

IBM DEPARTMENT OF EDUCATION  
Customer Administrative Program - Endicott, New York  
Type 701 Computation Seminar - Class No. 941  
May 3 to 6, 1954

Adams, Charles (Dr.) Massachusetts Institute of Technology-Project High	Boston, Mass.
Amaya, Leland H., Mathematical Engineer Lockheed Aircraft Corporation	Burbank, Calif.
Amdahl, G. M. IBM Corporation	Poughkeepsie, N. Y.
Armer, Paul, Chief, Numerical Analysis Department The Rand Corporation	Santa Monica, Calif.
Backus, John W., IBM Corporation	Poughkeepsie, N. Y.
Beckman, Frank IBM Corporation	New York, N. Y.
Blum, Dorothy T., (Mrs.), Technical Consultant National Security Agency	Washington, D. C.
Carlson, Bengt, Group Leader University of California, Los Alamos Scientific Lab	Los Alamos, N. M.
Chancellor, Justus (III) IBM Corporation	Endicott, N. Y.
Crossman, Stuart L., Head, Machine Computation Lab. United Aircraft Corporation	E. Hartford, Conn.
DeCarlo, Charles R., IBM Corporation	New York, N. Y.
Dobrusky, William B., Computing Engineer Douglas Aircraft Co., Inc.	El Segundo, Calif.
Fagg, Peter, Analyst - 701 Programmer National Security Agency	Washington, D. C.
Fernbach, Sidney, Physicist University of California, Radiation Laboratory	Livermore, Calif.
Furth, Donald H., Computing Analyst Douglas Aircraft Co., Inc.	Santa Monica, Calif.
Griffith, John E., Engineer University of California	Livermore, Calif.
Grosch, Herbert R. J., Manager - Numerical Analysis General Electric Company	Cincinnati, O.
Hart, Donald E., Supervisor, Special Problems Dept. Research Laboratories Div., GMC	Detroit, Mich.
Heising, W. P. IBM Corporation	Washington, D. C.
Horner, John T., Senior Project Engineer General Motors Corporation, Allison Division	Indianapolis, Ind.
Hughes, John B., Unit Engineer General Electric Company	Cincinnati, O.
Johnson, Richard L., Computing Engineer Douglas Aircraft Co., Inc.	El Segundo, Calif.
Johnson, W. H. IBM Corporation	New York, N. Y.

Kogbetliantz, E. G. , Applied Science Department IBM Corporation	New York, N. Y.
<u>Kolsky, Harwood G. , Assoc. Group Leader</u> Los Alamos Scientific Laboratory	Los Alamos, N. M.
Liggett, I. C. , Product Planning IBM Corporation	Poughkeepsie, N. Y.
Lowe, John R. , Chief Computing Engineer Douglas Aircraft Co. , Inc.	Santa Monica, Calif.
Madden, John D. , Associate Mathematician The Rand Corporation	Santa Monica, Calif.
Malone, Fred C. , Assistant Design Specialist The Glenn L. Martin Company	Baltimore, Md.
McClelland, William F. , IBM Corporation	Santa Monica, Calif.
Meyer, Warren E. , Aerophysics Engineer Consolidated Vultee Aircraft Corporation	Fort Worth, Tex.
Oldfield, Bruce G. , Mathematician USNOTS Math Division	China Lake, Calif.
Owen, Frank T. , (Lt.) USN Aviation Supply Office	Philadelphia, Pa.
Pendery, D. W. IBM Corporation	Los Angeles, Calif.
Porter, Randall E. , Senior Group Engineer Boeing Airplane Company	Seattle, Wash.
Ramshaw, Walter A. , Supervisor, Analytical Operations United Aircraft Corporation	E. Hartford, Conn.
Rochester, Nathaniel IBM Corporation	Poughkeepsie, N. Y.
Shuman, Frederick G. , Research Meteorologist Joint Numerical Weather Pred. Unit (Princeton U.)	Princeton, N. J.
Simonet, William P. , Tabulating Analyst Lockheed Aircraft Corporation	Burbank, Calif.
Smagorinsky, Joseph, Meteorologist Numerical Weather Prediction Unit	Washington, D. C.
Stockman, John F. , Associate Research Engineer Boeing Airplane Company	Seattle, Wash.
Strong, Jack A. , Supervisor, IBM Computing Group North American Aviation, Inc.	Los Angeles, Calif.
Sweeney, Dura W. , Assistant Group Leader University of California Los Alamos Scientific Lab.	Los Alamos, N. M.
Tillitt, Harley E. , Head Computing Branch USNOTS Math Division	China Lake, Calif.
Utman, Richard E. (Lt.) Officer in Charge USN Aviation Supply Office	Philadelphia, Pa.
Voorhees, Edward A. , (Jr.) , Staff Member University of California	Los Alamos, N. M.
Wagner, Francis V. , Engineering Computing Coordinator North American Aviation, Inc.	Los Angeles, Calif.
Walters, L. IBM Corporation	Poughkeepsie, N. Y.
Wolanski, Henry S. , Aerophysics Group Engineer Consolidated Vultee Aircraft Corporation	Fort Worth, Tex.

IBM Representatives

- C. C. Hurd - New York
- G. T. Hunter - New York
- J. C. McPherson - New York (Monday)
- G. W. Petrie III - Washington
- W. J. Eckert - Watson Laboratory
- D. Mace - Watson Laboratory
- L. H. Thomas - Watson Laboratory

IBM DEPARTMENT OF EDUCATION  
Customer Administrative School - Class No. 941  
May 3 to 6, 1954

PICTURE IDENTIFICATION

ROW I

1. L. S. Crandall
2. S. Crossman
3. H. R. J. Grosch
4. D. Blum
5. E. G. Kogbetliantz
6. L. H. Amaya
7. R. E. Porter
8. F. C. Malone
9. G. T. Hunter
10. L. Walters

ROW II

1. W. C. Davison
2. W. F. McClelland
3. P. Fagg
4. W. P. Simonet
5. H. S. Wolanski
6. G. M. Amdahl
7. H. Tillitt
8. D. W. Pendery
9. S. Fernbach

ROW III

1. J. Backus
2. D. H. Furth
3. W. B. Dobrusky
4. J. T. Horner
5. D. E. Hart
6. H. G. Kolsky
7. F. G. Shuman
8. J. F. Stockman
9. B. Carlson
10. J. D. Madden
11. I. C. Liggett
12. J. A. Strong

ROW IV

1. J. C. McPherson
2. W. L. Williams
3. B. Oldfield
4. J. Griffith
5. G. W. Petrie
6. W. A. Ramshaw
7. J. B. Hughes
8. J. Chancellor
9. R. L. Johnson
10. D. W. Sweeney
11. P. Armer

\* \* \* \* \*

IBM DEPARTMENT OF EDUCATION  
Customer Administrative Program - Endicott, New York  
Type 701 Computation Seminar - Class No. 941  
May 3 to 6, 1954

