 & -7 \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}, \left\| a_{ij} \right\|, \left[a_{ij} \right] \quad (a_{ij}), \qquad A \text{ are}$$

equivalent symbols to denote a matrix.

Definition: Two matrices are equal if and only if they have the same size and correspondingly placed elements are equal.

We will define a multiplication process by introducing the concept of linear substitutions to matrices. Suppose we are given the set of equations:

$$\begin{aligned} 3x + 2y + 7z &= a \\ 2x - y - 3z &= b \\ x + y + z &= c \end{aligned}$$

In matrix form this looks like:

$$\begin{bmatrix} 3 & 2 & 7 \\ 2 & -1 & -7 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

For the present, this means:

$$\left[\text{coefficients}\right] \times \left[x\text{'s}, \quad y\text{'s}, \quad z\text{'s}\right] = \left[a\text{'s}, \quad b\text{'s}, \quad c\text{'s}\right]$$

Later we will show that the form used is consistant with our definition of multiplication.

Now suppose further that:

$$\begin{aligned} u + v &= x \\ 2u - v &= y \\ u - 3v &= z \end{aligned} \qquad \text{or in matrix form} \qquad \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ 1 & -3 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad (2)$$

It would not be very helpful to use this thing called a matrix unless we could substitute from equation (2) into equation (1) and obtain

$$\begin{bmatrix} 3 & 2 & 7 \\ 2 & -1 & -3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ 1 & -3 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad (3)$$

We would also like to believe that this represents the same relationships that would be obtained if we had performed the actual substitutions and written the results in matrix form. The steps in making the substitutions are:

$$\begin{aligned} 3(u+v) + 2(2u-v) + 7(u-3v) &= a \\ 2(u+v) - 1(2u-v) - 3(u-3v) &= b \\ 1(u+v) + 1(2u-v) + 1(u-3v) &= c \end{aligned}$$

$$\begin{aligned} (3\cdot 1 + 2\cdot 2 + 7\cdot 1)u + (3\cdot 1 + 2\cdot(-1) + 7\cdot(-3))\,v &= a \\ (2\cdot 1 - 1\cdot 2 - 3\cdot 1)u + (2\cdot 1 - 1\cdot(-1) - 3\cdot(-3))\,v &= b \\ (1\cdot 1 + 1\cdot 2 + 1\cdot 1)u + (1\cdot 1 + 1\cdot(-1) + 1\cdot(-3))\,v &= c \end{aligned} \qquad (4)$$

$$\begin{aligned} 14u - 20v &= a \\ -3u + 12v &= b \\ 4u - 3v &= c \end{aligned}$$

$$\begin{bmatrix} 14 & -20 \\ -3 & 12 \\ 4 & -3 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad (5)$$

We want (3) and (5) to say the same thing. Multiplication will defined so that

$$\begin{bmatrix} 3 & 2 & 7 \\ 2 & -1 & -3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & -1 \\ 1 & -3 \end{bmatrix} = \begin{bmatrix} 14 & -20 \\ -3 & 12 \\ 4 & -3 \end{bmatrix}$$

By examining (4), we can see the manner in which the product matrix should be formed.

(a) Each element of the product matrix is the sum of three products.
(b) Each product contains one factor from the left matrix and one from the right.
(c) Elements of the i-th row of the product matrix are formed from elements of the i-th row of the left matrix.
(d) Elements of the j-th column of the product matrix are formed from elements of the j-th column of the right matrix.

If a_{ij} , b_{ij} and c_{ij} are typical elements of the matrices A, B, and C where A is the original coefficient matrix, B is the substitution matrix and C is the product matrix then

$$c_{ij} = \sum_{k=1}^{3} a_{ik} \; b_{kj} \qquad i = 1, 2, 3 \qquad j = 1, 2$$

In general: IF $A_{r \times s}$ and $B_{s \times t}$ are multiplied, the result is given by

$$A_{r \times s} \; B_{s \times t} = C_{r \times t} \,, \quad c_{ij} = \sum_{k=1}^{s} a_{lk} \; b_{kj} \;, \quad i = 1, 2, \cdots, r \;, \quad j = 1, 2, \cdots, t$$

Notice that matrix multiplication is not defined unless the number of columns of the left matrix equals the number of rows of the right matrix. Even when such multiplications are defined, it is not true in general that AB = BA. Examples:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 3 \\ 3 & 5 \end{bmatrix} \qquad \begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 6 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} -2 & -2 \\ 6 & 8 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 11 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} \text{ is not defined}$$

When the multiplication is defined $\quad A\,(BC) = (AB)\,C$

The proof is omitted here but it can be shown to be true by applying the definition of multiplication twice to each side to determine the typical element.

We can see that our original matrix equations are consistant with this definition.

A square matrix with 1's on the diagonal and zeros elsewhere is called the identity matrix. If there are n rows and columns, it is denoted by In. When no confusion is apt to arise about size, the subscript is dropped.

Thm For every matrix A, AI = A and IA = A (The size of I may have to be adjusted if A is not square).

Proof Let $(a_{ij}) = A$, $(b_{ij}) = I$ then $b_{ij} = 0$ unless $i = j$ and $b_{ii} = 1$.

consider a typical element in the product AI:

$\sum_{k=1}^{n} a_{ik} \; b_{kj} = a_{ij}$ The only term in the summation that survives is the one for $k = j$ (otherwise $b_{kj} = 0$). This leaves $a_{ij} \; b_{jj} = a_{ij} \cdot 1 = a_{ij}$

The other half of the theorem is proved in an analagous manner.

For some matrices A, there exist corresponding matrices B having the property that AB = I. When this occurs B is said to be a right inverse of A (A is a left inverse of B). If A is square then AB = BA = I. B is the inverse of A and is usually denoted by $B = A^{-1}$. Just statement - no proof here.

$$\begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}$$

A diagonal matrix is a square matrix with all non-diagonal elements equal to zero. The identity matrix I is a special case.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -7 \end{bmatrix} \quad \text{is a diagonal matrix}$$

The multiplication of a matrix by a number is called scalar multiplication. The multiplier is called a scalar and the product matrix is obtained by multiplying each element of the original matrix by the scalar.

$$\text{IF} \quad A = \begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{bmatrix} \quad \text{then} \quad 7A = A7 = \begin{bmatrix} 7 & 14 & 21 \\ 42 & 35 & 28 \end{bmatrix}$$

Incidentally, the elements of a matrix are scalars too.

A scalar matrix is a diagonal matrix with all diagonal elements equal.

The identity matrix is a special case.

$$\begin{bmatrix} -3 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & -3 \end{bmatrix} \quad \text{is a scalar matrix}$$

For every square matrix A and every scalar matrix S of the same size AS = SA.

When the terms are defined $\quad (AB)^{-1} = B^{-1} . A^{-1}$

$$(B^{-1}A^{-1})(AB) = \left[(B^{-1}A^{-1})\,A\right] B = \left[B^{-1}(A^{-1}A)\right] B = (B^{-1}B) = I$$

so $(B^{-1}A^{-1})$ is the inverse of (AB)

Addition of matrices is accomplished by adding correspondingly placed elements. It is obvious then that the matrices must be of the same size.

$A + B = C \quad \text{if} \quad c_{i_j} = a_{i_j} + b_{i_j}$

$$\begin{bmatrix} 1 & 2 & 3 \\ 6 & 5 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 0 & -1 \\ 2 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 \\ 8 & 6 & 6 \end{bmatrix}$$

Properties of addition: When the operations are defined:

(1) $A + (B+C) = (A+B) + C$
(2) $A + B = B+A$
(3) $A + Z = Z+A = A$ (where Z is of proper size and all elements are zero)
(4) For every matrix A there is a matrix $B = (-1)\,A$ such that $A+B = B+A = 0$. B is usually denoted by $-A$.

When the operations are defined $A(B+C) = AB+AC$
and $(A+B)C = AC+BC$.

Suppose we are presented with matrices $A_{3\,x\,5}$, $B_{5\,x\,7}$, $C_{5\,x\,7}$ and we wish to determine $D = A(B+C)$. The following computer program will accomplish this.

The matrix B will be lost in the process.

```
C      SAMPLE PROGRAM
C
       DIMENSION A(3,5),B(5,7),C(5,7),D(3,7)
C
C      FORTRAN II  I/O BECAUSE ITS SHORTER
C
       READ 1,((A(I,J),J=1,5),I=1,3)
       READ 1,((B(I,J),J=1,7),I=1,5)
       READ 1,((C(I,J),J=1,7),I=1,5)
     1 FORMAT (5E15.8)
C
C      NOW TO ADD B AND C
C
       DO 2 I=1,5
       DO 2 J=1,7
     2 B(I,J)=B(I,J)+C(I,J)
C
C      NOW TO MULTIPLY A TIMES THE SUM
C
       DO 3 I=1,3
       DO 3 J=1,7
       D(I,J)=0
       DO 3 K=1,5
     3 D(I,J)=D(I,J)+A(I,K)*B(K,J)
C
C      NOW TO PRINT THE RESULTS
C
       PRINT 4,((D(I,J),J=1,7),I=1,3)
     4 FORMAT (5E15.8/2E15.8///)
       STOP
       END
```

Now let's start looking for this nebulous thing A^{-1}. Remember - it doesn't always exist. Suppose we are presented with

$$A = \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 2 \\ 2 & 4 & 5 \end{bmatrix} \quad \text{and we wish to determine} \quad A^{-1} = B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

This is equivalent to solving three sets of simultaneous equations

$$\left.\begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 2 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\right\} \equiv \left\{\begin{array}{l} \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 2 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{21} \\ b_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 2 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} b_{12} \\ b_{22} \\ b_{32} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 & 4 \\ 3 & 1 & 2 \\ 2 & 4 & 5 \end{bmatrix} \begin{bmatrix} b_{13} \\ b_{23} \\ b_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{array}\right.$$

If we choose the elimination technique, we can solve the three sets simultaneously since the operations depend only on the coefficients. Writing the constants in a rectangular array:

$$\begin{bmatrix} 2 & 3 & 4 & 1 & 0 & 0 \\ 3 & 1 & 2 & 0 & 1 & 0 \\ 2 & 4 & 5 & 0 & 0 & 1 \end{bmatrix}$$

Dividing the first equation in each set by the leading coefficient (2) we get

$$\begin{bmatrix} 1 & 3/2 & 2 & 1/2 & 0 & 0 \\ 3 & 1 & 2 & 0 & 1 & 0 \\ 2 & 4 & 5 & 0 & 0 & 1 \end{bmatrix}$$

Copying the first equations: then subtracting 3 times the first from the second: and then 2 times the first from the third.

$$\begin{bmatrix} 1 & 3/2 & 2 & 1/2 & 0 & 0 \\ 0 & -7/2 & -4 & -3/2 & 1 & 0 \\ 0 & 1 & 1 & -1 & 0 & 1 \end{bmatrix}$$

Interchange 2nd and 3rd equations in each set.

$$\begin{bmatrix} 1 & 3/2 & 2 & 1/2 & 0 & 0 \\ 0 & 1 & 1 & -1 & 0 & 1 \\ 0 & -7/2 & -4 & -3/2 & 1 & 0 \end{bmatrix}$$

Subtract 3/2 of 2nd from 1st
Subtract -7/2 of 2nd from 3rd

$$\begin{bmatrix} 1 & 0 & 1/2 & 2 & 0 & -3/2 \\ 0 & 1 & 1 & -1 & 0 & 1 \\ 0 & 0 & -1/2 & -5 & 1 & 7/2 \end{bmatrix}$$

What have we accomplished thus far? Let's interpret that portion which represents the first set of equations.

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}b_{11}\\b_{21}\\b_{31}\end{bmatrix}=\begin{bmatrix}-3\\-11\\10\end{bmatrix}$$

By looking at the second and third sets of equations we can see that

$$B=\begin{bmatrix}-3&1&2\\-11&2&8\\10&-2&-7\end{bmatrix}\qquad\text{This Means}\qquad\begin{bmatrix}2&3&4\\3&1&2\\2&4&5\end{bmatrix}\begin{bmatrix}-3&1&2\\-11&2&8\\10&-2&-7\end{bmatrix}=\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}$$

How did we do it? We wrote the given matrix and appended the identity on the right. We transformed the given matrix into the identity using only the following operations.

1. Multiplication of a row by a constant.
2. Interchange of two rows.
3. Addition of a multiple of a row to a different row.

By carrying out these same operations on the identity, it was transformed into the identity.

While no special order is necessary, only these operations are valid.

ILL-Conditioned Sets of Simultaneous Linear Algebraic Equations

Suppose we are presented with the system:

$$\begin{aligned}L_1 &= 10x_1+7x_2+8x_3+7x_4 = 32\\L_2 &= 7x_1+5x_2+6x_3+5x_4 = 23\\L_3 &= 8x_1+6x_2+10x_3+9x_4 = 33\\L_4 &= 7x_1+5x_2+9x_3+10x_4 = 31\end{aligned}$$

Suppose further that we have by some means arrived at an approximation to the solution: $x_1 = 9.2$, $x_2 = -12.6$, $x_3 = 4.5$, $x_4 = -1.1$

How good is it? Is it acceptable? We can substitute these values into the left members above and see if the equations are satisfied. This gives:

$$\begin{aligned}L_1 &= 32.1\\L_2 &= 22.9 \qquad \text{worst relative error} \quad \frac{1}{230} \approx .43\%\\L_3 &= 33.1\\L_4 &= 30.9\end{aligned}$$

This looks like it might be acceptable. The worst error is less than 1/2%. But is it? The true solution is: $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$.

The worst error is 1360%. That is not acceptable. What constitutes an acceptable criterion? It depends upon why the problem is being solved but some possibilities are:

(a) Make the residuals small. (the first test applied)
(b) Make the solution nearly exact. (the second test applied)
(c) Determine numbers such that only a small change in the coefficients is necessary to make the solution exact.

Unfortunately b) is usually the test which must be satisfied.

A FAMILY OF TEST MATRICES

A. C. R. Newbery
University of Alberta, Calgary, Alberta, Canada

A family of test matrices with the following properties is here described: (a) An explicit inverse is given, (b) The characteristic polynomial is easily obtained, (c) A large measure of control over the eigenvalues is possible, (d) In special cases the eigenvalues and eigenvectors can be given explicitly, and the P-condition number can be arbitrarily assigned.

Consider a matrix of the form $Q = \begin{bmatrix} S & R \\ C & D \end{bmatrix}$, where S is a scalar, R is a row-matrix $\{r_2, r_3, \dots, r_n\}$, C is a column-matrix $\{c_2, c_3, \dots, c_n\}^T$ and D is a diagonal matrix with elements $d_2, d_3, \dots, d_n$. By use of the bordering method [1] the inverse is found to be $Q^{-1} = \begin{bmatrix} S' & R' \\ C' & M' \end{bmatrix}$, where each submatrix of Q^{-1} has the same form as the corresponding submatrix of Q, except that M' is generally not diagonal. Letting the subscripts of R', C', M' run from 2 to n, we find that

$$S' = 1/[S - \sum_2^n r_i c_i/d_i], \quad c_i' = -S' c_i/d_i, \quad r_i' = -S' r_i/d_i,$$

$$M_{ij}' = [\delta_{ij} - c_i r_j']/d_i,$$

where δ_{ij} is the Kronecker delta. The inversion can be performed in $2(n-1)(n+2)+1$ long operations; it might be possible to improve this figure with some ingenuity.

<u>The eigenvalue problem</u>. Let λ be an eigenvalue of Q, and let $\bar{x} = \{1, x_2, \dots, x_n\}^T$ be the associated eigenvector. This leads to the following set of n equations:

$$S + \sum_2^n r_i x_i = \lambda, \quad c_i + d_i x_i = \lambda x_i \quad \text{for } i \geq 2.$$

On eliminating the x_i we obtain

(1) $$S + \sum_2 r_i c_i/(\lambda - d_i) - \lambda = 0.$$

If we write $\Pi(\lambda) = \prod_2^n (\lambda - d_i)$, $\Pi_i(\lambda) = \Pi(\lambda)/(\lambda - d_i)$, then on clearing the fractions in (1) we obtain

(2) $$(\lambda - S)\Pi(\lambda) - \sum_2^n r_i c_i \Pi_i(\lambda) = 0.$$

This is the characteristic equation. The following statements can be made concerning the eigenvalues:

(A) If all $r_i c_i > 0$ and all d_i are distinct, then all the eigenvalues are real and are separated by the d_i.

(B) If all d_i are equal to d, then there are n-2 eigenvalues equal to d; the remaining two are zeros of the quadratic function $\lambda^2 - (S+d)\lambda + Sd - \sum_2^n r_i c_i$. These zeros are real if, and only if, $(S-d)^2 + 4\sum r_i c_i \geq 0$.

(C) If all d_i are equal to d, then the eigenvectors associated with the multiple eigenvalue d have zero as their first component, and they are orthogonal to the vector $\{0, r_2, \dots, r_n\}$. Eigenvectors corresponding to the other two eigenvalues are $\{\lambda_p - d, c_2, \dots, c_n\}$, where λ_p is a zero of the quadratic given in (B).

Proof of (A). Let $H(\lambda)$ denote the left side of (1), and let $\{d_i'\}$ denote a reordering of the $\{d_i\}$ so that $d_i' < d_{i+1}'$. We note that $H(\lambda)$ is continuous in any interval which does not enclose any of the d_i', and that for sufficiently small ε $H(d_i' + \varepsilon) > 0$ and $H(d_{i+1}' - \varepsilon) < 0$. Hence there is a zero of $H(\lambda)$ between each consecutive pair of the $\{d_i'\}$; moreover since $H(-\infty) > 0$ and $H(\infty) < 0$, there are two more real zeros of $H(\lambda)$ outside the interval (d_2', d_n').

Proof of (B). If all the d_i are equal to d, then $\Pi(\lambda) = (\lambda-d)^{n-1}$ and $\Pi_i(\lambda) = (\lambda-d)^{n-2}$. The characteristic equation (2) then reduces to $(\lambda-d)^{n-2}[(\lambda-S)(\lambda-d) - \sum_2^n r_i c_i] = 0$. The discriminant of the quadratic factor is $(S-d)^2 + 4\sum r_i c_i$. Statement (C) may be directly verified.

The P-condition number, i.e. the largest absolute ratio to two eigenvalues [2], can most conveniently be assigned by letting $d_i = d$; then, using statement (B), we can choose S and $\sum r_i c_i$ in such a way as to assign any desired zeros to the quadratic; hence any desired maximum ratio of eigenvalue magnitudes may be procured.

Remarks. If the inverse matrices are included along with the original family, then we have freedom within the family to specify sparse, non-sparse, symmetric, non-symmetric, well- or ill-conditioned matrices; furthermore we can require that the eigenvalues shall be all real or mixed real and complex. This should provide sufficient versatility for most test purposes.

REFERENCES

1. FADEEVA, V. N., Computational Methods of Linear Algebra. Dover New York, 1959.

2. MARCUS, M., Basic Theorems of Matrix Theory, N.B.S. Appl. Math. Ser. 57, 1960.

THE IBM 1620 AS ANALYTICAL AND PRE-COMPOSITIONAL AID IN 12-TONE MUSIC

By Albert Tepper, Associate Professor of Music

Hofstra University
Computer Center

1620 Users Group No. 1320

* * * * * * * *

Time required for presentation: 20 minutes

Special projection equipment required: Tape recorder, overhead projector

* * * * * * * *

In the early 1920's the Austrian musician Arnold Schoenberg evolved a compositional technique, projected from an analysis of his own works, which he called A METHOD OF COMPOSING WITH THE TWELVE TONES RELATED ONLY TO EACH OTHER. In this method all twelve pitch classes in the chromatic scale are arranged in some specific order called a "tone row", "row" or "series". The total number of rows possible, by the way, is 12!; i.e., 479,001,600.

The row is a constant group of relationships for a particular composition and all aspects of pitch organization derive from it: two or more sequential pitches played successively create melody; two or more sequential pitches played simultaneously create harmony.

When read forward (from left to right), the row is said to be in its "original" or "prime" form. Read backwards the row is in its "retrograde" form--a mirror image. Another mirror image--the inversion--is created by altering the direction of each successively adjacent pair of tones. For example: if pitch #2 lies three semi-tones above pitch #1, its inversion will lie three semi-tones below; if pitch #3 lies one semi-tone below pitch #2, its inversion will lie one semi-tone above. A backward reading of the inversion gives us the retrograde inversion. These four forms of the same row--prime, retrograde, inversion and retrograde inversion--constitute the "basic set". Each basic set is capable of being moved--that is, transposed--from its own pitch level to every other pitch level of the chromatic scale, twelve pitch levels in all. Thus forty-eight specific pitch orderings are spanned by one row. The composer chooses from among the forty-eight in any order he sees fit.

Milton Babbitt of Princeton University has shown that the four original forms of the row plus all transpositions have the properties of a set and may be stated as a matrix. Rather than label the first pitch in the prime form #1, the second pitch #2, and so on, let us designate each pitch by its semi-tonal distance from pitch #1, which is set at zero. Since our interest is in pitch classes rather than specific pitches (all C's have the same value, all F sharps have the same value, all B flats have the same value, etc.), let us also arbitrarily assume C to be pitch #1 with the value of zero. In our present equally tempered tuning system, C sharp and D flat are the same pitch and lie one semi-tone above C, thereby having the value of 1. D has the value of 2, D sharp and E flat the value of 3, E the value of 4, etc. B, one semi-tone below C, is also eleven semi-tones above, and therefore has the value of 11.

As the distance between any two pitches can never be less than zero, and as any positive value greater than 11 duplicates a pitch class in the zero to 11 range, it follows that we are limited to whole numbers with base 12.

A 12 X 12 pitch matrix can thus be stated in purely arithmetical terms. Each rank is both prime and retrograde forms at one of the twelve pitch levels, each column is both inversion and retrograde inversion forms at one of the twelve pitch levels. Since the matrix shows all row possibilities in compact form and at a glance, its value to both the 12-tone composer and musical analyst is obvious. But while the 144 cells of the matrix may be filled in by "hand", the job is a tedious one, taking 20 minutes, with 132 initial possibilities for error. It seemed worthwhile, therefore, to program the computer to produce a matrix from a given row.

Mr. Lowry L. McKee, Assistant Director of the Hofstra University Computer Center, guided, instructed, helped and sustained me while I struggled with the problem. Certain refinements were added as we went along, as follows:

1. Since each pitch class already had an assigned value, it was no longer necessary to start each row with zero. The twelve cards of the data deck, each with a number from zero to 11, may be placed in the Read Hopper in any order.

2. The matrix is stated in three forms: as a set of numbers; as a set of pitch classes designated by the letters of the musical alphabet from A to G, in their natural and sharp variants (the plus sign substitutes for the sharp); and as a set of pitch classes from A to G in their natural and flat variants (the minus sign substitutes for the flat).

3. An interesting phenomenon, which can be useful to the 12-tone composer, is the ability of certain row forms to combine and create permutations. George Rochberg of the University of Pennsylvania has devised an arithmetical test for this. An odd number from 1 to 11 is added as a constant to each of the first six values in the prime form which begins with zero. The same constant is then added to every pair of the same six values. If the sum in each instance is neither 12 nor 24, the first six values (that is, the first half) of the prime form and the first half of the inversion will combine and create a 12-tone permutation at the semi-tone distance of the constant. The second halves of both prime and inversion will also combine in the same manner. The result is of course transposable to all pitch levels. The program devised by Mr. McKee and myself performs this operation and supplies the result in those instances where the result is positive.

Delivered to the 1620 Users Group Meeting, 1964, in Chicago.

Logic Theorem Detection Program

The program is designed to take conventionally written (i.e., not in a bracket-free notation) well formed formulae (WFF) of the propositional calculus as input, test them for theoremhood, and state the result as output. There are several subsidiary output results possible. Input and output are via the typewriter. The program is written in SPS for the IBM 1620 computer. It uses about 5500 cores and consists in about 400 commands plus storage.

Conditions of use: The original WFF may contain only three primitive variables (P, Q, & R) and four operators (those for conjunction, disjunction, implication & negation). The WFF may be up to 49 symbols in length and may contain up to seven pairs of brackets. Thus the WFF which may be processed conform to the requirements of a fully developed propositional calculus and there is, in any case, no theoretical difficulty in extending the range of WFF which may be processed. The computer does not take all the conventional logical symbols and the following symbolisation has therefore been used: disjunction '+', conjunction '.', implication '/', negation '-'.

The rules for WFF are formulated in different ways for the propositional calculus. The following formulation is used here: Any primitive variable is well formed. If anything which is a WFF is designated by X, Y, etc., then: -X is well formed; (X.Y) is well formed; (X + Y) is well formed; (X/Y) is well formed. The brackets round the whole of a WFF to be tested for theoremhood need not be included, e.g., P/(P + Q) may be tested for theoremhood as it stands. It should be noticed that (P.Q.R), (P + Q + R), etc. would not be well formed in this formulation but would have to be written (P.(Q.R)), ((P + Q) + R), etc. These rules are entirely typical for a propositional calculus.

The program operates as follows (see attached sheet for sample): It announces itself and invites typein of a WFF. It then types the result. The following subsidiary results may be obtained on the Consul switches: (1) By well known theorems of the propositional calculus, any number of negatives greater than one ('stacked negatives') before a WFF may be reduced to one or none. On switch 1, the original WFF is typed without stacked negatives. (2) The method of processing employed is to break the original WFF down into a two variable form, the computer supplying new variables where required. These new variables themselves stand for WFF which are broken down in the same way. The effect is to produce a two variable list in which no brackets are required (they are not required for the same reason that brackets are not needed round the whole of a WFF to be tested); the first item in the list is the original WFF in two variable form and the subsequent items define the new variable or variables introduced by the computer. This process continues until all the introduced variables are defined. On switch 2, the list is typed, with each introduced variable explicitly defined. (3) The method employed in the logic section of the program is to build a truth-table, with a set of values for each variable, primitive or defined. On switch 3 this table, or a desired portion of it, can be obtained. The operator types in a variable and the computer gives the associated set of values; by typing in all the variables used or introduced by the computer, plus 'F' for the original WFF, a complete table is obtained. The table is biniary, containing either 01 (true) or 00 (false) in each of its eight columns; if the WFF being tested is a theorem, the table will contain eight entries of 01 for the original WFF. Typing in 'N' returns the computer to the main program.

Jon Wheatley, Philosophy,
Queen's University at Kingston,
Ontario, Canada.

LOGIC THEOREM DETECTION PROGRAM - WHEATLEY, PHILOSOPHY, QUEENS, NOV 1963

```
TYPE IN WFF
(P/-Q)/(Q/-P)
THEOREM


TYPE IN WFF
((P.-P).--Q)/((((Q+-P)+(R.-Q))+---P)
TEST ON C1 - WFF WITHOUT STACKED NEGATIVES
((P.-P).Q)/(((Q+-P)+(R.-Q))+-P)
TEST ON C2 - WFF AS TWO VARIABLE LIST
WFF S/U
 S= T.Q
 T= P.-P
 U= V+-P
 V= W+X
 W= Q+-P
 X= R.-Q
THEOREM
TEST ON C3 - TRUTH TABLE ON DEMAND
PRS0101010100000000
QRS0101000001010000
RRS0100010001000100
SRS0̅00̅00̅00̅00̅00̅00̅00̅0
TRS0̅00̅00̅00̅00̅00̅00̅00̅0
URS0̅10̅10̅10̅00̅10̅10̅10̅1
VRS0̅10̅10̅10̅00̅10̅10̅10̅1
WRS0̅10̅10̅00̅00̅10̅10̅10̅1
XRS0̅00̅00̅10̅00̅00̅00̅10̅0
FRS0̅10̅10̅10̅10̅10̅10̅10̅1
NRS


TYPE IN WFF
(-Q+-(R+(-Q+P)))/--(-R.(Q.P))
TEST ON C1 - WFF WITHOUT STACKED NEGATIVES
(-Q+-(R+(-Q+P)))/(-R.(Q.P))
TEST ON C2 - WFF AS TWO VARIABLE LIST
WFF S/V
 S= -Q+-T
 T= R+U
 U= -Q+P
 V= -R.W
 W= Q.P
NO THEOREM
```

An Additive Pseudo-random Number Generator

H.T. Wheeler*, J.K. Lewis, E.A. Cherniak

Department of Chemistry

Carleton University

Ottawa, Canada

INTRODUCTION - This generator was developed for use in a machine language programme (1) requiring random digits and short random fields. The method of generation exploits the variable field length feature of the 1620 by adding fields of several hundred digits in length. A small table of random digits is generated and stored in memory. From this table are selected random digits and/or fields as required. When any table is used up a new table is generated using the old table as input data.

The original programme has been modified and rewritten in SPS for use as a Fortran subroutine. Since the method of generation involves addition only, this generator is faster than the usual multiplicative generator.

The tests for randomness which have been performed on the output of the generator have given quite satisfactory results.

METHOD - The generator requires an initial random number of 501 digits. This initial number may be conveniently obtained from a table of random numbers such as the Rand Corporation, "1,000,000 Random Digits with 100,000 Normal Deviates".

The initial number, which will be denoted N_1, is divided into two component fields, A_1 and B_1, of 311 and 190 digits respectively. Thus:

$$N_1 = (A_1)\ (10^{190}) + B_1$$

A second number, C_1, is formed by reversing the order of the two component fields. Thus:

*Speaker (to whom enquiries concerning this program should be sent)

$$C_1 = (B_1)\ (10^{311}) + A_1$$

The second random number, N_2, is formed by adding C_1 to N_1 and discarding the high order carry, if any:

$$N_2 \equiv (C_1 + N_1)\ (\text{modulo } 10^{501})$$

The number N_i is generated from N_{i-1} by the same procedure.

The choice of the values 501, 311, and 190 for the field lengths was largely arbitrary although it was intuitively felt that better performance would be obtained if each of the values had few prime factors and no common factors existed among the three values. 311 is a prime number; 501 and 190 factor into (167) (3) and (19) (5) (2) respectively.

RATE OF GENERATION - The 501 digit number is generated in 61.5 milliseconds (for a Model 1). The average time taken to obtain an 8 digit field, normalize, and store in FAC, when the generator is used as a Fortran floating point subroutine, is about 4.5 milliseconds. The time taken by the Fortran 2 variable precision subroutine is roughly given by (3.5 + f/8) milliseconds, where f is the mantissa length.

TESTS FOR RANDOMNESS - The major tests which have been performed on the generator were for the freouency distributions of single digits, of ordered pairs of digits, and of runs of repeated like digits. These tests were performed on the 501 digit numbers without any division into smaller fields. Some of the test results are shown in the following tables.

In Table 1 the results of a digit frequency test on one million digits are shown. Except for the somewhat large chi-square values for blocks 9 and 10 the results are very satisfactory.

Table 2 shows a typical result of the ordered pair analysis. The expected value for each entry in the matrix is 501. The frequency of the ordered pair xy is the yth term in the xth row.

In Table 3 the repeated like digit analysis results are shown.

The results of these tests are sufficiently good to indicate a usable degree of randomness. If a greater degree of randomness is required the output of two separate generators could be added to produce an improved 501 digit table.

We are grateful to the National Research Council of Canada for the financial assistance which made this work possible and the Computer Policy Committee of Carleton University for granting us the necessary machine time.

1. Lewis, Wheeler, Cherniak - A Model Diffusion-reaction programme for the 1620 - 1620 Users Group Joint Meeting (Canadian and Mid-Western Regions) Chicago, February, 1964.