Amaya, Leland H. , Mathematical Engineer Lockheed Aircraft Corporation	Burbank, Calif
Amdahl, T. W. IBM Corporation	Poughkeepsie , N. Y.
Armer, Paul, Chief, Numerical Analysis Department The Rand Corporation	Santa Monica, Calif.
Backus, John W. IBM Corporation	Poughkeepsie , N. Y.
Beckman, Frank IBM Corporation	New York, N. Y.
Blum, Dorothy T. , (Mrs.) , Technical Consultant National Security Agency	Washington, D. C.
Carlson, Bengt, Group Leader University of California	Los Alamos, N. M.
Crossman, Stuart United Aircraft Corporation	E. Hartford, Conn.
DeCarlo, Charles R. IBM Corporation	New York, N. Y.
Dobrusky, William B. , Computing Engineer Douglas Aircraft	El Segundo, Calif.
Fagg, Peter, Analyst - 701 Programmer National Security Agency	Washington, D. C.
Fernbach, Sidney, Physicist University of California	Livermore, Calif.
Furth, Donald H. , Computing Analyst Douglas Aircraft	Santa Monica, Calif.
Griffith, John E. , Engineer University of California	Livermore, Calif.
Grosch, Herbert R. J. , Manager - Numerical Analysis General Electric Company	Cincinnati, O.
Hart, Donald E. , Supervisor, Special Problems Department Research Laboratories Division	Detroit, Mich.
Heising, W. P. IBM Corporation	Washington, D. C.
Horner, John T. , Senior Project Engineer General Motors Corporation	Indianapolis, Ind.
Hughes, John B. , Unit Engineer General Electric Company	Cincinnati, O.
Johnson, Richard L. , Computing Engineer Douglas Aircraft	El Segundo, Calif.
Johnson, W. H. IBM Corporation	New York, N. Y.
Kogbetliantz, E. G. , Applied Science Department IBM Corporation	New York, N. Y.

Kolsky, Harwood G. , (Dr. ), Assoc. Group Leader Los Alamos Scientific Laboratory	Los Alamos, N. M.
Lowe, John R. , Chief Computing Engineer Douglas Aircraft	Santa Monica, Calif.
Madden, John D. , Associate Mathematician The Rand Corporation	Santa Monica, Calif.
Malone, Fred C. , Assistant Design Specialist The Glenn L. Martin Company	Baltimore, Md.
Mason, D. R. , Applied Science Department IBM Corporation	New York, N. Y.
McClelland, William F. IBM Corporation	Santa Monica, Calif.
Meyer, Warren E. , Aerophysics Engineer Consolidated Vultee Aircraft Corporation	Fort Worth, Tex.
Oldfield, Bruce G. , Mathematician USNOTS Math Division	China Lake, Calif.
Owen, W. , (Lt. ), USN Aviation Supply Office	Philadelphia, Pa.
Pendery, G. W. IBM Corporation	Los Angeles, Calif.
Porter, Randall E. , Senior Group Engineer Boeing Airplane Company	Seattle, Wash.
Ramshaw, Walter A. United Aircraft Corporation	E. Hartford, Conn.
Rochester, Nathaniel IBM Corporation	Poughkeepsie, N. Y.
Shuman, Frederick G. , Research Meteorologist Joint Numerical Weather Prediction Unit	Princeton, N. J.
Simonet, William P. , Tabulating Analyst Lockheed Aircraft Corporation	Burbank, Calif.
Smagorinsky, Joseph, Meteorologist Numerical Weather Prediction Unit	Washington, D. C.
Stockman, John F. , Associate Research Engineer Boeing Airplane Company	Seattle, Wash.
Strong, Jack A. , Supervisor, IBM Computing Group North American Aviation, Inc.	Los Angeles, Calif.
Sweeney, Dura W. , Assistant Group Leader University of California	Los Alamos, N. M.
Tillitt, Harley E. , Head Computing Branch USNOTS Math Division	China Lake, Calif.
Utman, R. E. (Lt. ) USN Aviation Supply Office	Philadelphia, Pa.
Voorhees, Edward A. , (Jr. ), Staff Member University of California	Los Alamos, N. M.
Wagner, Francis V. , Engineering Computing Coordinator North American Aviation, Inc.	Los Angeles, Calif.
Walters, L. IBM Corporation	Poughkeepsie, N. Y.
Wolanski, Henry S. , Aerophysics Group Engineer Consolidated Vultee Aircraft Corporation	Fort Worth, Tex.

IBM Representatives

C. C. Hurd - New York

G. T. Hunter - New York

J. C. McPherson - New York (Monday)

G. W. Petrie III - Washington

W. J. Eckert - Watson Laboratory

R. Seeber - Watson Laboratory

L. H. Thomas - Watson Laboratory

IBM DEPARTMENT OF EDUCATION  
Customer Administrative Program - Endicott, New York  
Local Government - Class No. 940  
May 3 to 7, 1954

Boyle, Francis X., Analyst City of New York, Fire Department	New York, N. Y.
Buckley, Romanus J., Deputy Revenue Commissioner City of Philadelphia	Philadelphia, Pa.
Collette, Joe G., Chief Accounting Officer City of Winston-Salem	Winston-Salem, N. C.
Coons, Aubrey, Commissioner of Finance City of Poughkeepsie	Poughkeepsie, N. Y.
Cronan, F. L., Controller City of New Haven	New Haven, Conn.
Ermentrout, Donald, Chief Clerk Berks County Commissioners	Reading, Pa.
Hearn, James J., Deputy Revenue Commissioner City of Philadelphia	Philadelphia, Pa.
Hickey, Paul, Assistant Commissioner of Finance The Corporation of the City of Hamilton	Hamilton, Ont.
Horne, Archie J., Assessor, Board of Assessors City of Worcester	Worcester, Mass.
Howe, Raymond, Assistant Budget Dir., County of Ulster	Kingston, N. Y.
Kelly, Francis J., Financial Consultant to City Controller City of New Haven	New Haven, Conn.
McLernon, Joseph F., City Controller City of Bethlehem	Bethlehem, Pa.
Merklein, Carl J., County Auditor County of Onondaga	Syracuse, N. Y.
Myers, Donald A., Assistant Treasurer Prince George's County	Upper Marlboro, Md.
O'Connor, Leo T., Deputy County Treasurer County of Onondaga	Syracuse, N. Y.
Payrow, H. Gordon (Jr.), Councilman City of Bethlehem	Bethlehem, Pa.
Regan, Daniel, Supervisor, Tabulating Division, Treasury Dept. The Corporation of the City of Hamilton	Hamilton, Ont.
Smith, Ray B., First Assistant County Auditor Dallas County Auditor	Dallas, Tex.
Tevi, Aloysius F., Captain, Central Records Division City of Philadelphia, Police, Central Records Div.	Philadelphia, Pa.
Trimmer, Albert J., Inspector, Gen. Records & Communication Div. City of Philadelphia, Police	Philadelphia, Pa.
Zwicker, Lester, Analyst City of New York, Fire Department	New York, N. Y.

(21)

IBM Representatives

G. I. Cargin - Endicott



# IBM CUSTOMER ADMINISTRATIVE SCHOOL

## Announcements

### Identification Badges

It is asked that you wear the identification badge furnished you as it will facilitate getting acquainted with other members of your class. One of the values to be derived from the Customer Administrative School is the association with representatives of other companies with whom you can discuss subjects of mutual interest.

### Class List

Enclosed is a preliminary list of the members of your class. It is suggested that you bring this with you on Monday for use during the opening session.

### Recreation

The enclosed IBM Country Club guest card entitles the holder to use of all recreational facilities at the Country Club without charge.

### Mail

Mail addressed to our guests c/o Customer Administrative School or c/o IBM Homestead, Endicott, will be delivered to the school and distributed at recess and lunch periods.

Mail addressed to the IBM Homestead, Johnson City, will be delivered to the Homestead and placed in the rooms. Outgoing mail at the Homestead should be dropped in slot of the office door.

### Transportation

You will have an opportunity to consult with our transportation representative on Monday concerning return transportation and/or hotel reservations. If you have any government transportation orders or return tickets, it is suggested that you have them with you at that time.

### Laundry, Pressing and Cleaning

Three days are required for laundry service. Bag and slips are in room closet. Please record name and room number on slip. Outgoing laundry should be left in right hand closet in vestibule. Tags for pressing and cleaning are in desk drawer. Outgoing clothing should be left in right hand closet in the vestibule.

### Taxi Service

The following rates prevail:

Homestead to Binghamton	1.50	Use Binghamton taxi
Homestead to Johnson City	1.00	Use Johnson City taxi
Homestead to Endicott	1.50	Use Endicott taxi
Endicott to Binghamton	3.00	Use Endicott taxi
Endicott to Johnson City	2.00	Use Endicott taxi

Long Distance Telephone Calls

Homestead: A coin-operated telephone is located on the lower floor of the Homestead for your convenience. Change may be obtained from the Homestead office. If you use the telephone on the main floor, it is requested that you ask the operator to give you the charge immediately upon completion of the call in order that you may record it on the slip provided for you in the telephone booth and settle with the Homestead office.

School: It is suggested that you ask the secretary in Room 204 to place the call for you, advising her whether you want to pay the charge or have the call placed collect.

---

It is requested that men wear their coats in the dining room and on the main floor of the Homestead.

---

In order that all may enjoy their visit and none be offended, we ask observance of a company rule prohibiting all intoxicants on IBM property.

---

We respectfully ask you to assist us in complying with a New York State Law which makes IBM liable for a misdemeanor if guests engage in gambling at the Homestead.

---

It is earnestly requested that no gratuities be extended to any member of IBM.

---

The Homestead basement is the shelter area for all Homestead properties.

DINNER  
HONORING

CUSTOMER ADMINISTRATIVE CLASS 940  
CUSTOMER ADMINISTRATIVE CLASS 941

IBM HOMESTEAD

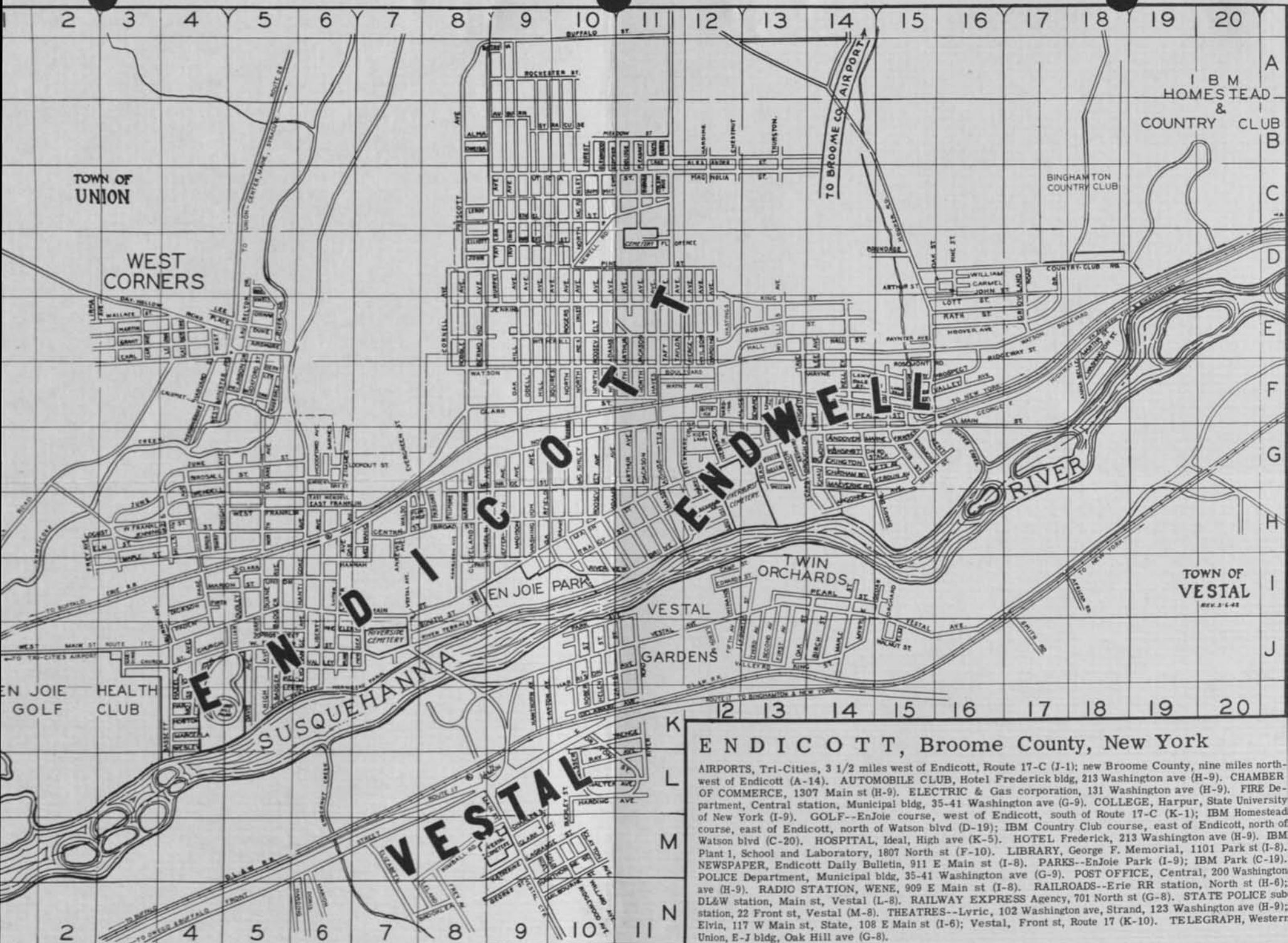
MAY 6 1954

TABLE	NAME	TABLE	NAME
18	MR F V ADAMS	14	MRS C H GREENE
18	MRS F V ADAMS	10	MR R H GREGG
15	MR L H AMAYA	14	MR J E GRIFFITH
2	MR G M AMDAHL		
3	MR R W ARMSTRONG	14	MR D E HART
		5	MR J J HEARN
10	MR J W BACKUS	15	MR W P HEISING
18	MR L G BARNES	3	MR H A HENDRICH
5	MR D E BEEMAN	3	MRS H A HENDRICH
5	MRS D E BEEMAN	8	MR P HICKEY
16	MR K BJORKMAN	6	MR A J HORNE
11	MRS D T BLUM	16	MR J D HOSIE
3	MR F X BOYLE	14	MR N E HOWARD
16	MR R J BUCKLEY	18	MR R HOWE
		8	MR G T HUNTER
11	MR G I CARGIN	8	MRS G T HUNTER
10	MR B CARLSON	11	MR C C HURD
6	MR S L CARR		
1	MR J CHANCELLOR 3RD	3	MR R L JOHNSON
6	MR W M CHAUDOIN		
1	MR J G COLLETTE	17	MR G F KENNARD
17	MR A E COOPER	9	MR E G KOGBETLIANTZ
		5	MR H G KOLSKY
11	MR W C DAVISON	9	MR J J KRAMER
8	MRS W C DAVISON	9	MRS J J KRAMER
15	MR D L DITTBERNER	12	MR E KYLE
5	MR W B DOBRUSKY		
12	MR W C DOUD	11	MR T V LEARSON
12	MRS W C DOUD	4	MR S L LIDA
8	MR R F DUNKIN		
		17	MR F C MALONE
1	MR T A ECK	12	MR E W MARTIN JR
1	MRS T A ECK	2	MR K T MC BRIDE
5	MR W J ECKERT	2	MRS K T MC BRIDE
17	MR D ERMENTROUT	4	MR W F MC CLELLAND
		12	MR C J MERKLEIN
8	MR S FERNBACH	7	MR W E MEYER
9	MR T FITZGERALD	15	MR D A MYERS
4	MR D H FURTH		
		1	MR R H NORD
14	MR C H GREENE		