TABLE I

DIGIT FREQUENCY TEST

ONE MILLION DIGITS

Block No.	0	1	2	3	4	5	6	7	8	9	CHI^2
1	4956	5014	5025	5077	4960	5050	4909	5009	4996	5004	4.24
2	5038	4927	5054	4934	4943	5056	5008	4965	4967	5108	6.85
3	5039	4954	4994	4978	5006	5061	4923	4926	5035	5084	5.52
4	4979	5002	5073	5041	5056	4947	4980	4967	5027	4928	4.16
5	5005	4907	4906	5117	5025	5094	4909	5079	4976	4982	11.21
6	4973	5108	4983	4996	4914	5010	5079	4877	5079	4981	9.63
7	4938	5028	4995	4902	5096	5063	5087	4999	4974	4918	8.48
8	5011	5047	4938	5064	5006	4871	5036	4957	5064	5006	6.85
9	4995	4806	4966	5040	5011	4884	4960	5026	5163	5149	20.78
10	5125	5063	5000	5062	4832	5117	5060	4924	4992	4825	21.08
11	4974	5032	4908	5062	5021	5019	4961	4908	5109	5006	7.34
12	4964	4934	5111	4930	5096	5018	4862	5004	5121	4960	13.54
13	4965	5004	5027	5021	5076	5044	5021	4995	4931	4916	4.48
14	5056	5077	4898	5081	5053	4972	4897	5000	4856	5110	14.61
15	4960	5094	5037	4876	4990	4995	5108	4963	5006	4970	8.24
16	4981	4930	5046	4982	5077	4970	5035	5083	4970	4926	5.80
17	4977	5071	5096	4955	4982	4863	4972	4962	5077	5045	9.22
18	5116	4996	4990	5009	4981	4972	4848	4987	4943	5157	13.19
19	5008	4880	5042	4943	5008	5093	5024	4968	5014	5020	6.08
20	4015	4998	4989	4950	5106	4968	5012	4963	4970	5029	3.67
	100075		100078		100239		99691		100270		
Total		99872		100020		100067		99562		100125	

CHI^2 (totals) = 4.660 Probability (> CHI^2) = 0.86

Of the 200 frequencies 64 deviate by more than sigma (=67.08)

(expected number = 63.4) and 9 frequencies deviate by more than two sigma
(expected number = 8.2).

Tests on column totals	CHI^2	Probability
Odd versus even digits	0.498	0.49
Within odd digits	2.372	0.68
Within even digits	2.287	0.69

TABLE II

ORDERED PAIR ANALYSIS ON 100 NUMBERS OF 501 DIGITS.

	0	1	2	3	4	5	6	7	8	9	CHI^2
0	472	504	544	477	509	490	496	496	493	454	11.54
1	519	494	511	516	487	527	503	516	503	506	3.64
2	502	490	497	514	474	502	530	505	519	511	4.62
3	514	520	499	494	451	517	479	505	507	518	8.31
4	479	480	489	493	472	496	508	456	462	530	12.84
5	511	502	520	529	482	492	505	514	472	475	6.76
6	495	542	498	476	521	495	517	530	527	513	9.38
7	468	507	507	506	489	513	486	483	498	509	4.18
8	476	562	492	496	508	469	544	475	479	487	17.42
9	500	483	493	509	469	498	540	479	530	499	8.65
CHI^2	7.32	13.46	5.31	5.20	12.59	4.73	10.51	9.58	9.58	9.07	

Probability ($CHI^2 > 16.92$) = 0.05

TABLE III

REPEATED LIKE DIGIT

RUN ANALYSIS RESULTS - TESTS ON 50,100 DIGITS

LENGTH OF RUN

TEST NO.	1	2	3	4	5	6
1	40723	3987	411	40	2	0
2	40592	4079	404	49	1	0
3	40639	4036	406	34	7	0
4	40488	4122	391	45	3	0
5	40570	4068	397	42	7	0
AVERAGE	40590	4058	402	42	4.0	0.0
EXPECTED	40581	4058	406	41	4.1	0.4

A MODEL DIFFUSION-REACTION PROGRAMME FOR THE 1620.

J.K. Lewis, H.T. Wheeler, E.A. Cherniak*
Department of Chemistry
Carleton University
Ottawa, Canada

Diffusion, particularly neutron-diffusion, has been studied by Monte Carlo on digital computers since the introduction of the ENIAC. However, as far as we know, no extensive applications of this method, to diffusion-reaction problems of a chemical nature, have yet been given. This paper describes one approach to such a problem.

In the radiolysis of liquid benzene, it is well known that a high energy particle leaves behind it a cylindrical track of excited molecules and ions. The track is of varying density, depending on the energy of the incident particle. It was believed that certain phenomena, notably the variation of hydrogen yields with stopping power, could be explained, in a way, suggested by Burns(1), by assuming that these activated molecular entities were of one form, B*, which could react in the following way:

$$B \rightsquigarrow B^*$$

$$B^* + B^* \xrightarrow{BR} H_2 + P(?)$$

$$B^* + B \xrightarrow{UR} 2B$$

The main feature then to be explained was the variation of H_2 yields with stopping power. This is shown in figure 1 where the experimental average values of the H_2 yields, compiled by Burns and Barker(2), from the results of the investigations of a number of workers, are given as a logarithmic function of the Ganguly and Magee(3) average stopping power Z.

* Speaker (to whom enquiries concerning this programme should be sent.)

In the programme which we eventually used, a particle was represented by its coordinates, each coordinate having four digits and the total comprising a field of twelve digits with a single flag on the high-order digit. Reacted particles were denoted by a flag on the low-order digit. The quenching reaction (UR) was treated as an unimolecular process and for the bimolecular process (BR) the coordinates of each particle had to be compared.

The "tracks" (see flow chart 1) were set up in the most elementary fashion. A few cards were read in, each with the coordinates of a particle and the number of particles with those coordinates to be set. When all the cards were read, the track could be stored elsewhere in memory. The numbers controlling the rates of reaction and the width of the initial distribution were then read in.

The initial particle distribution was then generated by allowing the particles to be moved the specified number of times without reacting. The distribution generated in this manner by, say, six pre-reaction moves is quite close to Gaussian.

The particles were then moved and reacted, alternately, (see flow chart 2) until all the particles were gone, after which the numbers of particles reacted was typed and the programme repeated.

Each particle was moved by adding to it a 10 digit field with a random digit in the first, fifth and the ninth positions from the right. Thus the centre of gravity of the particles "drifted" steadily through model space. It was found that random digits with a rectangular distribution, i.e. equal probabilities for all the digits, caused the particles to diffuse off at an inconveniently high rate. We were thus faced with the problem of generating large numbers of digits rapidly with a skewed distribution. The obvious method, playing a game of chance, proved cumbersome. However, a simple modification of the random number generator, described by Wheeler(4), proved successful.

To produce a distribution of digits the programme replaced the add tables with special add tables having digits in the desired frequency. Thus these digits could be produced in an average time $< 250\mu$sec./digit. Furthermore, the frequencies of the digits could be controlled to 1 part in 100, although as our particles had a moving centre of gravity, the frequency distribution had to be symmetrical about the mean move length, restricting our control to 1 part in 50. This was far more than sufficient.

The "react" routine (flow chart 2) contained a scanner which scanned the appropriate section of memory until it found either a record mark, indicating the end of the particles, or an unreacted particle. Then, depending upon the state of a simple flip-flop operating on bd's and tdm's, the particle would be "reacted" first unimolecularly....and then bimolecularly, or vice versa. This altergration on unimolecular and bimolecular from one particle to the next was found necessary at high concentrations of particles.

To ascertain whether a particle was to have reacted unimolecularly or not, a four digit field was taken from the random number generator and compared with a control number. If the random number chosen was less than the control number, the particle was assumed to have reacted, a flag was placed on its units position, a counter was incremented, and control returned to the scanner to find another particle. The four digit length was actually found to be necessary to give adequate control over the unimolecular rate. If the random number was greater than the control number, the program would proceed to the next particle or would consider the particle for bimolecular reaction, depending on the status of the flip-flop.

For the bimolecular reaction, the particles above the particle under consideration were examined for whether or not they were reacted. When an unreacted one was found, its coordinates were compared, by a single c instruction, with the particle under consideration. Analysis showed that it would probably be faster to compare all twelve digits of each sets of coordinates at once than divide them into 3, of which only the first four would be compared for two non-coincident particles. When two particles were found to coincide, the usual chance game was played to aetermine if reaction had taken place. Unlike the unimolecular reaction, however, only a three digit control number was used and two digits would have sufficed.

The programme described above, which was written in about 1000 machine language commands on a 40K 1620 with automatic divide, will "react ~ 150 particles to completion in 5-15 minutes, depending on conditions, but typically 7 minutes, with a standard deviation of ~15% in the results. A comparison of our results with the experimental data is shown in figure 1. By varying the control numbers it was found that the experimental data could be fitted quite closely and that the values this fitting procedure gave for the rate constants were in fact plausible.

We are now engaged in refining our interpretation to get improved values for rate constants. Actually, this problem of interpretation is the main disadvantage of our approach vis à vis a numerical integration method. Our programme is probably as fast or faster than a numerical programme for the same mechanism, and the ~15% deviation is not excessive for experimental values accurate to better than 10% are rare in this field. However, the process of getting from our model to the physical situation is rather involved. First we work out a one dimensional distribution of particles which have moved several times with the move distribution we use. We then fit a Gaussian curve to this and from this obtain a model diffusion coefficient. A good value for the real self-diffusion coefficient is available, so this gives us a value for "model length)^2/ model time" in real units. If a good model length is found, then the model time drops out. From the diffusion coefficient and the control numbers the two rate constants are readily obtained. However,

the "model length" poses the problem. After some thought we decided it should be twice the "collision diameter" of the molecule, but, in liquids this is a rather ill-defined quantity, and reasonable values based on various definitions tend to differ somewhat.

The main virtue of our program was its flexibility. It can handle regions of intermediate stopping power, where the track resembles billiard balls strung together on a cord, which are particularly difficult for standard numerical integration procedures. It can also handle varying amounts of particles corresponding to different input power, and with modifications could treat other problems, such as effects of small amounts of reactive solutes. These last two features were never used, partly because of a paucity of experimental data, because the effects are notable chiefly through their absence, but principally because crude hand calculations upon the "plausible" rate constant values were sufficient to show that these effects should in fact be small.

We are grateful to the National Research Council of Canada for the financial assistance which made this work possible and the Computer Policy Committee of Carleton University for granting us the necessary machine time.

1. Burns, Trans. Faraday Soc. 59 101 (1963).
2. Burns and Barker, United Kingdom A.E.R.E. Report 4240 (1963).
3. Ganguly and Magee, J. Chem. Phys. 25 129 (1956).
4. Wheeler, Lewis, Cherniak - A new random number generator -
1620 Users Group Joint Meeting
(Canadian and Mid-Western Regions)
Chicago, February, 1964.

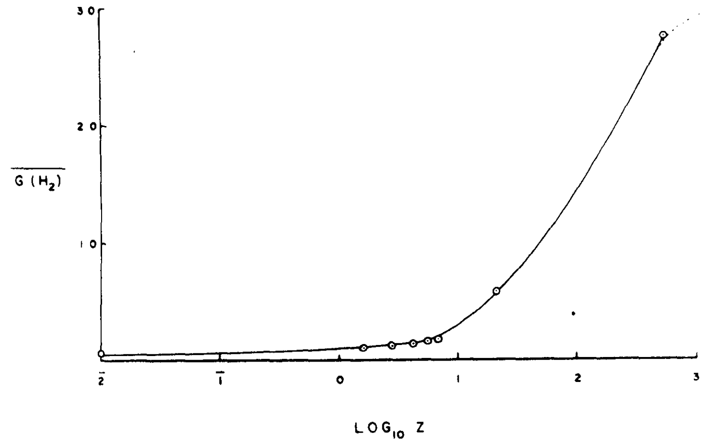
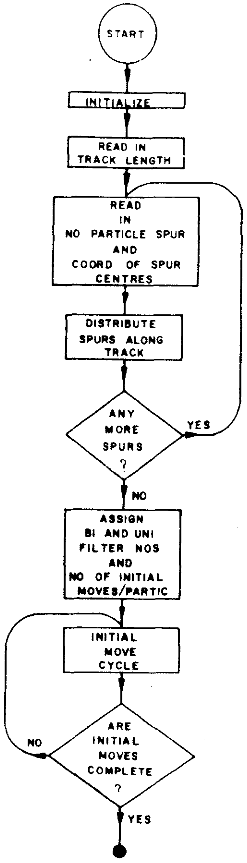
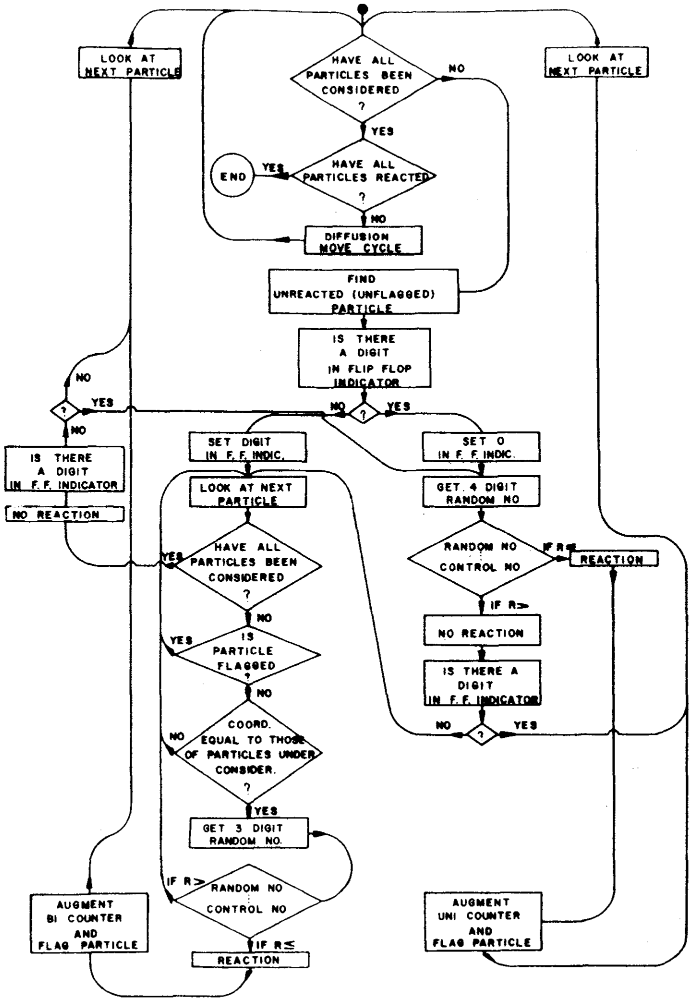


FIG 1. Variation of experimental $G(H_2)$ (⊙) with average L.E.T. in the radiolysis of liquid C_6H_6, ——— obtained by diffusion reaction programme, ------ predicted by diffusion reaction programme.


START
INITIALIZE
READ IN TRACK LENGTH
READ IN NO PARTICLE SPUR AND COORD OF SPUR CENTRES
DISTRIBUTE SPURS ALONG TRACK
ANY MORE SPURS ?
YES
NO
ASSIGN BI AND UNI FILTER NOS AND NO OF INITIAL MOVES/PARTIC
INITIAL MOVE CYCLE
NO
ARE INITIAL MOVES COMPLETE ?
YES

FLOW CHART I: Diffusion-reaction programme: track and initial particle distribution generator.


LOOK AT NEXT PARTICLE
HAVE ALL PARTICLES BEEN CONSIDERED ?
NO
LOOK AT NEXT PARTICLE
YES
END
YES
HAVE ALL PARTICLES REACTED ?
NO
DIFFUSION MOVE CYCLE
FIND UNREACTED (UNFLAGGED) PARTICLE
IS THERE A DIGIT IN FLIP FLOP INDICATOR
NO
?
YES
NO
YES
?
NO
SET DIGIT IN F. F. INDIC.
SET 0 IN F. F. INDIC.
IS THERE A DIGIT IN F. F. INDICATOR
NO REACTION
LOOK AT NEXT PARTICLE
GET 4 DIGIT RANDOM NO
YES
HAVE ALL PARTICLES BEEN CONSIDERED ?
RANDOM NO : CONTROL NO
IF R≤
REACTION
NO
IF R>
YES
IS PARTICLE FLAGGED ?
NO REACTION
NO
IS THERE A DIGIT IN F. F. INDICATOR
NO
COORD. EQUAL TO THOSE OF PARTICLES UNDER CONSIDER. ?
NO
?
YES
YES
GET 3 DIGIT RANDOM NO.
IF R>
RANDOM NO : CONTROL NO
AUGMENT BI COUNTER AND FLAG PARTICLE
IF R≤
REACTION
AUGMENT UNI COUNTER AND FLAG PARTICLE

FLOW CHART 2 Diffusion-reaction routine

AUTOSPOTLESS NUMERICAL CONTROL WITH THE 1620

by

E. R. Austin

Engineering Computer Facility
Combustion Engineering, Inc.
Chattanooga, Tennessee

Presented February 21, 1964

AUTOSPOTLESS NUMERICAL CONTROL WITH THE 1620

A great deal of emphasis is being given to the 1620 and its role in the numerical control of machine tools. This is certainly as it should be. However, the resulting emphasis on the APT (Automatically Programmed Tools) language and the Autospot subset of this language is unjustified.