TABLE	NAME
11	MR L T O CONNOR
9	MR B G OLDFIELD
18	LT F T OWEN
6	MRS U S OWEN
6	MR D W PENDERY
14	MR G W PETRIE 3RD
15	MR J K PLANCK
7	MR R D PUTTY
12	MR W A RAMSHAW
10	MR P A RAUTH JR
17	MR W M REEDY
7	MR D REGAN
17	MR J A RETKA
18	MR D N ROBBINS
18	MRS D N ROBBINS
4	MR S W ROBERTS
4	MRS S W ROBERTS
14	MR L ROBINSON
17	MR N ROCHESTER
10	MR F J SCHWARTZ JR
17	MR A G SHINN
5	MR G D SIMON
15	MR W P SIMONET
16	MR J SMAGORINSKY
4	MR R B SMITH
4	MR L H STINSON
14	MR J F STOCKMAN
5	MR J A STRONG
2	MR W C STYSLINGER JR
3	MISS I M SVENTEK
16	MISS E B SWANK
8	MR E H SWARTOUT

TABLE	NAME
2	MR D W SWEENEY
2	MR N TATUSKO
2	MRS N TATUSKO
1	MR C L TAYLOR
1	MRS C L TAYLOR
9	MR L E TAYLOR
9	MRS L E TAYLOR
7	MR J TEMPLETON
7	MRS J TEMPLETON
7	MR R A TERBOSS
7	MRS R A TERBOSS
16	MR H M THOMAS
16	MRS H M THOMAS
11	MR H E TILLITT
2	MR A F TREVI
11	MR A J TRIMMER
8	LT R E UTMAN
12	MR A A VAYDA
18	MR E A VOORHEES JR
3	MR F V WAGNER
6	MRS F WAY
15	MR J M WICKER
15	MRS J M WICKER
10	MR W L WILLIMAS
10	MRS W L WILLIAMS
6	MR H S WOLANSKI
6	MRS H S WOLANSKI
10	MR L ZWICKER

MAP OF  
E N D I C O T T  
BROOME COUNTY  
N.Y.



### ENDICOTT, Broome County, New York

AIRPORTS, Tri-Cities, 3 1/2 miles west of Endicott, Route 17-C (J-1); new Broome County, nine miles north-west of Endicott (A-14). AUTOMOBILE CLUB, Hotel Frederick bldg, 213 Washington ave (H-9). CHAMBER OF COMMERCE, 1307 Main st (H-9). ELECTRIC & Gas corporation, 131 Washington ave (H-9). FIRE Department, Central station, Municipal bldg, 35-41 Washington ave (G-9). COLLEGE, Harpur, State University of New York (I-9). GOLF--EnJoi course, west of Endicott, south of Route 17-C (K-1); IBM Homestead course, east of Endicott, north of Watson Blvd (D-19); IBM Country Club course, east of Endicott, north of Watson Blvd (C-20). HOSPITAL, Ideal, High ave (K-5). HOTEL Frederick, 213 Washington ave (H-9). IBM Plant 1, School and Laboratory, 1807 North st (F-10). LIBRARY, George F. Memorial, 1101 Park st (I-8). NEWSPAPER, Endicott Daily Bulletin, 911 E Main st (I-8). PARKS--EnJoi Park (I-9); IBM Park (C-19). POLICE Department, Municipal bldg, 35-41 Washington ave (G-9). POST OFFICE, Central, 200 Washington ave (H-9). RADIO STATION, WENE, 909 E Main st (I-8). RAILROADS--Erie RR station, North st (H-6); DL&W station, Main st, Vestal (L-8). RAILWAY EXPRESS Agency, 701 North st (G-8). STATE POLICE sub station, 22 Front st, Vestal (M-8). THEATRES--Lvric, 102 Washington ave, Strand, 123 Washington ave (H-9); Elvin, 117 W Main st, State, 108 E Main st (I-6); Vestal, Front st, Route 17 (K-10). TELEGRAPH, Western Union, E-J bldg, Oak Hill ave (G-8).

3. "Speedcoding at Work" - Ramshaw

Speedco: 3 address floating decimal interpreting program operating at about 1/20 actual machine speed

- Improvements United has made in Speedco,

1. Read order
2. Leave & Return order (Linkage)
3. Transfer OUT to machine language.
4. Write new File Tape X
5. Write ES#2, Read ES#2
6. Leave out DPBC & sign checks.
7. Accumulating Storage addition.
8. Decimal punching 5 nora per cd.
9. Table look-up. (tape search & interpolation)
10. Speedcode "1 1/2", 52 bit precision.
11. assembly program.
12. Training changes --- etc projected. -

7. "An IBM 701 Matrix Abstraction & Its Environment" - Amaya

Main routine accomodates 24<sup>th</sup> or lesser order matrices

- real or complex elements,

60 portions of memory are assigned abstract matrix address.

- double precision floating point (7 bit xp, 27 bit char.)

Command word:    OP    operation  
                   A    addr of [A]  
                   l    no. rows in [A]  
                   m    no. cols. in [A]  
                   B    addr of [B]  
                   n    no. cols in [B]  
                   p

Arithmetic operations [A] + [B]  
 [A] - [B]  
 [A] · [B]

Scalar mult p[B]

Invert [A]<sup>-1</sup>

Form check sums.

checks

150 orders possible,  
200 more on drum)

Logical ops.

Transpose

Store

Store  $\beta$  (arbitrary addr.)

Transfer

Flop (goes to Flop code)

Write EF

Input output

load

Print

Print (same)

Punch punches matrix [A]

Speed. (page 5)

Cost (page 6)

Simple prob. 100 matrix operators, 4.5 hrs machine time,  
(25 man hours)

10. "Theodolite Data Reduction" - Oldfield,

- aekania theodolite, (4 frames/sec)
- Scondlog punch film readers, (0 ident. & 6 quantiles)
- 3 camera least square soln (weighted)

602	6 minutes/point
604	165 sec/point.
CPC	80 sec/point.
701	3.6 sec/point.

Searching Routine: to find particular 2 or 3 cameras for part of trajectory

Camera Station: ident. & 13 constants (can use 10 diff. stations)

Camera Sta. Constants: 3 coords. (21 stations)

Control nos. (which cameras & frames to use)

Computing Routine:

- corrected angles.
- space position.
- residual angles.
- editing & computing - gives velocity & accelerations. (page 6)



- automatic data plotting - (saves 12 to 20 man hrs / test)

⑬ "Numerical Solution of 3 Simultaneous Second-Order Differential Eqs. arising in the Low Energy Neutron Theory of the deuteron" - Kalosky

}  
}

⑱ "Numerical Weather Prediction" Smagorinsky + Heising

(Two variables  $\psi$  &  $\phi$ ) Mesh  $19 \sqrt{19}$  3 deep. (1083 points)  
(half words storage)

2 variables at up to 3 diff time points.

3 time pts  $\times$  2 variables  $\times$  3 lines/variable  $\times$  564 half words/var./level  
= 6552 half words needed (used drums)

3 time steps of  $\psi$  &  $\phi$  (6552 half words) on drums.