The use of Autospot assumes:

1. The only output desired is a paper tape and a listing of its contents.
2. Sufficient 1620 time to use a four deck processor (including post processor) for each workpiece.
3. The existence of a post precessing program.
4. The existence of trained "parts programmers".

Due to the specialized nature of our numerically controlled machines, operator's instructions must accompany the tape as it enters the shop. Furthermore, Industrial Engineering must prepare standard hours for entry on the shop routing or traveler. Thus, it has been decided that all these documents should be computer created. Autospot does not lend itself to this effort.

Over 175 workpieces are processed monthly on our Ingersoll header drill. This represents drilling and chamfering some 70,000 holes each month. For the Autospot processor, this represents over 70 hours of computer time. This would mean second shift operation for almost any 1620 facility.

The creation of a post processor involves understanding of Autospot, the tool to be controlled, the 1620 and the controlling mechanism itself. This, coupled with the fact that Autospot is really more comprehensive than is necessary for our applications, makes the creation of a fixed format input processor most advisable.

With such fixed format input programs, no "parts programmers" are required. Industrial Engineering members can readily interpret engineering drawings and compactly represent this information on input sheets.

The total time from the interpretation of the drawing to the creation of pertinent documents is greatly reduced by use of this concept. To illustrate, the Ingersoll header drill with its accompanying functions and required documents can be cited.

The N/C Ingersoll header drill is used for drilling and counterboring cylinders on the order of 60 feet long, 1 ft. outside diameter and two inch (2") thick walls. The holes in this cylinder normally align themselves into six (6) or less rows down the header. The hole spacings are highly irregular and are a function of the boiler system of which the header is a part. The work which preceeds the actual drilling of a header is best described by fig. 1. The three (3) documents entering the shop serve the following functions:

1. Routing slip
   a. Provides operational sequence
   b. Shows standard hours allocated for each operation
   c. The approximate date on which each operation should occur is also shown
2. Tape contains
   a. Positions for drilling
   b. Drilling feeds and speeds
   c. Counterboring feeds and speeds
   d. Spindle starts and stops
   e. Gear changes
3. Operators instructions
   a. Shows angularity of each row from a given point
   b. Tells the operator when to use what tools
   c. Provides settings for limit switches

Proper representation of input data permits the creation of all these documents. Autospot does not lend itself to such a representation. For example, Autospot must be told which tool to use. Our feeling is that the program should select the tool. The computer selection of a tool eliminates a great deal of human thought and potential error since tool choice is a function of material type, nipple o.d., nipple wall thickness, thickness of header, etc. This selection then fixes the counterboring diameter, counterboring depth, drilling feed and speed, counterboring feed and speed, gear range, etc.

The input sheet used by our program is shown in fig. 2. The manner in which the completion of this sheet fits into the overall picture of fig. 1 is shown by fig. 3.

Several points are worth noting about this system:

1. No parts programmers are required. Technicians complete the input sheets.
2. No post processor is required
3. Two SPS programs can create all the described documents.
4. A typical header can be processed through the 1620 in less than .1 hour to produce all documents.

t. Autospot would require no less than .4 hours to create tape information alone.

When the total systems approach is applied to N/C problems, the fixed format input exemplified by this system and other systems like the IBM 1401 Autoprops seems to have definite advantages over APT processors.

FIGURE #1

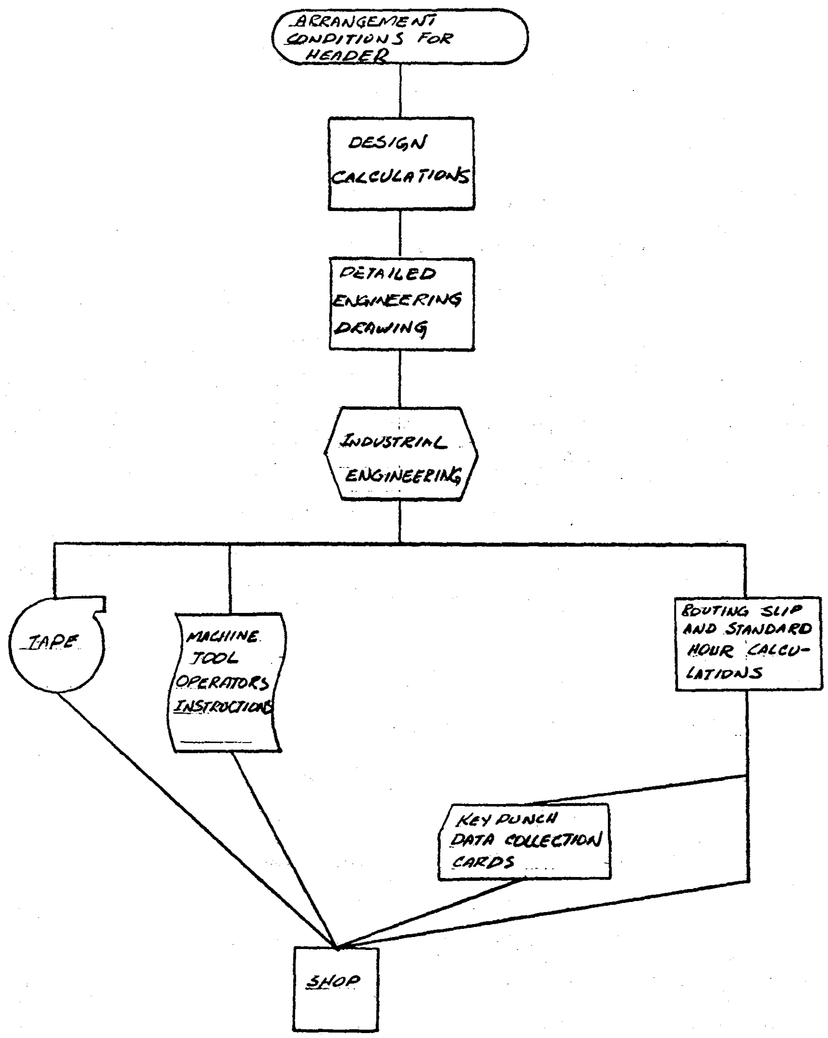

ARRANGEMENT CONDITIONS FOR HEADER
DESIGN CALCULATIONS
DETAILED ENGINEERING DRAWING
INDUSTRIAL ENGINEERING
TAPE
MACHINE TOOL OPERATORS INSTRUCTIONS
ROUTING SLIP AND STANDARD HOUR CALCU-LATIONS
KEY PUNCH DATA COLLECTION CARDS
SHOP

FIGURE #2

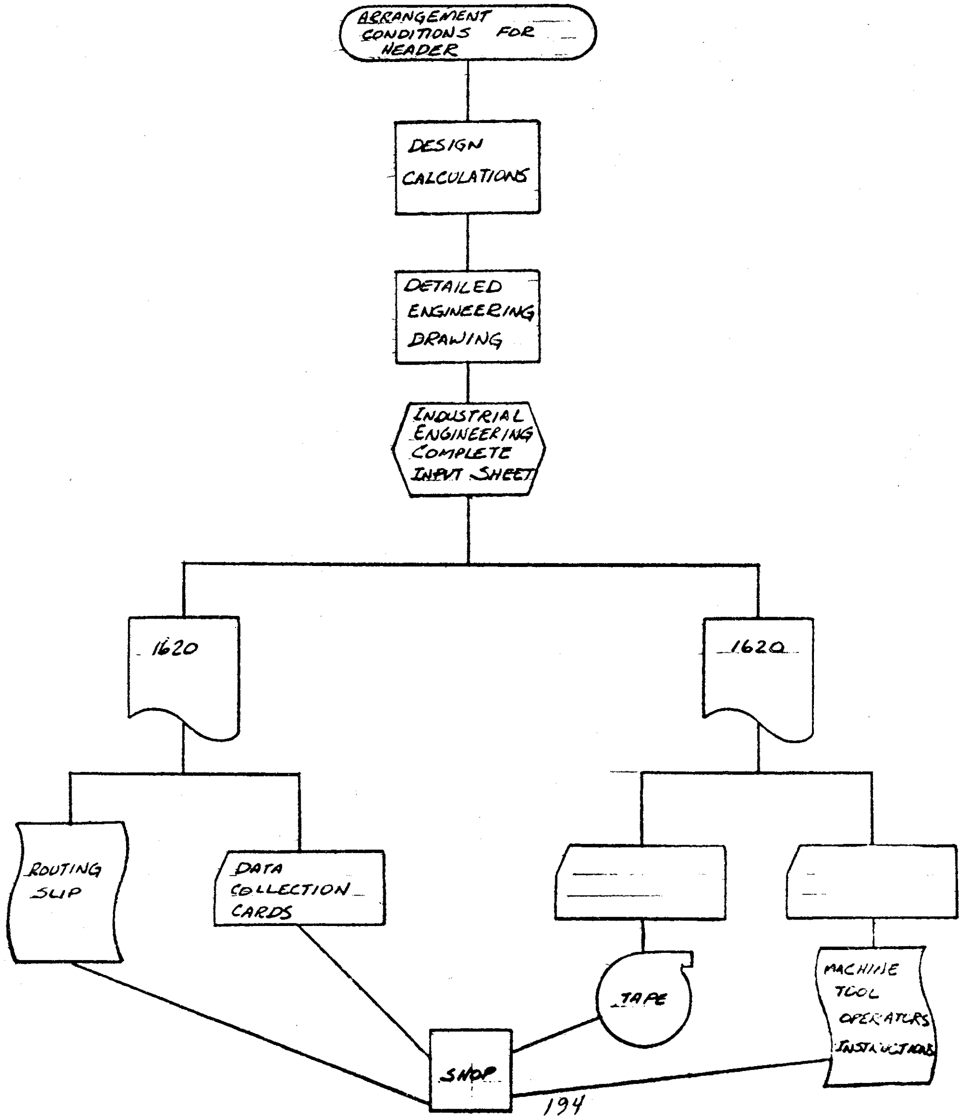
CUSTOMER S.O. NO. DRAWING NO. DATE

PART NAME NO. OF FINISHED PIECES PROGRAMMED BY

MATERIAL* MATERIAL SIZE & / " OD X & / "AW X '- & / " L

		NIPPLE	WALL	CHAM	INITIAL				FINAL	
ROW	ANGLE	OD	THK.	FER	X		SPACES	ΔX	X	

FIGURE #3


ARRANGEMENT CONDITIONS FOR HEADER
DESIGN CALCULATIONS
DETAILED ENGINEERING DRAWING
INDUSTRIAL ENGINEERING COMPLETE INPUT SHEET
1620
1620
ROUTING SLIP
DATA COLLECTION CARDS
TAPE
MACHINE TOOL OPERATORS INSTRUCTIONS
SHOP

IBM

Technical Publication

AUTOSPOT II PREPROCESSOR PROGRAM

By D.F. McManigal

PRG 26.0006

Poughkeepsie Manufacturing Engineering
Data Systems Division

195

ABSTRACT

The Autospot II Preprocessor is an IBM computer program used to help the parts programmer prepare correct input data for Autospot II.

January 2, 1964

1.0 INTRODUCTION

Autospot II (AUTOmatic System for POsitioning Tools, Model II) is a computer program for the IBM 1620 Data Processing System (see Figure 1). It was designed to aid the parts programmer in preparing instructions for numerically controlled point-to-point machine tools. Autospot permits the use of easily remembered codes, such as DRILL, TAP, and MILL, instead of the more complicated numeric codes used by numerical control machine tools. Autospot performs many computations for the parts programmer, and relieves him of much redundant coding.

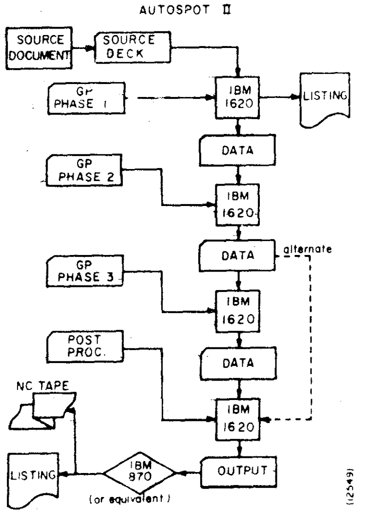


Figure 1. Autospot II Program Flow

Autospot II consists of a General Processor which is common to all machine tools and a Post Processor for each machine tool. The General Processor performs such operations as translation to numeric codes and machining pattern manipulation. The Post Processor tailors the output of the General Processor to suit the individual machine tool requirements. The input to the General Processor is a source statement card deck and the output from the Post Processor is a punched tape containing machine tool commands in the proper code.

2.0 THE NEED FOR A PREPROCESSOR

To reduce the number of passes required in the General Processor, error detection and diagnosis were limited to a minimum. Most source statement errors result in general error messages; however, many errors cause a computer check stop or hang-up condition, without an error message.

The lack of extensive diagnostic information is not a serious problem when the parts programmer has adequate experience with Autospot. Much time is lost, however, in identifying errors and the assistance of a 1620 programmer is frequently required. An inexperienced parts programmer often encounters so much difficulty with source program errors that much of the advantage of Autospot is lost. For example: If a parts programmer inadvertently substitutes a decimal point for a comma at the end of a coordinate dimension, the 1620 will hang-up in an infinite loop when the General Processor reads the statement. The parts programmer may then need the assistance of a 1620 programmer to locate the error.

3.0 PREPROCESSOR DESCRIPTION

The Autospot Preprocessor (see Figure 2) is a one pass program for the 1620 system. Its purpose is to detect and diagnose most of the errors which occur in Autospot II source statements. Error detection is sufficiently detailed to permit immediate recognition of most common errors, and to significantly reduce the time required to diagnose unusual errors. On-line editing permits immediate correction of most errors during preprocessing.

The Preprocessor is capable of detecting two types of error: format errors, such as typographical mistakes; and, violations of Autospot rules, such as illegal pattern manipulation. Most of the 35 possible error messages refer to rule violations because these errors are usually more difficult to diagnose than are errors of form.

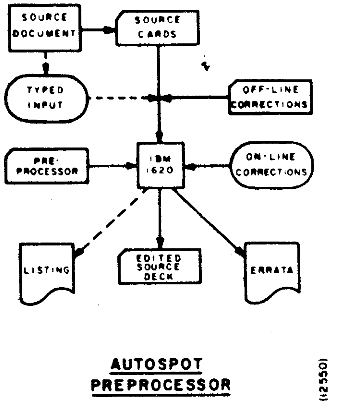


Figure 2. Autospot Preprocessor

EXAMPLE I

The use of too many machining patterns will result in the error message:

E COUNT PAT PAT I.

where PAT I is the symbolic label assigned to the pattern by the parts programmer. This error message is specific because the nature of the error is not readily evident.

EXAMPLE II

The use of the letters DQ instead of DP for specifying a depth will result in the error message:

C FORM AUX DQ.

This indicates a format error in the auxiliary section of the statement. The specific nature of the error is readily evident.

When an error is encountered, the preprocessor will type:

- one of the 35 different error messages,
- the entire erroneous line,
- the contents of the data field in question, and
- the punctuation terminating that field.

The data field or statement section is not necessarily in error, but this indicates that the error was recognized at that point. The actual error may appear anywhere up to that point.

3.1 PREPROCESSOR ERROR MESSAGES

The error messages are in abbreviated form and contain error type codes which indicate the corrective action to be taken. A blank code indicates that the typeout is for information only and requires no action. A "P" code indicates that the field in question is acceptable but unusual and is a possible error. An "E" code indicates that there is a definite error which cannot be corrected on-line, but which will result in an erroneous edited deck. A "C" code indicates that the error is definite but can be corrected on-line.

The computer takes no action on a blank or "P" coded error. The erroneous statement is deleted on an "E" type error but no halt occurs. Program switch settings determine the action on a "C" type error. If the editing feature is disabled, the error is treated as an "E" type error. If editing is required, a program halt occurs to permit correction or omission of the erroneous statement (at the discretion of the operator).

The non-stop mode of operation permits operators who are not familiar with Autospot to run the Preprocessor. The Preprocessor will also calculate effective drill lengths, a feature which reduces the number of calculations the parts programmer is required to make.

3.2 PREPROCESSOR ADVANTAGES

The effectiveness of the Preprocessor is illustrated by a test problem which was run at IBM Poughkeepsie. The part being programmed was an actual production piece, requiring approximately 3,000 lines of numerical controls for the Kearney and Trecker Milwaukee-Matic machine tool. The Autospot source deck required 126 lines, including 107 lines of machining statements. Using the Preprocessor, this large program was debugged in less than 30 minutes, of which only 14 minutes was 1620 computer time. This time included the initial run and two reruns after corrections (corrections being made off-line due to type"E" errors). The same source program was partially debugged,by the parts programmer who wrote it,in four hours. The experiment was then terminated and the job completed using the Preprocessor.

Little time is lost in running good source statements through the Preprocessor. Error free source cards are processed at the average rate of one line per second, assuming nearly full lines. If no errors are found, most source decks may be checked in less than one minute (including Program load time).

4.0 SUMMARY

The Autospot II General Processor provides limited error diagnosis. Because of this, many advantages are obtained by using the Autospot Preprocessor. The Preprocessor pinpoints most common errors and provides sufficient diagnostic information to significantly reduce the diagnosis time for unusual errors.

5.0 PREPROCESSOR DEBUGGING (SAMPLE)

Figure 3 shows the General Processor listing of a sample problem. Note error messages.

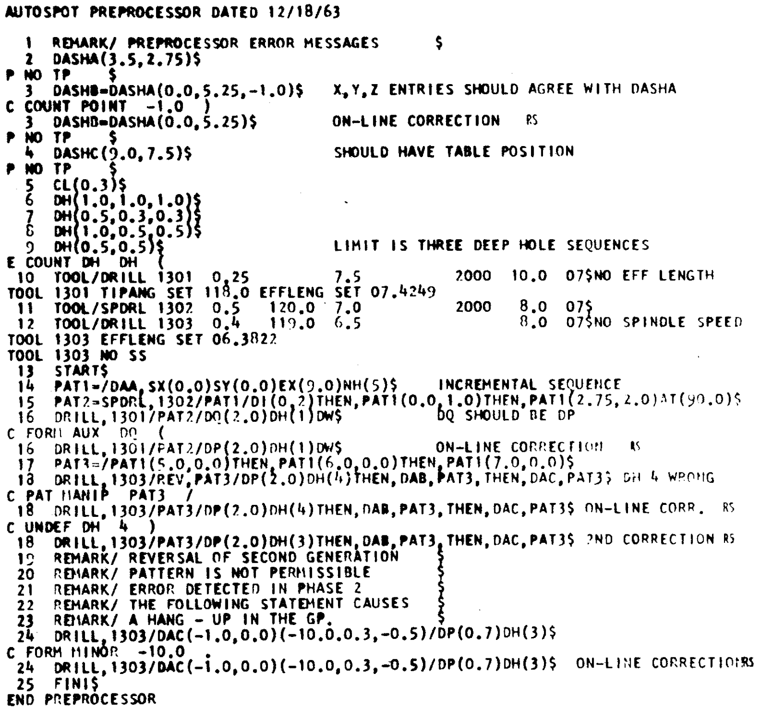
Figure 4 shows the output of the Autospot Preprocessor for the same program. Note error messages.

Figure 5 shows the rerun of the edited program deck.

```
 1  REMARK/ GENERAL PROCESSOR ERROR MESSAGES $
 2  DASHA(3.5,2.75)$
 3  DASHB=DASHA(0.0,5.25,-1.0)$   X,Y,Z ENTRIES SHOULD AGREE WITH DASHA
 4  DASHC(9.0,7.5)$               SHOULD HAVE TABLE POSITION
 5  CL(0.3)$
 6  DH(1.0,1.0,1.0)$
 7  DH(0.5,0.3,0.3)$
 8  DH(1.0,0.5,0.5)$
 9  DH(0.5,0.5)$                  LIMIT IS THREE DEEP HOLE SEQUENCES
10  TOOL/DRILL 1301  0.25          7.5           2000  10.0  07$NO EFF LENGTH
11  TOOL/SPDRL 1302  0.5    120.0  7.0           2000   8.0  07$
12  TOOL/DRILL 1303  0.4    119.0  6.5                  8.0  07$NO SPINDLE SPEED
13  START$
14  PAT1=/DAA,SX(0.0)SY(0.0)EX(9.0)NH(5)$    INCREMENTAL SEQUENCE
15  PAT2=SPDRL,1302/PAT1/DI(0.2)THEN,PAT1(0.0,1.0)THEN,PAT1(2.75,2.0)AT(90.0)$
16  DRILL,1301/PAT2/DQ(2.0)DH(1)DW$           DQ SHOULD BE DP
ERROR MINOR SECTION 16  DRILL,1301/PAT2/DQ(2.0)DH(1)DW$           DQ SHOULD BE DP
17  PAT3=/PAT1(5.0,0.0)THEN,PAT1(6.0,0.0)THEN,PAT1(7.0,0.0)$
18  DRILL,1303/REV,PAT3/DP(2.0)DH(4)THEN,DAB,PAT3,THEN,DAC,PAT3$ DH 4 WRONG
19  REMARK/ REVERSAL OF SECOND GENERATION    $
20  REMARK/ PATTERN IS NOT PERMISSIBLE       $
21  REMARK/ ERROR DETECTED IN PHASE 2        $
22  REMARK/ THE FOLLOWING STATEMENT CAUSES   $
23  REMARK/ A HANG - UP IN THE GP.           $
24  DRILL,1303/DAC(-1.0,0.0)(-10.0.0.3,-0.5)/DP(0.7)DH(3)$

          NOTE - INDICATED ERRORS WHICH CAUSED NO ERROR
                 MESSAGE ARE DETECTED IN PHASE 2, OR IN
                 THE POST PROCESSOR, OR NOT AT ALL.
```

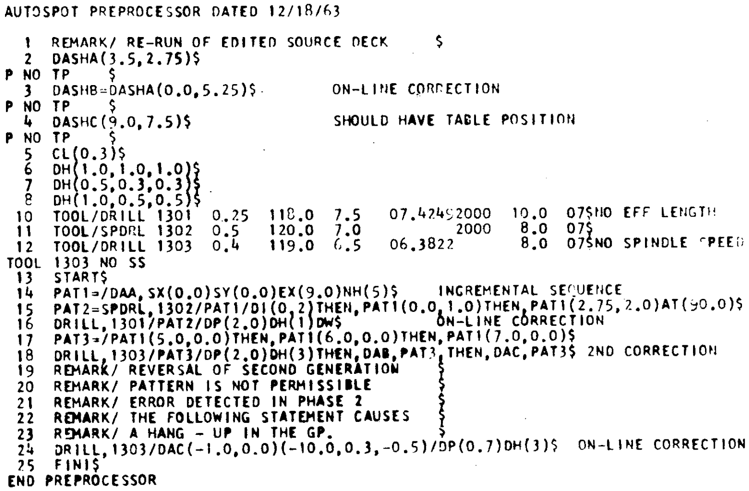
Figure 3. General Processor Listing of Sample Processing



```
AUTOSPOT PREPROCESSOR DATED 12/18/63

  1  REMARK/ PREPROCESSOR ERROR MESSAGES      $
  2  DASHA(3.5,2.75)$
P NO TP    $
  3  DASHB=DASHA(0.0,5.25,-1.0)$   X,Y,Z ENTRIES SHOULD AGREE WITH DASHA
C COUNT POINT  -1.0  )
  3  DASHD=DASHA(0.0,5.25)$        ON-LINE CORRECTION  RS
P NO TP    $
  4  DASHC(9.0,7.5)$               SHOULD HAVE TABLE POSITION
P NO TP    $
  5  CL(0.3)$
  6  DH(1.0,1.0,1.0)$
  7  DH(0.5,0.3,0.3)$
  8  DH(1.0,0.5,0.5)$
  9  DH(0.5,0.5)$                  LIMIT IS THREE DEEP HOLE SEQUENCES
E COUNT DH  DH  (
 10  TOOL/DRILL 1301  0,25         7.5          2000  10.0  07$NO EFF LENGTH
TOOL 1301 TIPANG SET 118.0 EFFLENG SET 07.4249
 11  TOOL/SPDRL 1302  0.5   120.0  7.0          2000   8.0  07$
 12  TOOL/DRILL 1303  0.4   119.0  6.5                 8.0  07$NO SPINDLE SPEED
TOOL 1303 EFFLENG SET 06.3822
TOOL 1303 NO SS
 13  START$
 14  PAT1=/DAA,SX(0.0)SY(0.0)EX(9.0)NH(5)$    INCREMENTAL SEQUENCE
 15  PAT2=SPDRL,1302/PAT1/DI(0,2)THEN,PAT1(0.0,1.0)THEN,PAT1(2.75,2.0)AT(90.0)$
 16  DRILL,1301/PAT2/DQ(2.0)DH(1)DW$           DQ SHOULD BE DP
C FORM AUX  DQ  (
 16  DRILL,1301/PAT2/DP(2.0)DH(1)DW$           ON-LINE CORRECTION   RS
 17  PAT3=/PAT1(5.0,0.0)THEN,PAT1(6.0,0.0)THEN,PAT1(7.0,0.0)$
 18  DRILL,1303/REV,PAT3/DP(2.0)DH(4)THEN,DAB,PAT3,THEN,DAC,PAT3$ DH 4 WRONG
C PAT MANIP  PAT3  /
 18  DRILL,1303/PAT3/DP(2.0)DH(4)THEN,DAB,PAT3,THEN,DAC,PAT3$ ON-LINE CORR.  RS
C UNDEF DH  4  )
 18  DRILL,1303/PAT3/DP(2.0)DH(3)THEN,DAB,PAT3,THEN,DAC,PAT3$ 2ND CORRECTION RS
 19  REMARK/ REVERSAL OF SECOND GENERATION    $
 20  REMARK/ PATTERN IS NOT PERMISSIBLE       $
 21  REMARK/ ERROR DETECTED IN PHASE 2        $
 22  REMARK/ THE FOLLOWING STATEMENT CAUSES   $
 23  REMARK/ A HANG - UP IN THE GP.           $
 24  DRILL,1303/DAC(-1.0,0.0)(-10.0.0.3,-0.5)/DP(0.7)DH(3)$
C FORM MINOR  -10.0  .
 24  DRILL,1303/DAC(-1.0,0.0)(-10.0,0.3,-0.5)/DP(0.7)DH(3)$  ON-LINE CORRECTIONRS
 25  FINI$
END PREPROCESSOR
```

Figure 4. Autospot Preprocessor Output



```
AUTOSPOT PREPROCESSOR DATED 12/18/63

  1  REMARK/ RE-RUN OF EDITED SOURCE DECK     $
  2  DASHA(3.5,2.75)$
P NO TP    $
  3  DASHB=DASHA(0.0,5.25)$          ON-LINE CORRECTION
P NO TP    $
  4  DASHC(9.0,7.5)$                 SHOULD HAVE TABLE POSITION
P NO TP    $
  5  CL(0.3)$
  6  DH(1.0,1.0,1.0)$
  7  DH(0.5,0.3,0.3)$
  8  DH(1.0,0.5,0.5)$
 10  TOOL/DRILL 1301  0.25  118.0  7.5   07.42492000  10.0  07$NO EFF LENGTH
 11  TOOL/SPDRL 1302  0.5   120.0  7.0          2000   8.0  07$
 12  TOOL/DRILL 1303  0.4   119.0  6.5   06.3822       8.0  07$NO SPINDLE SPEED
TOOL 1303 NO SS
 13  START$
 14  PAT1=/DAA,SX(0.0)SY(0.0)EX(9.0)NH(5)$    INCREMENTAL SEQUENCE
 15  PAT2=SPDRL,1302/PAT1/DI(0,2)THEN,PAT1(0.0,1.0)THEN,PAT1(2.75,2.0)AT(90.0)$
 16  DRILL,1301/PAT2/DP(2.0)DH(1)DW$          ON-LINE CORRECTION
 17  PAT3=/PAT1(5.0,0.0)THEN,PAT1(6.0,0.0)THEN,PAT1(7.0,0.0)$
 18  DRILL,1303/PAT3/DP(2.0)DH(3)THEN,DAB,PAT3,THEN,DAC,PAT3$ 2ND CORRECTION
 19  REMARK/ REVERSAL OF SECOND GENERATION    $
 20  REMARK/ PATTERN IS NOT PERMISSIBLE       $
 21  REMARK/ ERROR DETECTED IN PHASE 2        $
 22  REMARK/ THE FOLLOWING STATEMENT CAUSES   $
 23  REMARK/ A HANG - UP IN THE GP.           $
 24  DRILL,1303/DAC(-1.0,0.0)(-10.0,0.3,-0.5)/DP(0.7)DH(3)$  ON-LINE CORRECTION
 25  FINI$
END PREPROCESSOR
```

Figure 5. Edited Program Deck (Rerun)

MANAGEMENT INFORMATION

BY
ALBERT C. MAAS
DIRECTOR, OPERATIONS RESEARCH AND STATISTICAL ANALYSIS
GREEN GIANT COMPANY
LE SUEUR, MINNESOTA

PAPER PRESENTED AT THE
MID-WESTERN REGION - IBM 1620 USERS' GROUP
CHICAGO, ILLINOIS
FEBRUARY 21, 1964

1964
GREEN GIANT COMPANY
LE SUEUR, MINNESOTA

MANAGEMENT INFORMATION

There are different reasons why one may address himself to and accept the challenge of discussing, presenting, or reviewing a given topic. He may be an authority on the subject and discuss it in that capacity, or he may hold a position of responsibility in which the method may be applied and so discuss it from that point of view. When I was invited to present this paper, I was aware that my qualifications are not those of an authority but rather those of a practitioner in the field of Management Information. An interest in the subject and a recognition that there is applicability in present day business management provides motivation for preparing and presenting these remarks.

One way of organizing material for a presentation such as this would have been to search the literature and quote the authorities. I did not do this but rather elected to speak on the subject as I see it in general and somewhat specifically in my company. If a bibliography of writings on this topic were to be assembled, I am sure it would be impressive. I am also sure that the list of articles on the subject will grow at an increasing rate in the years before us and that our present state of knowledge and level of practice will be dwarfed by future developments.

My comments will be organized around the following points:

1. A review of what constitutes information.
2. A definition of terms that are associated with the subject.
3. A brief discussion of the functions that constitute the totality of company activity.
4. The state of the business world with special concern about the need for scientifically developed Management Information.

5. Relationships among Management Information, the computer, and personnel.

Information

In discussing this topic one must soon come to grips with what is meant by the word "information", and what the connotation must be when used in discussing business problems. A dictionary or academic definition may add a note of scientific precision to this presentation but, in keeping with the intent of the paper, it seems more appropriate to define the word by using it in context for a few paragraphs. It is the intent here to define its meaning as used in the language of business decision making.

Let us first agree that usage of the word suggests the addition of something new to the hearer's store of knowledge. Let us also agree that this information may be used as the basis for making decisions, either in the business world or in private lives; that it will be used as the basis for setting in motion courses of action. We must agree, if the preceding has been accepted, that information must have some value; that is, it must be appropriate, accurate, and it certainly must be timely.

We can quite likely agree that information, in part, may consist of reports, lists, graphs, comparisons, counts, or any other statements that something is or is not. In a moment we will be considering that information might be classified as available before an act takes place or after an act takes place, thereby giving it a form of time dimension.

A common example of information that is available after an act has taken place is found in performance measurements. These would include data on production to date, sales to date, costs incurred, capacity utilized, asset position, and liability position. Position records such as earning statements and the balance sheet could be looked to as other examples of information made available after an act has taken place. In all of these cases, the information is an accounting for something that has happened or a position that has been reached.