~~2~~ 1 time step of  $\psi$  &  $\phi$  (2184 half words) in ES

cyclicly permute drum addresses  $n-1 \rightarrow n-2$  (etc)

Printing: Square array 0.70" by 0.668" <sup>max</sup>  
(present printer  $\frac{1}{6}$  ~~max~~ by  $\frac{1}{7}$  horiz)

1700 instructions needed.

made 3 24 predictions during trial (3 hrs total)

- Error restart from drum data ignoring E.S.

- if drum was wrong read back from tape & reloaded drum.

- page 7 for future plans.

## IBM COMPUTATION SEMINAR

TYPE 701

May 3 - 6, 1954

Monday

9:00 - 10:15

General Opening, Picture, Announcements

10:15 - 12:00

Boeing - "Engineering Computing Considerations" - Porter  
Douglas - "Organization of a Computing Group" - Lowe

1:00 - 3:00

Rand - "Organization of a 701 Installation" - Madden  
United Aircraft - "Speedcoding at Work" - Ramshaw

3:00 - 4:00

Discussion

Tuesday

9:00 - 12:00

Douglas - "Quick and Double Quick Abstraction" - Dobrusky  
Lockheed - "An IBM Type 701 Matrix Abstraction and its  
Environment" - Amaya

North American - "Single Address Assembly Program" - Strong

1:00 - 4:00

Discussion

*Voorhees - Sweeney  
Kogbetliantz*Wednesday

9:00 - 12:00

Inyokern - "Theodolite Data Reduction" - Oldfield  
Inyokern - "Information Searching" - Tillitt  
Convair - "Aerodynamic Calculations" - Meyer  
Los Alamos - "Numerical Solution of Simultaneous  
Differential Equations" - Kolsky

1:00 - 3:00

Los Alamos - "Monte Carlo" - Carlson  
IBM - WHQ - "Diagonalization of Matrices" - Kogbetliantz

3:00 - 4:00

Discussion

Thursday

9:00 - 12:00

Los Alamos - "Coding and Checking" - Voorhees  
NSA - "Sorting on the 701" - Fagg

1:00 - 3:00

General Electric - "Remote Data Transmission" - *Grosch Hughes*  
Numerical Weather - "Weather Calculations" - Smagorinsky  
IBM - "Weather Calculations" - Heising

3:00 - 4:00

Discussion

Friday

Visit 701 at IBM - New York City (Optional for all)

COMPUTATION SEMINAR - TYPE 701

May 20

TABLE OF CONTENTS

\*\*\*\*\*

1.	Douglas - "Organization of a Computing Group"	J. R. Lowe
2.	Boeing - "Engineering Computing Considerations"	R. E. Porter
> 3.	United Aircraft - "Speedcoding at Work"	W. A. Ramshaw
4.	General Electric - "Remote Data Transmission"	<del>H. R. J. Grosch</del> <i>Hughes</i>
5.	Rand - "Organization of a 701 Installation"	J. D. Madden
6.	Douglas - "Quick and Double Quick Abstraction"	W. B. Dobrusky
> 7.	Lockheed - "An IBM Type 701 Matrix Abstraction and its Environment"	L. H. Amaya
8.	Los Alamos - "Coding and Checking"	E. A. Voorhees, Jr.
9.	North American - "Single Address Assembly Program"	J. A. Strong
> 10.	Inyokern - "Theodolite Data Reduction"	B. G. Oldfield
11.	Inyokern - "Information Searching"	H. E. Tillitt
12.	Convair - "Aerodynamic Calculations"	W. E. Meyer
> 13.	Los Alamos - "Numerical Solution of Simultaneous Differential Equations"	H. G. Kolsky
14.	Los Alamos - "Monte Carlo"	B. Carlson
15.	IBM - WHQ - "Diagonalization of Matrices"	E. G. Kogbetliantz
16.	Rand - "Applications of Cathode Ray Tube Read-Out Device for 701"	P. Armer
17.	NSA - "Sorting on the 701"	P. Fagg
> 18.	Numerical Weather - "Weather Calculations" IBM - "Weather Calculations"	J. Smagorinsky W. P. Heising
19.	IBM Poughkeepsie - "701A Machine" IBM Poughkeepsie - "Programming the Type 701A"	G. M. Amdahl J. Backus
20.	IBM Corp. - "EDPM - Type 702"	

April 29, 1954



## THE DOMESTICATION OF A 701

by

John R. Lowe, Douglas Aircraft Company, Inc.  
Santa Monica Division

### INTRODUCTION

The 701 is a computing instrument of enormous power, but efficiently applying this power to the problems of a large engineering department is not simple. Such matters as how much computing time is justified by certain jobs, co-ordination of the computing efforts of the various engineering groups, efficient coding techniques, and smooth flow of information between the engineers and the computing facility must be considered. Perhaps most important of all, engineering methods must be re-examined in the light of this new computing power.

This paper is largely an outline of methods found useful by Douglas, Santa Monica. No pretense is made that these methods are "best". They have been evolving for over two years and are still being improved.

### ORGANIZATION

The steps in solving a problem on a digital computer (and, with some changes of terminology, on any computer) are:

- (1) State the physical problem
- (2) Translate to a mathematical model
- (3) Translate to a numerical form
- (4) Translate to a machine code
- (5) Compute
- (6) Interpret the results in terms of (1).

The Computing Engineering Section operates as a service function for the rest of the engineering department. As such, it assumes full responsibility for steps (3), (4), and (5) and often assists in steps (2) and (6).

The Section is divided into a 701 coding group of twenty-six people, a 701 operating group of three people, an analog group, a standard punched-card equipment group, and a key-punch group. Most of the people in the coding and operating groups have a Bachelor's or Master's degree in mathematics or physics.

The coding group has a leader who schedules and assigns work, and who is responsible for its over-all efficiency. Under him are three co-ordinators. Each job to be coded is assigned to a co-ordinator and to a less experienced man who does most of the actual work. The co-ordinator is responsible for the choice of optimum techniques and for the technical integrity of the job.

#### HANDLING NEW PROBLEMS

An engineer wishing to have computations performed normally contacts the leader of the coding group and explains his problem. A co-ordinator and coder are assigned to the job. Their first efforts are to explore the job to determine its proper scope and decide whether or not existing programs can handle it, wholly or in part. For instance, it may be discovered that Computing should properly do considerably more or considerably less of the total job than originally proposed by the engineer, or that some existing program, with slight modification, can solve the problem. Also, it is decided whether the job can best be done on the 701, standard punched-card equipment, analog computers, or on some combination of the three.

This analysis of a problem often reveals that its mathematical statement can be simplified. Manipulations of arrays of numbers are often best handled as matrix operations. The use of complex numbers may yield a simpler and clearer form than other representations. Another simplification may be to integrate numerically rather than to evaluate the analytic expression for an integral.

A complete and exact statement of the problem is developed on vellum so that all interested persons can have blue-lined copies. From this vellum

statement a computational statement is prepared, showing sequence of calculations, numerical techniques to be used, checks to be made, ranges of variables, input and output. Actual flow charts and coding are prepared from this computational statement.

The coder prepares test cases and checks out the job on the machine. He also prepares a complete write-up of the job, including operating instructions and loading vellums, and turns it over to the operating group for production runs. Thereafter the engineer deals with the operating group.

#### CODING METHODS

The salient feature of the coding system is the facility it provides for handling a problem in logical blocks, or regions. Normally, a master flow chart is prepared from the computational statement, each block representing a region. This process can be iterated, the blocks of the master flow chart being separately charted and each block again becoming a region, etc.