Information that is made available before an act takes place is of considerably greater interest, in this paper, than after-the-act information. Examples of this are found in a broad category of planning statements. The document or report that describes plans for expanding production capacity, for entering a new market, for assigning facilities, and others of this type constitute information before an act takes place.

It is the intent in this paper to consider an important and necessary element in the information that is generated in the before-an-act-takes-place category. This is the element that changes information from a recitation of facts to true decision-making support. We might think of this element as that property in the totality of information that answers the question about what the consequences would be to taking alternate courses of action. It is, therefore, to some extent, a plan and a prediction.

In past years and currently, the element has to a greater or lesser degree been supplied by management; that is, the decision maker. The advances made in methodology and equipment have given support to the organization of information-generating groups who, as technically trained management science practitioners, are able to provide this element. The change that this implies, within the area of Information Generation, may either be advocated or predicted. In either case it is taking its place on the scene of business operations.

The relative effort spent on developing information by the before and after categories is an indication of company vitality. Information after the act might be compared to the rear view mirror in an automobile. By looking into the mirror the driver is able to see where he has been. It follows that the more strategically located and the larger that mirror the better the view will be of where the driver has been.

Information before the act is then likened to the windshield of the automobile. If this is large and clear, the driver is able to see where he is going and can take steps necessary to get him there fast and safely. The relative size of the windshield in comparison to the rear view mirror is important in an automobile and it certainly is important in the operation of a business enterprise.

It must be recognized at this point that these remarks are concerned with degree rather than absolute lack of conformity to the concept being discussed. Historical information has and always will be used for preparing predictions, and analysts have always contributed toward producing that element within the totality of information upon which decisions can be based. It is argued, however, that considerably more effort should be directed toward using the analytical techniques known to management science personnel, and that this be used to generate decision-making information before it is given to management. The decision maker; that is, the manager, should be in a position to ask the question, "What will happen if I take this or if I take that course of action?" The management scientist, using the analytical tools available to him and working with historical data, is able to add that element which will make it a more reliable basis for planning courses of action within the business enterprise.

Definition of Terms

It will be convenient, for expository purposes, to define some terms and expressions that are used in discussing the activities in the business world. The definitions are not intended to be precise in the academic sense but rather as clarifications for the purpose of presenting views in this paper.

A. Business Problems

The activities of a business enterprise constitute a process in which the resources under the control of the business firm are used, in a production phase, to create added value and then to bring into a realization that added value through a distribution and marketing phase. Business problems exist because the total process does not operate without disturbances. Resources, including materials, supplies, facilities, and the skills of employees are limited and imperfect. There is resistance in the market to paying more than necessary for the products of the business enterprise and there is a constant need for attention to the mechanism of the production phase. The existence of these disturbances, as well as the need to plan for the growth of the enterprise, constitute business problems.

B. Courses of Action

These relate to the steps that are and must be taken by management to correct a business problem. The course of action is therefore simply the doing of something, the execution of the plan that resulted from a management dicision.

C. Dynamic

A moment's reflection on the business problem and its resolution in a business enterprise suggests that many problems occur repeatedly. In fact, it will soon be observed that the majority of the operating problems are recurring. The frequency with which they occur and especially the speed with which they can be resolved play an important role in the competitive position of a business enterprise. These observations partially provide the basis for describing a business, especially as measured by its problems, as dynamic.

D. Information Retrieval

This term relates to a fairly well-defined process of cataloging the content of articles, abstracts, books, and the like and for providing a means for locating the document, or a brief statement of its content, in response to the user's need. Management Information, the title of this paper, is not related to Information Retrieval except as the latter may be a part in the process of generating decision-making information for management. An issue is made of this comparison since there is a possibility for confusion, the belief that the ability to rapidly extract data from files will serve the need of management for information.

Functions of a Company

The totality of activity associated with the operation of a company can be categorized in various ways. To focus attention on a specific function, the generation of information, four categories are formed, which, by definition should include all the activities that can and do take place in a business enterprise. These categories include: 1. Production-Marketing-Distribution. 2. Recording and Control. 3. Decision Making. 4. Information Generation.

Production, marketing and distribution, the operations function in a company, include the obvious activities of utilizing facilities and resources to produce something, to market it, and to move it through the distribution channels into the consumer's hands. In a processing industry such as the canning industry, this will appear as, and in fact is, the dominant function of the enterprise.

The activities of maintaining company operations records, company operating plans, and providing a measure of performance against plans constitute, in part, the function of recording and control. There is

obviously considerably more that could be said about this and about the operations function, but since it is the objective in this paper to discuss Information Generation, further elaboration on these other functions will be omitted.

Decision making is a function that is executed at all levels of company operations. As a first impression it appears that this might be a function reserved for the top executives. This is not true, however, since the worker on the line must, and does, make decisions, or at least apply a measure of judgment, in operating a piece of equipment or using a resource. Top executives make decisions about such matters as finance, plant or production expansion, personnel assignments and the like. The vast majority of the decisions in any business enterprise, however, are made by the operating and management personnel between the line worker and the top executive. In any case, the decisions at all levels must be appropriate and they must be timely. The skill with which this function is executed will be reflected in the effectiveness of the operations function and also in the effectiveness of recording and control.

If it is agreed that setting in motion the appropriate courses of action at the different levels in a company is dependent on the quality and timeliness of decisions, the foregoing statement is obviously supported.

The function to be discussed in greater detail in this paper is that of Information Generation. The importance of this is underscored by recognizing that the function of Decision Making is not executed in a vacuum, it is not independent of the other functions. A course of action within the operations function is not put into motion unless there has been a decision to do this and unless there has been a decision to commit certain of the company's resources.

The basis upon which a decision is made, however, is that of the information available to the decision maker. Information about the process and information developed in the planning sense must be offered to the manager, the person who will translate it into a decision. It is at this point that the function of Information Generation achieves its significance.

State of the Business World

Business decisions are not made in a vacuum and business enterprises are not operated independently of the business world environment. One of the characteristics of the business world, it is contended, is that changes are taking place rapidly and that the function of decision making, as a result, is becoming increasingly complex.

If we accept as true that there is, in fact, a rapidly changing climate in the business world, then we must also accept that the advanced techniques for coping with these changes must be developed and applied. It is especially required that support for decision making be made available accurately, adequately, and timely. This constitutes the heart of the total Management Information idea.

Technology in problem solving has changed and has improved very rapidly during recent years. The mathematical methods of linear programming, critical path analysis, inventory control, estimating, forecasting, and many others have been developed, improved, and made available to management. Books, articles, courses, and seminars have been employed during the years since World War II to disseminate the information.

It is of special interest to observe that the mathematical techniques, if considered by themselves, are of limited value. These techniques must be a part of the total problem solving system if they are to be of service to a business enterprise. The process of information generation is built around this concept. It is one of the objectives in this paper to demonstrate that information, in addition to being a record or recitation of events that have taken place, also includes those elements of information that will point up the most ideal steps that can be taken in the decision-making process.

Another characteristic of the business world today, in comparison to past years, is the intensification of competition and its attendant problems. New products are coming on the market at a faster pace and the costs of developing them are higher than was true several years ago. The advantage to the company developing a new product, it is contended, is either short lived or the margin of profit is narrow. This is the result of competition not only among manufacturers of the same product, but among all manufacturers competing for the consumer's dollar. This underscores the necessity for having pertinent information available to management, information that can be used as a basis for rapidly formulating decisions and effecting courses of action. The significance of these observations is in the necessity for much faster action than in previous years, and for fewer mistakes in committing companies' resources to an operations course of action.

A single development that has been instrumental in stepping up the pace of business activity, and has also been providing a means for servicing the stepped-up pace, is that of the computer and the technology for programming and operating it. This combination of equipment and technology has made possible the rapid processing of voluminous data, as well as analyzing data complexes such as are common to the management science field. The reduction of voluminous records and the evaluation of complex sets of data provides a source of information that has not been available in the past.

In addition to the data processing equipment, there has been development in communications which makes possible real time or near real time data analysis for decision making. All of this clearly dictates the need to develop a system through which the tools of information generation can be employed most effectively. It must be possible to develop clear and concise elements of information that can be used in the decision-making process with a minimum of further analysis or data reduction by the user; that is, by the decision maker,

A logical consequence of the foregoing is that management by exception will be and must be practiced. It is not possible and certainly not necessary for a manager to weigh all the facts that can be developed by an information generation system. Rather, he must be given those elements of information to which he can add his skills and thereby reach the decisions that are most beneficial to the company.

It is also necessary that the Information-Generating process produce facts that can be translated directly into routine courses of action. A certain percentage, perhaps quite high, of this type of tasks in a company can be reduced to decision rules that can be operated upon by an electronic computer or, at most, require clerical attention. The net result is that this will leave additional time to the manager to deal with the more complicated decision problems, problems that cannot, or at least not very readily, be reduced to a decision rule.

The intent of these comments has been to demonstrate that the role of the manager, decision maker, is changing rapidly as a result of the technological advances. The business climate within which the decision maker works is being changed by him and in turn requires that he change with it. He is, in a sense, a victim of his profession.

Relationships Among Management Information, The Computer, and Personnel

The remarks to this point have been intended primarily to set the stage for a detailed review of the Information Generating function, its place in the company, and the impact it may have upon the Decision-Making process. Its impact upon the personnel involved and a review of the current state of the art will be considered briefly. As a point of departure, it will be well to take a look at what is meant by "Information Generation."

It certainly is a function and it has a place among those that define the totality of company activity. Information generation is not new. Rather it has been practiced as long as businesses have been operated. The method for doing it, especially its organization within the company, have changed over the years and the importance it has played and plays now is certainly changing. It shall be the objective in the following sections of this paper to present views as to what constitutes Information Generation, how it has changed over time, and what might be expected in the future.

We may think of this function as an operating process with inputs, service, and output stages. This analogy with the operations functions of a company will provide a convenient medium for presenting some of the basic ideas.

The inputs to the process initiate at various sources. Company accounting records provide data on costs such as those for personnel, power, raw materials, supplies, and others. Operating standards, capacities, and facilities availability data can generally be obtained from company engineering records. Prices of merchandise offered for sale become available from the company's marketing department.

Institutional data constitute another input to the process. These would include such items as taxes, insurance, interest rates, freight rates, economic indicators, and the like. Agency data, such as facts about industry stock position, and industry prices provide a third source of input. A fourth source would need be recognized to include estimates by knowledgeable persons. There are many blanks in the data requirements associated with a given analysis, blanks that must be filled before the analysis can be made. In many cases the best estimates of knowledgeable persons will constitute the total availability of this type of input information.

The input information is directed into the service phase of the information-generating function, an area designed for and increasingly delegated to the management science personnel. By way of contrast it might be observed that the service phase could be limited to the organization of data into reports, tabulations, graphs, ratio tables, and the like. This service could be and likely would be provided by the general accounting or by the cost accounting groups of the company. The management science personnel, however, are, or at least should be, qualified to add that element to the information flow which changes it from a presentation of history to a basis for deciding upon a course of action.

The management science contribution at this point should therefore be to work with operating personnel, decision makers, and upon recognition of a business problem, define and formulate it for the analysis phase. After the problem has been defined, it is obviously required that the actual solution be effected and the results prepared in a form that will be most useful to the decision maker.

There is an impressive array of analysis tools available to the management scientist with which he is able to cope with the complexities of the problems to which reference was just made.

It is not the intent in this paper to discuss in detail the analysis tools that are available. It is rather the intent to describe some of the characteristics of the analysis methods and to support a claim that many and powerful tools of this type are available. Some of the characteristics, with which these analysis techniques can cope, are:

1. There are involved, inter-relationships among the factors of the problem. These are inter-relationships that cannot be dealt with readily by means other than an appropriate mathematical formula and the necessary computing facilities. An example of such a problem is the one in which shipping schedules are formulated. The factors of this problem are the supply of the homogeneous product at a number of origin points, the demand for the product at a number of destination points, and the shipping cost per unit for moving a unit of the product from a point of origin to a point of destination. The objective in the solution is to find that combination of routes which, if followed, will transfer the merchandise from the points of origin to the destinations at the lowest possible total freight cost. In working with problems of this type, it is soon found that interaction frequently necessitates the use of the rates, other than the lowest because, if this were not done, another rate of even greater disadvantage would be forced into use. This is all brought about by the complex inter-relationships of the factors in the problem. Solution to a problem of this type is brought about readily with the analytical tool known as the Transportation Model.

2. In these problems there is either a maximum or a minimum that must be found and that serves as a criterion in evaluating the solution. In the Transportation Model, the minimum freight bill is found, whereas in another type of analytical tool a maximum profit might be found.
3. The solutions to problems may lead directly to the application of results in a routine type course of action or they may lead to alternative courses of action in a planning type analysis. In the latter case various conditions might be evaluated through a simulation of the process.

In direct solutions there must have been a prior implementation of the procedure so that the results of a given analysis can be fed directly to it. This is a form of automated decision making.

In another case, the output of the information-generation process, frequently involving simulation, takes the form of a report to management.

The manager or decision maker receives this information and adds to it his knowledge of the process. This, then, is the basis upon which decisions about a course of action can be made.

In the discussion of the service phase of this function it was pointed out that direct solutions might be used in implementing courses of action where a procedure has been implemented and where the course of action is routine. In those cases where that is not done, there is management by exception; that is, the manager is concerned with those steps in the operation of a business that cannot be processed or put into force through decision rules programmed into an electronic data processing system,

The output from the service phase of the information-generating process may therefore take two forms. It may be a decision rule that can put into effect routine courses of action through the medium of the data processing system or the intervention of a clerk. In the other case, and in a more important sense, the output will be guides for personnel in the decision-making function who will act to initiate those courses of action that are associated with planning and the operations function of the company.

A system does not function without people, and therefore, consideration must be given to the personnel involved in the Information-Generating function. Just as there is no clear distinction between persons involved in the decision making and in the operating functions, there is also no clear distinction among persons involved in information generation and the other functions in the company. It is rather to be found that the persons in the company are or should be aware of this function and become associated with it in whatever position they may hold. They may be involved directly, as suppliers of data, as a user of the output, or in a capacity that is a combination of these.

A logical way to establish who is part of the information-generating function and what the relationship between those persons and others outside that function is, is to consider this in the light of information flow. A look at the input-service-output analogy discussed in the preceding section will provide some guidance.

The output of the Information-Generating function is the input to the Decision-Making function and takes the form of reports that have been developed from prime data. The prime data is the input to the Information-Generating system. The personnel involved, therefore, include those responsible for supplying data from prime records, those who analyze the data, and those who deliver the output to the decision-making personnel.

A question that can and must now be considered is concerned with the relationship between accounting and management science personnel. If an integrated and consistent flow of information is to be generated it is not reasonable to expect that some reports into the decision-making process shall originate in the accounting group and others in the management science group. There can be no guarantee that such an arrangement will assure consistent and noncontradictory information. It creates the possibility of sending still picture type of information into the Decision-Making process when the dynamics of the business call for information of the motion picture type. The conclusion that follows from these comments is that the Information-Generating function must be organized and managed in such a way that it will assure the generation of the most valuable information possible and that it will be sent in its most appropriate form into the Decision-Making function.

The comments made in the preceding paragraphs suggest that there might need be a change in the concept of information generation today as compared with that applying in past years. The idea of information generation is not new, but some concepts associated with the total management information methods has in it aspects to which there must be adjustment by the personnel involved in that function. Some observations about the difference of concept may be itemized as follows:

1. Reports based on individual studies could be, and many times should be, replaced by information logs derived from a series of simulation analyses. This replaces the static snapshot report with the dynamic motion picture type report.
2. Reports of individual projects will be, and certainly can in many places, be replaced by the results of team effort. Team effort has in its favor, many attributes even though it does carry with it the problem of rivalries, and other problems associated with having persons work as a team.
3. A greater reliance will be placed on decision rules programmed into the data processing system. This will be true partly because of the much greater magnitude of data that needs be reviewed and also because of the analytical and data processing techniques that are available for accomplishing this. This will lead to greater emphasis on management by exception.
4. The environment or climate within the company must be created in which the Information-Generating function can be executed effectively. Managers must realize that the working paper study or report cannot and does not give them all the information they need for decision-making responsibilities. The manager must also learn to accept that a large part of the routine decisions for which he may be responsible can be processed on electronic equipment. The reluctance to relinquish detailed control over the activities for which he is responsible can prove to be one of the greatest hindrances in establishing a management information system.