A specification is written for each region to be coded. From these specifications, the actual coding can be prepared and checked out by inexperienced people working independently of each other and requiring no knowledge of the problem as a whole.

These regions operate essentially as subroutines. They are controlled by a master routine which does no computation but contains the logic of the problem. Again, there may be a hierarchy of such master routines.

There are several advantages to this block system. It permits attacking a problem in its details rather than in its dismaying whole. By allowing several people to work on the coding at one time, elapsed time can be reduced. It minimizes the necessity for checking out large blocks of programming. It facilitates changes and permits easy use of all or part of one program in another program.



Three coding systems are being used: actual machine language, a floating point abstraction similar to the Los Alamos "Dual", and a matrix abstraction. Actual machine language accounts for the largest number of jobs and the largest volume of calculation.

Library subroutines are written on magnetic tape in a form such that they can be stored in any portion of memory, the necessary modifications of address parts being accomplished while reading from the tape. This tape is normally on unit #4, being removed only for rare jobs requiring four tape units for data storage. Each time a problem is run, the required subroutines are called from the tape. The library presently contains about 8000 instructions. About five seconds are required to read the entire tape.

It is felt that standardization of coding methods is very important and considerable effort has been expended to insure this. Each coder has a copy of the Coding Manual which contains explanations of standard nomenclature and methods, and complete descriptions of all library subroutines.

Loading vellums are prepared by the coder for most programs. These are used by the engineers to submit data for production runs. Beside each number space, the identification of the quantity in the engineer's language and its storage location in the machine are shown. Positions of decimal points are carefully indicated. At the top of each sheet, the number of the input routine being used and the program number are printed. From these vellums the engineers have blueines made as required. When the data for a particular run have been entered, the sheets go to the operating group and then to key-punch, which can punch and verify cards without further instructions. From key-punch the sheets and cards go back to the operating group which again has all the information necessary for running the job.

Printed output is made easy to read and use. Careful attention is given to the printing format, including decimal points and alphabetic

headings. The form of the output is often a compromise between these considerations and economy of machine time.

#### PARTICULAR PROGRAMS

1. Assembly Program. Electrostatic storage is divided into three sections: instructions, data storage, and temporary storage. Instructions are introduced in the form of symbolic decimal cards in sequence by location. These are packed into the lower section of memory, translated from symbolic to actual form and assembled. The program computes the origins of temporary storage and data storage. Both symbolic and actual instructions are printed and the actual instructions are punched, 46 per card. In addition, the symbolic instructions are punched in binary, 15 per card. The symbolic binary deck can be combined with decimal symbolic changes as input to a re-assembly program for rapid modification of an assembly.

The fact that it is not necessary to specify origins to the assembly program makes it possible to assemble a block of instructions, or region, in various ways. For example, a region can be assembled with a few simple orders to test it, and later with other regions to form a complete program. Also, a region coded for one job can be lifted bodily and used in another job.

It is possible to specify origins for the three sections of storage, and this is done where it is convenient or necessary to have only a part of the total program in electrostatic storage at one time. In this case, each block, while assembled separately, must refer to the same origins of temporary and data storage.

2. Matrix Abstraction. This program operates in the single address mode and provides 32 operations. All numbers are represented in single precision floating point form. Artificial storage units for matrices are established on tapes and drums. A tape may store any number of matrices up to its full capacity, but, of course, the normal rules of tape operation must be observed by the coder. The first word of every



matrix gives its number of rows and columns, this information being read in as part of a data matrix or computed as part of a result. The coding is independent of the sizes and shapes of the matrices, subject to the rules of matrix algebra and a limit of 625 elements in one matrix. The use of check sums is optional. To multiply two 25x25 matrices or to invert a 25x25 matrix takes about one minute.

Considerable care has been taken in this program to preserve accuracy in intermediate calculations. For example, in multiplying two matrices the positive and negative terms of a summation are summed separately, double precision, and then the two halves added and rounded to single precision form.

The matrix abstraction has been found useful for small matrix jobs and for exploratory calculations. However, it is usually considered to be too slow for extensive production use. As the library of matrix subroutines grows and as the coders become more familiar with them, the matrix abstraction is being used less and less.

3. Matrix Subroutines. A number of these are available, many in both fixed and floating point form, so that the programmer can take advantage of the peculiarities of his job. Of particular interest are:

- a) Gauss-Seidel iteration of a quasi-diagonal matrix which minimizes storage requirements. In one case, this routine obtained a solution to a system of 100 equations in about five minutes.
- b) Matrices containing mostly zero elements occur extensively in some types of engineering problems. Here it is economical of storage and time to store only non-zero elements together with their row-column identifications. A number of "addressed element" programs are available to handle these matrices. In one class of these programs the operands are on two tapes and the result is written on a third tape. In this manner very large matrices can be easily manipulated.
- c) A routine which multiplies a rectangular matrix stored on a drum by a vector stored in electrostatic memory, all operations being done between drum copies.



4. Input Routines. A general input subroutine which allows the card to be divided into arbitrary fields of between one and eleven digits, the limit being 72 one-digit fields. The fields are converted as integers and stored in consecutive locations. In all practical cases, sufficient time for scaling and relocation is available between cards.

5. Output Routines.

- a) A roving point print subroutine which prints six 11-decimal digit numbers on a line. The calling sequence of the subroutine specifies the locations of the binary points in the numbers to be printed, and the subroutine determines the location of the decimal point, properly positions it in the card image and prints it in the proper place within the decimal number.
- b) A general print subroutine which is similar to the general input subroutine. It operates on pre-scaled binary numbers stored in consecutive full or half words and forms a card image containing an arbitrary number of fields of from one to eleven decimal digits. Again the limit is 72 one-digit fields. Normally, a printer board which emits the decimal points must be wired for each application of this subroutine.

6. Memory Print-Out Program. This program takes advantage of the fact that all programs begin with standard instructions in memory locations zero through seven. It first dumps the contents of memory on a drum, prints out the contents, and finally restores memory to its original state. Instructions and numbers are printed in their proper form, four full words per line. Any line containing all zeros or all ones is omitted.

7. One Card Programs. Several one card utility programs are available:

- a) A program to read instructions punched in either octal or binary, together with their locations, and store them in the proper locations. This is useful for modifying programs.

- b) A memory to tape or drum program which writes the contents of memory on any tape or drum, preceding it with self-load instructions. If tape or drum one is used, memory can be restored to its original condition by pressing the load button.
- c) A first difference memory print-out program which compares an executed program with the original binary deck and stores zeros in place of those instructions which have not changed. The memory print-out program is then used to print the instructions which have been changed.

8. Plotting Program. Computed ordinates can be punched in binary, 120 per card. After a run through a collator and a reproducer, these summary cards are plotted on a 407 at 150 points per minute. Up to five curves, with different symbols for each curve, can be plotted on one page.

#### FUTURE PROGRESS

Further contributions by computing to the engineering effort depend on three factors: better machines, better computing methods, and engineering techniques which better utilize the new power of computation.

The 701 would be much more useful for processing data if the entry and output of data could be speeded up. As one step in this direction, Douglas is planning to build a device for translating PWM magnetic telemetry tape into 701 tape.

Progress in computing methods seems to lie in the direction of mechanizing the coding. This should reduce errors, speed up programming, and conserve the time of valuable people and machines.