5. It must be recognized that the electronic data processing equipment can serve a purpose much greater than that served in billing, processing accounts receivable and accounts payable, recording inventory and the like. The electronic equipment properly managed by technically trained management science personnel can produce that element in the Information-Generating function that could tip the scale from mediocre to high level and effective decision making.

In summary, let us conclude that management information is the product of our efforts which, when coupled with a well-executed Decision-Making function, puts into effect the correct courses of action with respect to business problems, and which in turn find expression in profit generation.

KINGSTON FORTRAN II

FOR THE IBM 1620 DATA PROCESSING SYSTEM

by:

J.A.A. Field,[1] D.A. Jardine,[2], E.S. Lee,[1]

J.A.N. Lee,[3] and D.G. Robinson[2]

Presented at the Joint Canadian-Midwest Region
Meeting of the 1620 Users Group, Chicago,
February 19-21, 1964

[1] Dept. of Electrical Engineering, University of Toronto, Toronto, Ontario.

[2] Research Centre, Du Pont of Canada Limited, Kingston, Ontario

[3] Computing Centre, Queen's University, Kingston, Ontario

ACKNOWLEDGEMENTS

During the early stages of developing this system, many people in the 1620 Users Group were canvassed for useful ideas on compiler and systems construction. To all those who, in any way, contributed to this venture, the authors extend their heartfelt thanks.

We would like to recognize the following people who made particularly useful contributions to the project:

J.W. Holmes[1] - for his extremely well written arithmetic and function subroutines which appear, with some modification, in this system.

F.H. Maskiell[2] - for many helpful suggestions, particularly in the coding and testing of the arithmetic and function subroutines.

C.H. Davidson[3] - 1620 Users Group representative on the A.S.A. Fortran II subcommittee, for explaining to us the structure of American Standard Fortran II, and for pointing us in the right direction for extending the language.

[1] Cooper-Bessemer Corp., Mount Vernon, Ohio.

[2] McGraw-Edison Corp., Penn. Trans. Div., Canonsburg, Pa.

[3] University of Wisconsin, Madison, Wis.

HISTORY

The writing of compilers seems to be one of the more popular pursuits of the members of the 1620 Users Group. At least six different FORTRAN compilers for the 1620 have been written by non-IBM personnel, which testifies to the enthusiasm and ability of 1620 users and to their very real desire to build the best possible mousetrap.

All previous user-written compilers have accepted variations of the FORTRAN I language, with the exception of the University of Wisconsin FORGO, a load-and-go compiler for student problems, which accepted a somewhat restricted FORTRAN II. To our knowledge, KINGSTON FORTRAN II is the first user-written FORTRAN II for the 1620. We hope that this initial effort will encourage others to tackle the problem and improve on our system in the same way that improvement followed improvement in the user-written FORTRAN I compilers.

The initial impetus for KINGSTON FORTRAN II came in about August 1963, from those of us living in Kingston, Ontario, when we started to find out how UTO FORTRAN operated, with the intention of providing a suitable FORTRAN for a 40K 1620. It soon became apparent that many useful features of FORTRAN II could be incorporated at little extra work. Messrs. Lee and Field, authors of UTO FORTRAN, were approached for ideas and suggestions, the outcome of which was a decision to join forces. After some preliminary discussion, it was found that it would be no more work to write a whole new system than to make the desired alterations in UTO FORTRAN.

The basic concepts were conceived in three rather long evening sessions during the October 1963, 1620 Users Group Meeting in Pittsburgh, Pa. By the end of this meeting the source language structure and the organization and general logic of the compiler were developed and agreed upon. The various sections were then allocated to the individuals best qualified to handle them. By the first week in January, the main sections of the compiler had been written and tested and it remained to tie the pieces together in a operating system. This was done in Kingston, Ontario, during late January, when all 5 authors worked for five days on two identical 40K 1620's (Du Pont of Canada and Queen's University).

We hope that Users with 40K 1620's will find the system useful and easy to operate. We have tried to include every useful idea from other people's efforts so that the system would be as speedy and compact as possible.

The work was divided as follows:

J.A. Field - Input/Output statements, DO statements, input/output subroutines, FORMAT statement.

D.A. Jardine - Arithmetic and function subroutines, write-ups and operating manuals.

E.S. Lee - Compilation of arithmetic expressions.

J.A.N. Lee - Compilation of everything not handled by the other authors.

D.G. Robinson - Symbol table organization, including COMMON, DIMENSION, EQUIVALENCE, TYPE.

KINGSTON FORTRAN II

This write-up describes a FORTRAN system for the IBM 1620 equipped with automatic division, indirect addressing, additional instructions (TNS, TNF, MF), card input-output and minimum 40K memory. It is assumed that a Model E-8 or larger 407 is available for listing.

The language is that of IBM's FORTRAN II with a few modifications and a number of additions. For the purposes of this write-up it is expected that the reader is at least on speaking terms with the FORTRAN II language.

The compiler for this system batch compiles a source program in one pass, at approximately twice the speed of existing compilers for the 1620. The execution speed of the object program is also approximately twice that of IBM's FORTRAN II. Considerable effort has been made to speed up all important parts of the system; in addition, more core storage is available for the object program than existing FORTRAN II compilers allow.

SOURCE PROGRAM CARDS

These are as required for IBM FORTRAN II. Any number of continuation cards are possible, but the statement may not contain more than 300 characters (blanks not included except in Format statements).

ARITHMETIC PRECISION

Real numbers: 8 digit mantissa, 2 digit exponent.

Notation is excess 50; (i.e. 1.0 ≡ $\bar{5}110000000$)

Integer numbers: 4 digits, modulo 10000

VARIABLES

These are as in IBM FORTRAN II. 1 to 6 alphabetic or numeric characters, starting with a letter, which, for integer variables, must be one of I, J, K, L, M, N, unless otherwise specified in a TYPE declaration.

SUBSCRIPTS

A variable with, at the most, two subscripts appended to it can refer to an element of a one- or two-dimensional array. Three dimensional subscripting is *not* permitted. *A subscript may be an expression of any* desired complexity, provided only that the result of the evaluation of the expression be an integer quantity. This should be positive if you want to avoid trouble. However, a zero or a negative subscript can be used. To use this effectively, the programmer must know how data areas are laid out in memory. See the operating instructions:

Examples of Subscripts:

```
I
3
2+MU
MU+2
J*5+M
5*J
6*J-K+2-10/L+M
4*J(K+2-L+M)+K(M(N+2))/3
FIXF(A*B+3.0**SIN(X))+L/2
```

The variable in a subscript may itself be subscripted, and this process of subscripting may be carried on to any desired depth of subscripting. It can, in fact, be carried far beyond the point where the average programmer understands what he is doing.

SUBSCRIPTED VARIABLES

Only singly or doubly subscripted arrays may be defined. The size of these must be specified in a DIMENSION statement.

EXPRESSIONS

These are defined and organized exactly as in IBM FORTRAN II.

LIBRARY FUNCTIONS

Ten library (closed) functions are included in the KINGSTON FORTRAN II System. These are listed in Table I.

TABLE 1

Closed Subroutines

Function	Function	No. of	Type Of	
Definition	Name(s)	Arguments	Function	Argument
Sine of the argument	SIN	1	Real	Real
Cosine of the argument	COS	1	Real	Real
Exponential (e^x) of the argument	EXP	1	Real	Real
Natural logarithm of the argument	LOG	1	Real	Real
Arctangent of the argument	ATAN	1	Real	Real
Arctangent of (arg_1/ arg_2)	ARCTAN	2	Real	Real
Signum of the argument; =-1.for X<0.,=0. for X,0.,=+1. for X>0.	SIGNUM	1	Real	Real
Absolute value of Arg 1 with the sign of Arg 2	SIGN	2	Real	Real
Choosing the larger value of the two arguments	AMAX1	2	Real	Real
Choosing the smaller value of the two arguments	AMIN1	2	Real	Real

Table 2 lists the open or built-in functions. These are compiled in-line every time the function is referred to.

TABLE 2

Function	Function	No. of	Type of	
Definition	Name	Arguments	Function	Argument
Absolute value of the argument	ABS	1	Real	Real
	ABS	1	Integer	Integer

Table 3 lists closed functions which are permanently stored in the machine, whether or not they are mentioned by name in a FORTRAN source program.

TABLE 3

Function Definition	Function Name	No. of Arguments	Type Of Function	Type Of Argument
Floating an integer	FLOAT	1	Real	Integer
Truncation, sign of argument times value of the largest integer in the argument	FIX	1	Integer	Real

THE ARITHMETIC STATEMENT

The arithmetic statement is the same as in IBM FORTRAN II except for the extensions in complexity of evaluation of subscripts.

CONTROL STATEMENTS

The control statement flexibility in standard FORTRAN's leaves something to be desired, particularly where the program is complex and core storage is at a premium. These conditions, it might be noted, are the normal ones for almost all problems. KINGSTON FORTRAN II attempts to improve this situation by expanding the capabilities of the ASSIGN and assigned GO TO statement and by extending the ASSIGN concept to the other control statements.

ASSIGN STATEMENT

```
        ASSIGN i to n
```

In IBM FORTRAN II, the ASSIGN statement is used only in conjunction with an assigned GO TO statement. For instance,

```
        ASSIGN 3 TO J
        GO TO J, (3,5,9,243)
```

will cause a branch to the statement numbered 3.

The effect of the ASSIGN statement is to "equate" the non-subscripted integer variable J to statement number 3. The subsequent GO TO J, (3,5,9,243) is then interpreted as GO TO 3.

In KINGSTON FORTRAN II, this concept has been modified and expanded considerably. To describe these changes, the following definitions are used:

Statement Label - A statement label is the name attached to the memory location containing the first instruction compiled from the statement identified by the label. There are two kinds of statement labels:

Numeric Statement Label - usually known as a statement number. An unsigned integer number of from one to four digits long.

Alphabetic Statement Label - A variable which may be subscripted to any desired complexity and which by one or more ASSIGN statements has been equated to a numeric statement label (statement number).

It is most important to realize the difference between a statement label and an arithmetic variable. ASSIGN 3 TO J will place in J the address of the first instruction compiled from statement number 3. J = 3 will cause the number 0003 to be placed in J. The sequence of statements

```
ASSIGN 3 TO J
GO TO J
```

will cause a branch to statement numbered 3. However,

```
J = 3
GO TO J
```

will result in disaster. Moreover,

```
ASSIGN 3 TO J
J = J + 1
GO TO J
```

will not transfer control to the statement numbered 4. Arithmetic on assigned variables is not permitted; assigned variables are not in any way the same as arithmetic variables, except that they may be subscripted and stored in an array. They may also appear in COMMON, DIMENSION, and EQUIVALENCE statements.

It is possible in KINGSTON FORTRAN II, to equate two alphabetic statement labels by an ASSIGN statement. If the first statement label in the ASSIGN statement is alphabetic, it must be enclosed in parentheses.

The following examples illustrate the ASSIGN statement:

```
ASSIGN 3 TO N     (St. label N is equated to St. label 3)
ASSIGN (N) TO J   (St. label J is equated to St. label N)
ASSIGN 3 TO I(K)  (same as the line above.  J must have been
                   defined before this statement and I must be
                   dimensioned).
ASSIGN (I(K))  TO  L(3+M/4-M**3)
                  (same as above.  The alphabetic statement
                   labels can be subscripted as desired).
```

Since the primary definition of a statement identifier is its occurrence as a statement number, it is necessary that any given statement identifier must ultimately be defined (through a series of ASSIGN statements if necessary) in terms of a statement number. Failure to observe this rule will cause trouble. For example,

```
        3    A = B
             —
             ASSIGN (J) TO K(L)
```

is not correct, because J has not been associated with any statement identifier when the ASSIGN statement is executed. However,

```
        3    A = B

             —
             ASSIGN 3 TO J
             —
             ASSIGN (J) TO K(L)
             —
```

is correct.

Alphabetic statement labels may be used in the following control statements:

GO TO (both unconditional and assigned)
IF (SENSE SWITCH i)
IF (arithmetic expression)
Computed GO TO

Alphabetic statement labels may not be used in a DO statement.

GO TO STATEMENT

GO TO n   unconditional GO TO

GO TO n, $(n_1,n_2,---n_m)$ assigned GO TO where n is a statement label. If n is alphabetic, then it must previously have been defined in an ASSIGN statement. The assigned GO TO statement is treated exactly like the GO TO statement. The comma and parenthesized list are optional and will be accepted but ignored by the compiler.

Computed GO TO Statement

GO TO (n_1,n_2,n_3---n_m),i

where n_1,n_2---n_m are statement labels. If alphabetic they must have been previously defined by ASSIGN statements. i is a fixed point (integer) variable or expression. i may be subscripted as desired.

ARITHMETIC IF STATEMENT

IF(a)n_1,n_2,n_3

where a is an integer or real (floating point) expression of any complexity, and n_1,n_2,n_3 are statement labels. If alphabetic, n_1,n_2,n_3 must have been previously defined in ASSIGN statements.

IF (SENSE SWITCH) STATEMENT

IF (SENSE SWITCH i)n_1,n_2

where i is a one or two digit unsigned integer number or an integer expression, and n_1,n_2 are statement labels. If i is an integer expression, the low order two digits of the value of the expression are used as the value of i. The two digit numbers resulting from this are the numbers of machine indicators, not just console switches.

THE DO STATEMENT

DO n i = m_1,m_2,m_3

where n is a statement number, i is an unsigned integer variable which may be subscripted and m_1,m_2,m_3 are integer variables or integer expressions of any desired complexity, positive or negative. n may not be an alphabetic statement label, and i may not be an expression. There are no particular restrictions on m_1,m_2,m_3. In particular they may be positive or negative quantities. If m_1=m_2, the DO will be executed once only. m_1,m_2,m_3 should be chosen so that the DO loop terminates. See below for an example of a never-ending DO-loop.

Example:

```
DO 5J = K+L-5, M-I(JOB(KK)),-L
```

If m_1,m_2,m_3 are expressions, their values are the values of the expressions when the DO statement is encountered at object time, and these values are unaffected by alteration inside the DO of the values of the variables in the expressions m_1,m_2,m_3.

As a result of allowing positive or negative values for m_1,m_2,m_3, it is legal to have DO loops which count down. For example,

```
DO 3 I = 10, 1,-1
```

will cause I to run from 10 to 1 in steps of (-1). The following is also permitted.

```
DO 10 J = -10,5,2
```

which will cause J to assume successively the values -10, -8, -6, -4, -2, 0, 2, 4. If the DO variable assumes zero or negative values, it may be used, with caution, as a subscript. Intelligent use of negative or zero subscripts demands knowledge of the layout of data areas in memory, as described in the operating instructions.

Care should be taken to see that the DO index terminates properly. For instance,

```
DO 20 K = -10, -1, -2
```

will increment nearly 5000 times before termination. The same is true of

```
DO 40 K = 10, 1, 2
```

Termination in both cases occurs because integer arithmetic is performed modulo 10000.

All the restrictions on DO statements currently imposed by IBM FORTRAN II are also in force in KINGSTON FORTRAN, except as already mentioned.

CONTINUE STATEMENT

Same as IBM FORTRAN II.

PAUSE STATEMENT

```
PAUSE
```

PAUSE n, where n is a fixed point constant, variable or expression.

The typewriter types PAUSE n, together with error messages (see operating instructions) and the machine halts. If n is a variable or expression, its current value is typed. PAUSE (without n) generates an in-line halt command; there is no typing. In either case, depression START will cause resumption of program.

STOP STATEMENT

STOP

STOP n, where n is a fixed point constant, variable or expression.

The typewriter will type STOP, followed by the current value of n. If n is not specified, STOP 0000 will be typed. CALL EXIT is then executed (see operating instructions).