Development of better engineering techniques can take several directions. Problems formerly handled piecemeal now can be more efficiently handled as one job. Since programming is slow and costly, it is advantageous to generalize problems so that one program can handle several physical problems. Problems formerly solved by extensive testing, by making too many simplifying assumptions, or by "educated guessing" should be examined to

see if computing can help. Methods which were so slow that they could be applied only to analysis of final designs can now be used at early stages to help develop the design.

ENGINEERING COMPUTING CONSIDERATIONS

By Randall E. Porter

CALC		REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE
CHECK					
APPD				BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	
APPD					

## ENGINEERING COMPUTING CONSIDERATIONS

### ABSTRACT

The barriers between engineers with problems to solve and mathematicians with large scale computing machines at their disposal are reviewed and suggestions made to increase the teamwork between these groups necessary to solve existing engineering problems. The minimization of restrictive machine program planning rules and of problem solution set up time achieved through the use of a "free machine" is discussed. A method used to keep the benefits of an internally stored utility program library without sacrificing machine freedom is outlined. Specifications for general purpose or utility type programs are stated together with suggested ways of implementing them. The role of the machine assembly program and its assistance to the problem planner is explained. The importance of the program write up in the dissemination of pertinent program information is emphasized, and a typical program write up included. Acknowledgement is made of the assistance rendered by others to the Boeing Engineering Computing Facility in its installation and successful operation of the IBM Model 701 Electronic Data Processing Machine as a powerful new engineering tool.

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE
CHECK						
APP					BOEING AIRPLANE COMPANY	
APP					SEATTLE 14, WASHINGTON	

15 people  
1 1/2 shift

## ENGINEERING COMPUTING CONSIDERATIONS

### I. Computing Considerations

The problems facing engineering organizations today encompass such a variety of physical problems as to make impractical if not impossible the formation of a group of mathematical and engineering specialists capable, by themselves, of reducing the solutions to all problems to forms suitable for numerical solution. The engineer has in the past used his ingenuity, engineering experience, and what might be termed "garden variety" mathematics to attack his everyday problems. Such tools have repeatedly proven to be the means by which engineering problems, impractical to theoretically formulate, have been solved. He has developed the ability to make accurate simplifying assumptions based on experience to account for complex interrelationships existing in the physical system being analyzed. The engineer's present ability and experience cannot be replaced by the mere substitution of complex mathematical equations, even though digital computing machines can now accomplish the solution of such equations in reasonable lengths of time. Instead, his abilities must be augmented by placing at his disposal an engineering tool limited only by his own ability to utilize it; the high speed stored program digital computing machine. A machine of this type, such as the IBM Model 701, with its ability to produce the proper combination of both logical and arithmetic results is capable of becoming a more integral part of engineering analysis than was possible for either the digital or analog computers which preceded it. Whether or not this capability is realized depends to a large extent upon the mutual education of both engineers and mathematicians.

The IBM Model 701 must become an easily used and trusted tool before an engineer can be expected to fully exploit its capabilities in the solution of his problems. The average engineer feels, and not without justification, that if he goes into all the details of his problem a sufficient number of times to fix them in the mind of a machine programmer unfamiliar with the engineering physics and policies involved, more time would be expended than is available for the solution of the problem. He, therefore, only considers using high speed computing assistance when faced with analyses which require a volume of arithmetic computation beyond the scope of manual computation methods. Many an engineer has been initially oversold by the optimistic claims of mathematicians who were still "starry eyed" over the capabilities of their latest acquisition in computing machines, only to be badly

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 1
CHECK						
APFD					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	
APFD						

disillusioned by the elapsed time required for his problem's solution. He overlooks the fact that he carved a piece from the middle of his real problem and passed this piece together with a minimum of supporting explanation along to the mathematician. He then expected this mathematician to come up with a comprehensive set of typical solutions among which he hoped to find part of the solution to his real problem by the exercise of his ingenuity and engineering experience.

The mathematician, on the other hand, tends to make solution method refinement and machine efficiency his primary goals instead of the minimization of the overall solution time required for the problem submitted to him. These are natural impulses arising from academic training in the first case and dictated by necessity in the latter case when using machines of limited storage capacity and cyclic speed. The mathematician familiar with present day engineering computations is also often led by the circumstantial evidence of impractical parameter ranges, incompletely defined problems, poorly arranged initial data, and mathematically inelegant methods of solution to believe that the engineer doesn't know his own problem. Both the engineer and the mathematician need to revise their present concepts regarding the proper place in engineering analyses which can be most efficiently filled by the high speed digital computing machine.

The engineer must learn to translate his present engineering methods into mathematical and logical terms. He must learn to define in a machine program the manner in which he now examines numerical results and makes decisions based on his experience. He must exercise his foresight and provide in advance for contingencies which may arise during the numerical solution of his problem rather than dealing with each eventuality as it arises.

The mathematician with high speed digital computing equipment at his disposal must learn to pool his machine "know how" with the engineer's experience. He must develop methods which use the computing machine itself to handle as many of the clerical details of problems as possible. He must use methods of analysis which are easily understood by the engineer who is, after all, the one held responsible for the validity of the solution.

It is worthy of note that too much publicity of the "giant brain" variety, while excellent from the viewpoint of the lay person who does not have to deal with such "beasts," is actually a deterring factor to the engineer unfamiliar with computing machinery. He often resents the implication that a machine can do any but the most slavish type of arithmetical analysis better than he. He is sometimes fearful of committing his problem to the questionable mercy of a mechanical monster because of the rather lurid results described in the latest

<table border="1"> <tr> <td>CALC</td> <td>Porter</td> <td>3-54</td> <td>REVISED</td> <td>DATE</td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	CALC	Porter	3-54	REVISED	DATE	CHECK					APPD					APPD					<p>ENGINEERING COMPUTING CONSIDERATIONS</p> <hr/> <p>BOEING AIRPLANE COMPANY SEATTLE 14 WASHINGTON</p>	<p>PAGE 2</p>
CALC	Porter	3-54	REVISED	DATE																		
CHECK																						
APPD																						
APPD																						



science fiction magazine. An engineer may subconsciously hesitate to use machine assistance because the incredible computing speed advertised by machine manufacturers might work him out of his job as he visualizes it today. While such resentments or fears are never given voice, they nevertheless increase the barrier between the engineer and his confident use of computing machinery.

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 3
CHECK						
AFPD						
AFPD						
					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	

## II. The "Free Machine" Concept

The combining of the engineer's knowledge of his problem with the mathematician's knowledge of numerical analysis methods becomes largely a one way street when the engineer must learn machine limitations and conventions set up by the mathematician interested solely in promoting machine efficiency. The engineer, whose primary work leaves him little time for learning machine details, especially in the arbitrary convention category, is not at all impressed with machine efficiency but is chiefly concerned with minimizing the total time and effort he must spend to solve his problem. He is by nature interested in the mechanics of computing devices themselves and will gladly make an intelligent effort to understand the physical concepts upon which their design was based. Once this is understood, however, he chafes at having to abide by arbitrarily chosen conventions such as a rule which denies him the use of a certain portion of the machine's memory because someone else has stored some constants there for use at a later date, even though some of them may be useful to him in his own problem. He dislikes being forced into the rut of someone else's thinking on how a computing device with the speed, capacity and versatility of the IBM Model 701 should be used. His educational and working background have trained him to understand complex mechanisms and to adapt them to new uses rather than to accept the original designer's ideas as the only applicable ones. He is not, however, a complete individualist, since no one is more aware than he that the best results are achieved by intelligent compromise among the ideas of many competent individuals engaged in the various facets of a major problem. Hence he is willing to accept and use those ideas which contribute to the solution of his problem without at the same time disrupting his own plan or thought pattern for that solution.