END STATEMENT

END is an instruction to the compiler that the program is complete. An END statement must be physically the last card of the main line program and of each sub-program which is associated with the job. The END statement results in CALL EXIT except in a sub-program, where it is interpreted as a RETURN statement.

FUNCTION AND SUBPROGRAM STATEMENTS

FUNCTION and SUBPROGRAM statements are the same in KINGSTON FORTRAN as in IBM 1620 FORTRAN II, and the same restrictions apply.

Because the compiler is one-pass, the subprograms are not compiled separately from the main program. See the operating instructions for further details.

INPUT/OUTPUT STATEMENTS

The INPUT/OUTPUT statements in KINGSTON FORTRAN II are similar to those of IBM FORTRAN II, except that expressions are permitted, as well as simple variables, in certain places in INPUT/OUTPUT lists. Indexed lists, array names (to handle a whole array) and all other standard FORTRAN II features are allowed. It is not necessary to specify a FORMAT statement number in an I/O statement. If no FORMAT statement number is given, the system will supply FORMAT (5N). See the description of FORMAT for an explanation of FORMAT (5N).

The permitted INPUT/OUTPUT statements are:

READ (card input), ACCEPT TAPE, ACCEPT (input on console typewriter), REREAD (re-reads last input record), PUNCH, PUNCH TAPE, TYPE (console typewriter), PRINT (on-line printer).

Indexed I/O Lists

As in IBM FORTRAN II, the statement

```
READ 10, ((A(I,J), I=1,10), J=1,10)
```

will cause 100 numbers (A(1,1) to A(10,10) to be read into array A. Similarly,

```
READ 10,((A(I,J), I=K,L),  J=M,N)
```

will cause various elements of A to be read in under the control of the indices I and J.

In KINGSTON FORTRAN II, the limits on the implied DO's (I=K,L; J=M,N) may be expressions. Furthermore, the names of the input variables may be subscripted to any desired depth (not exceeding 40). For example:

```
READ 10,((A(I(K1), J(M1), K1=K-JOB*2,L+5-J6),M1=M*8-MM9,N-3*N18)
```

will be executed as

```
        DO  100 M1 = M*8-MM9, N-3*N18

        DO  100 K1 = K-JOB*2,L+5-J6

    100 READ 10, A(I(K1), J(M1))
```

where I and J are names of one-dimensional arrays which must previously have been defined.

KINGSTON FORTRAN II permits the same kinds of expressions in indexing as are permitted in standard DO statements. The implied DO in and I/O list may run forward or backward, and may have integer expressions of any desired complexity.

INPUT LISTS

In an input list, the variables may be only simple variables or indexed variables. Input of expressions is meaningless, and not permitted. For example:

```
READ 10, M, Q, A(I(K+4*L), M(N-5*L+4)),B
```

is permitted, provided I, K, L, N and M are previously defined.

READ 10, A+B-C(K) is not permitted.

OUTPUT LISTS

Output lists may be fully indexed lists, as described above. In addition, expressions may appear in the list as output quantities. For example:

```
PUNCH 20, C*D/(LOGF(X-Y*Z)+10.3, Y, D
```

will cause

```
C*D/LOGF(X-Y*Z)+10.3
```

to be calculated at the time the punch statement is encountered and its value to be punched, together with the values of Y and D, on a card, according to Format statement 20. The value of the expression in an output list is lost when it is output, and is not available for further calculation. The expression in an I/O list may be of any desired complexity, and may be indexed as required, either by DO statements, or by implied DO statements in the list itself. For example:

```
PUNCH 20,(((C*SQRTF(A(I,J))-M(I)),I=1,L+4,3),J=I+1,K-10,5)
```

will cause values of C*SQRTF(A(I,J))-M(I)

to be punched out for values of J from I+1 to K-10 in steps of 5 and values of I from 1 to L+4 in steps of 3.

ASSIGNED FORMAT NUMBERS

Format statement numbers may be assigned by ASSIGN statements in the same way any other statement number can. Hence, input/output statements may use alphabetic statement labels in place of Format statement numbers. For example, the following program is permitted:

```
3     FORMAT (5(I3,F10.5))
4     FORMAT (5I5)
5     FORMAT (5I7)
      ASSIGN 3 TO J
      ASSIGN 4 TO K(1)
      ASSIGN 5 TO K(2)
      READ J, (M(I),A(I), I=1,5)
      DO 10 L=1,2
10    READ K(L), (M4(I), I=1,5)
```

Note that the first statement will be executed according to Format statement 3, while the second READ statement will be executed according to Format Statement 4 when L=1, and according to Format Statement 5 when L=2.

The subscripted variables in all the above examples must previously have been mentioned in a DIMENSION statement.

ARRAY NAMES IN I/O LIST

As in IBM FORTRAN II, array names without subscripts may appear in I/O lists. Mention of an array name will cause the entire array, as specified in the DIMENSION statement to be input or output. Two dimensional arrays are handled column-wise -

```
DIMENSION A(10,10)
READ, A
```

will cause the entire 100 elements of A to be read in, in 5N notation. The elements of A must be in order A(1,1), A(2,1), A(3,1), A(4,1), A(5,1), A(6,1), etc.

FORMAT STATEMENTS

Format statements are, in general, equivalent to Format statements allowed in 7090/94 FORTRAN II. E, F, I and A conversion are permitted. Repetition of field format is allowed before E, F,I or A. Thus FORMAT (I2,3E12.4) is equivalent to

```
FORMAT (I2,E12.4,E12.4,E12.4)
```

Parenthetical expression is permitted in order to enable repetition of data fields according to certain Format specifications within a longer FORMAT statement. The number of repetitions is limited to 99. Thus,

```
FORMAT (2(F10.6,E10.2),I4)
```

The level of parenthesizing can be extended to a second level, thus:

```
FORMAT (2(I4,2(F6.2,F8.3))) is equivalent to

FORMAT (I4,F6.2,F8.3,F6.2,F8.3,I4,F6.2,F8.3,F6.2,F8.3)
```

The depth of such nesting of parentheses must not exceed 5, which appears to be more than would ever be necessary.

N-Format

Rigid format on input data is not always desirable, and in many cases makes key-punching more difficult. KINGSTON FORTRAN allows so-called "free form" input, as well as the more familiar fixed or rigid format. If the FORMAT statement specifies I, E or F format on input, then the input data record must conform to the normal rules for such format as specified in IBM manuals. However, if N format (denoting "free form") is used, the data numbers may appear anywhere on the card, and input is controlled by the input list.

N format is used like E, F or I format except that no width or decimal point location digits are required or permitted. For example,

```
READ 10, I, J, A, C, Z
10 FORMAT (5N)
```

will cause the program to read in a record of 2 integer numbers followed by 3 floating-point numbers. In N format, a number is defined as: any number of leading blanks, followed by a meaningful collection of digits, followed by 1 trailing blank. Note that the blank column immediately following the right-most digit or character of the number is considered part of the number, and serves to delineate the right-hand end of the number.

In the case of E numbers handled with N-format, blanks after the letter E are ignored, and the machine uses the next set of digits as the exponent. For example:

```
b1.2345678E-05b
```

will be interpreted as .000012345678.

The number b1.2345678Ebbbbb-05b

will be interpreted in the same way.

```
b1.2345678Ebbbb103
```

will result in an error condition (see operating instructions).

```
b1.2345678E bb 00005
```

will be interpreted as 123456.78. Leading zeros before either the mantissa or exponent are ignored.

An E- type number handled by N-format ends with the blank after the exponent digits.

A FORMAT statement may specify N, E, F, I or A format as required, thus allowing both free and rigid format on the same card. Note that, in N format, if a floating point number does not have a decimal point, it is assumed to be after the low-order digit of the number.

Some examples may help:

```
    READ 10, I, J, A, C, Z

10  FORMAT (5N)
```

The card might look like:

```
bb123bbbbbb12bbb16.3bbbbb1.2E6b123000bbb etc.
```

N Format requires only that at least 1 blank column follow the number. In this case, I, J, A, C, Z would be stored as 123, 12, 16.3, 1.2E06, 123000. resp.

```
    READ 11, I, J, A, C, Z

11  FORMAT (I3, I6, N, F10.3, N)
```

The Format requires that I, J, C follow rigid format. The card might look like:

```
                   ↓
b12bbb12bbbbbb120.b bbb1234567bbb16.8bbb etc.
```

This would give the following results:

Variable	Value
I	12
J	120
A	120.
C	1234.567
Z	16.8

Note that the F-specification for C starts on the first column after the blank following 120., (see the position of the arrow) since this blank is considered part of the value of an N-Format number.

An output, N format is equivalent to 1PE14.7,1X for floating point numbers, and I5,1X for integer numbers.

N Format allows repeated format and parenthesizing, and follows the usual rules for them.

If a number is positive, the output under E, F, I or N Format will not contain a leading plus sign. On I Format, no space is left for it, so that it is possible to construct a fully packed output record provided all numbers are positive. N Format generates a space for a + sign and a space following the number.

If a floating point number is output under Iw Format, the integer part of the floating point number is convered to Iw Format. Thus 128342.56 output with I10 Format would appear as bbbb128342.

SCALE FACTORS

To permit more general use of E and F conversion, a scale factor followed by the letter P may precede the specification. The scale factor is defined such that

$$\text{Output number} = \text{internal number} \times 10^{\text{scale factor}}$$

$$\text{Internal number} = \text{input number} \times 10^{-\text{scale factor}}$$

This operates exactly the same as in IBM FORTRAN II for the larger machines. For example

```
FORMAT (2PF10.4)
```

used on output will multiply the number by 100 before output. On input, it will divide the external number by 100 before storing it in the machine.

On E-Format output, the effect of P-scaling is to shift the decimal point in the mantissa and to adjust the exponent by the amount of the shift.

Thus, if FORMAT(E15.8), used for output, produced the number .12345678E-04, then FORMAT (3PE15.5) would produce 123.45678E-07 for the same number. Note that for E-Format output, P-scaling does not change the magnitude of the number. It shifts the decimal point, and makes a compensating change in the exponent. For F-Format, P-scaling alters the magnitude of the number on input/output.

VARIABLE FORMAT

KINGSTON FORTRAN II allows variable Format. That is, Format specifications may be read in at object time. In this way, data may be read in under control of a Format Statement which itself has been read in. Variable Format statements must be read under A-Format into an array by means of a normal Read statement.

For example:

```
             DIMENSION FMT (15)

             READ 10, (FMT(I), I=1,14)

        10   FORMAT (15A5)
```

will cause 70 characters of input record (i.e. the Format Statement being read in) to be stored in array FMT. It is then possible to write:

```
             READ FMT, A, B, X, Z, (A(J),J=1,10)
```

where the input variables will be read in according to the Format Statement stored in array FMT.

It is also possible to alter array FMT by programming. This should be done with some care, otherwise the Format Statement stored in array FMT may become completely unintelligible.

The name of the variable Format specification must appear in a DIMENSION Statement, even if the Array size is only 1.

The Format read in at object time must take the same form as a source program Format Statement except that the word Format is omitted, i.e. the variable Format begins with a left parenthesis.

SPECIFICATION STATEMENTS

COMMON

Variables, including array names, appearing in COMMON statements will be assigned core storage locations beginning at the high end of memory, and will be stored at object time in descending sequence, 10 digits per variable, or per item of a dimensioned variable, as they are encountered in the COMMON statement. If a variable is a dimensioned variable, the size of the dimensioned array must appear in the COMMON statement, and the variable must not again be dimensioned in a DIMENSION statement. The COMMON statement must precede EQUIVALENCE or DIMENSION statements (if any) and must precede the first statement of the source program. For example:

```
             COMMON A,B,I,J,X(10,3),Y(5)
```

(Inclusion of dimensioning information in COMMON statements is allowed in FORTRAN IV).

DIMENSION

The DIMENSION statement is the same as IBM FORTRAN II except that variables already mentioned in COMMON may not again be dimensioned and that only 2 subscripts are allowed.

```
DIMENSION Z(10,5),V(400) is permitted

DIMENSION X(10,5,10) is not permitted
```

EQUIVALENCE

```
EQUIVALENCE (a,b,c,---), (d,e,f,--),---
```

where a,b,c,d,e,f, are variable names. KINGSTON FORTRAN imposes some restrictions on EQUIVALENCE statements which are not present in IBM FORTRAN II. These are noted below:

1. Single variables may be equivalenced only to single variables.
2. Arrays may be equivalenced to other arrays, of the same size only.
3. Single variables may not be equivalenced to individual items of arrays, nor may single items of two arrays be equivalenced. In general, no subscripts may appear in an Equivalence statement.
4. Because the compiler is single pass, it is crucial that the order in the source deck be:

   COMMON (if any), DIMENSION(if any), EQUIVALENCE (if any).

   They must precede the first executable statement of the program.

5. If arrays are to be equivalenced, the first item only in the list must have been defined previously in a COMMON, or DIMENSION declaration, and the remaining items in the list must not have been so defined. The Equivalence statement itself defines these remaining items. If single variables are to be equivalenced, and any item in the Equivalence list has been defined in a previous COMMON or TYPE statement, it must be first in the Equivalence list, and the other items must not have been defined in a COMMON or TYPE statement. For example,

   ```
   COMMON A,B(10,3),C
   DIMENSION D(50)
   EQUIVALENCE (A,F,G),(D,X)
   ```

   This puts A, array B, and C in common storage; defines array D; defines F and G as single variables in the same memory location as A; and defines X as a 50-item vector in the same location as D. The following are errors: (in the example above).

```
EQUIVALENCE (D,A)        (para.1,2)
EQUIVALENCE (B(1,1),G)   (para. 3)
EQUIVALENCE (X,D)        (para.5, X not defined)
EQUIVALENCE (G,A,F)      (para.5, G not defined,
                          A defined)
EQUIVALENCE (D(50),X(50)) (para.3)
```

6. To preserve compatibility with other FORTRAN systems, which require DIMENSION statements for all array variables in an Equivalence list, KINGSTON FORTRAN allows extra DIMENSION statements after the Equivalence statements. Such DIMENSION statements may be used to mention the equivalenced variables, but since they have already been defined in the Equivalence Statement, the compiler will ignore them. It will not, however, call them errors. For example:

```
DIMENSION X(10), Y(20)
EQUIVALENCE (X,A,B), (Y,C,G)
DIMENSION A(10), B(10), C(20), G(20)
```

is permitted. The variables A,B,C,G in the second DIMENSION statement are ignored by the compiler, because they have already been defined in the preceding EQUIVALENCE Statement.

7. It is possible to equivalence items not of the same type or mode: e.g. EQUIVALENCE (A,I) - where A is real and I is integer.

TYPE

Two TYPE declarations are permitted. These statements determine the type of variable associated with each variable name appearing in the statement. This TYPE declaration is in effect throughout the program. The two declarations are

```
INTEGER a,b,c,....

REAL a,b,c,....
```

where a,b,c, are variable names appearing within the program. Function names may not appear in TYPE declarations.

Rules:-

(1) A variable defined to be of a given type remains of that type throughout the program.

(2) INTEGER indicates that the variables listed are integer, and over-rides the alphabetic naming convention.

(3) REAL indicates that the variables listed are floating point, and over-rides the alphabetic naming convention.

The TYPE declaration must occur before the first executable statement of the program. If any of the variables mentioned in a TYPE declaration are mentioned in a COMMON or DIMENSION statement, the TYPE declaration must follow such mention.

If a TYPE declaration precedes an EQUIVALENCE statement, then it defines a variable in the sense required by the EQUIVALENCE statement, and all variables equivalenced to the one declared in the TYPE statement will be of the same type.

If a TYPE declaration follows an EQUIVALENCE statement, then only the specific variable names mentioned in the declaration will be affected.

Examples,

```
1.   INTEGER A
     EQUIVALENCE (A,B,C)

2.   EQUIVALENCE (A,B,C)
     INTEGER A

3.   EQUIVALENCE (A,B,C)
     INTEGER A,B,C

4.   INTEGER A,B,C
     EQUIVALENCE (A,B,C)
```

Examples 1 and 3 cause A,B,C, to be integer variables and occupy the same memory location.

Example 2 causes A to be integer, B,C to be real, and A,B,C to occupy the same memory location.

Example 4 is an error in KINGSTON FORTRAN (see para. 5 under EQUIVALENCE).