The foregoing considerations coupled with the impracticability of obtaining enough machine specialists who are at the same time masters of the many engineering sciences involved in aircraft design caused the Boeing Engineering Computing Facility to adopt the "free machine" concept for the operation of the IBM Model 701 computer. This concept is, that at the time a solution to any problem is planned for the Model 701, the entire machine is at the disposal of the planner. No portion of memory is reserved. He may use any or all of the tapes or drums as he pleases. He may use general purpose programs written by others or write his own programs. He may operate the Model 701 himself or use the computing facility's operators to run the machine solution at his direction. The IBM Model 701 may in fact be used by him in any way he chooses subject only to the design limitations of the machine itself.

The freedom of planning outlined in the preceding paragraph is not to be regarded as license. Good and bad computing practices are explained

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 4
CHECK						
APPD					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	
APPD						

to the engineer by the computing facility personnel. He is urged to employ general purpose programs whenever possible to minimize problem programming time. He is asked to participate in the selection of and specifications for the types of general purpose programs useful in the numerical analyses he may encounter in his future work once he becomes familiar with the possibilities of the Model 701. He is given the assistance of skilled mathematicians and machine programmers whenever he so requests. They may be employed as full partners in the solution planning or asked only to fill in those portions of the machine programming which require specialized detailed skill such as the timing of input-output components. Only in the case of obvious incompetency would an engineer be denied the right to use the Model 701 as he sees fit.

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 5
CHECK						
APPD						
APPE						
					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	

### III. The Program Library

A library of machine programs, especially those which are general purpose in nature, is required to eliminate duplication of effort, minimize programming time, reduce the specialized machine knowledge needed by an engineer using the machine, decrease time lost because of programming errors, simplify program checkout on the machine and permit machine participation in the assembly of programs to solve each particular problem. The free machine concept precludes setting aside any portion of the IBM Model 701's internal memory for library purposes when computing the solution to a problem. Yet, such an internal library to which the machine itself has rapid access is an advantage which cannot be sacrificed even to provide a free machine. This fact caused the Boeing Engineering Computing Facility to divide machine solution planning into two distinct parts, namely "assembly time" and "computing time." When the machine itself is to be used in the assembly of its own programs, certain portions of the machine's memory capacity are of necessity reserved to hold the general purpose program library, and a few restrictions consistent with the functions of an assembly program are imposed upon the problem planner. These reservations and restrictions do not apply at the time the problem's solution is actually computed by the Model 701. Programs may therefore be assigned to temporary electrostatic storage locations during assembly time while actually being designed to work from other electrostatic locations at computing time.

The time required for random block tape search eliminates the use of magnetic tape as a library storage medium if any faster method can be found. Electrostatic storage is far too small to contain a very comprehensive library and still provide a place in which to assemble the desired program components in practical sized pieces. These facts caused the Boeing facility to choose magnetic drums as the units in which a machine program library can be efficiently stored and rapidly referred to at assembly time. Since a library cannot be left permanently on the magnetic drums in a free machine operation, the library must be kept in a form which permits rapid reloading. This form must also permit rapid library alteration, addition or deletion, and must be safeguarded against unwitting destruction or alteration by a user of the machine. While IBM cards meet some of the above specifications, the loading of four magnetic drums with information recorded on binary cards requires too much time when compared with the input rate of magnetic tape.

A magnetic tape meets all of the library storage and consecutive reloading requirements admirably. The entire contents of four magnetic drums may be recorded on approximately fifty feet of tape. The library tape may be left permanently mounted on a tape unit to save the reel removal and mounting time resulting from the rather cumbersome method employed

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 6
CHECK						
APPD						
APPD						
					BOEING AIRPLANE COMPANY	
					SEATTLE 14, WASHINGTON	

on the IBM Model 701. The library tape may also be safeguarded from accidental erasure by mounting a metal leader on both ends of the library section. Only the deliberate lifting of the metal leader from the guide roller will permit the library tape to be advanced to the reading or writing position. When not in the latter positions the library tape may be left on the take-up reel and the machine user's tape simply attached to the metal leader on the beginning of the library tape. Due to the variety of program "building blocks" required by different engineering branches, separate libraries containing those programs peculiar to units which use the Model 701 most extensively may also be kept in readiness in a similar fashion. This precludes limiting the total number of general purpose programs available from the machine's library by the storage capacity of the four magnetic drums.

General purpose programs written in "regional" form may be assigned to any desired locations at the time they are entered in the machine's electrostatic storage from IBM cards. The card reading time consumed and the necessity of making new regional assignments each time a new location is chosen make it impractical to use general purpose programs in this form after their initial entry in the machine's library. The storage of such programs within the machine library in regional form requires an excessive amount of internal storage since for each instruction the regional location number, the operation code, and the regional address must be stored. This internal storage problem may be minimized by storing each general purpose program less its erasable storage in actual form relative to the locations it occupies in the magnetic drum library. That is, if the program occupied electrostatic storage addresses identical to its magnetic drum library addresses, it would actually operate the Model 701 correctly when control was transferred to it. A "relocation program" may then be used within the Model 701 to move any selected library program from the magnetic drums into electrostatic storage and correct the address parts of its instructions so that the program will work from whatever storage location it may be assigned to by the user.

The address parts of the single address instructions used by the IBM Model 701 fall into two categories, namely those which depend upon their relative location in a program sequence and those which do not. Each of the preceding address categories may be readily identified by referring to its accompanying operation code. For example, the address parts of such operations as ADD, SUBTRACT, TRANSFER, and etc., are normally dependent upon their relative locations in a program while the address parts of such operations as READ, SHIFT, REWIND, and etc., are normally independent of relative location. The Boeing relocation program checks the first "U" half words of a library program on the premise that they are "normal" in character and treats the remaining

CAIC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 7
CHECK						
APFD						
APFD						
					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	

half words as "exceptions." Program or instructional constants are the type of half word usually placed in the exception portion of a library program. Normal instructions have their addresses modified if they are dependent on location and their addresses left unmodified if they are independent of location as they are moved into electrostatic storage. The "exceptions" are not subject to modification and are therefore transferred without alteration.

<table border="1"> <tr> <td>CALC</td> <td>Porter</td> <td>3-54</td> <td>REVISED</td> <td>DATE</td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	CALC	Porter	3-54	REVISED	DATE	CHECK					APPD					APPD					<p>ENGINEERING COMPUTING CONSIDERATIONS</p> <hr/> <p>BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON</p>	<p>PAGE 8</p>
CALC	Porter	3-54	REVISED	DATE																		
CHECK																						
APPD																						
APPD																						

#### IV. The General Purpose Program

The final machine program used to compute the solution of a large scale problem using the IBM Model 701 computer must usually be constructed from a number of "building block" or "general purpose" programs coupled with that portion of the program unique to the particular problem being solved. The man hours required make it impractical if not impossible to write a completely new program for each new problem's solution. The advance preparation and repeated use of general purpose programs is therefore mandatory when a variety of problems must be handled. The basic requirements of a general purpose program are;

- a. That it be designed to permit assignment or reassignment by the Model 701 computer itself to any electrostatic storage location selected by its user.
- b. That after completing its task the program must return machine control to a known location in the main program controlling the solution of the problem.
- c. That it be self checking insofar as it is practical.
- d. That in case of random machine error the program repeat itself and try to obtain a correct result without causing the machine to stop.
- e. That in case repeated machine errors occur, the program stop the machine and indicate the type of failure which causes the check to fail.
- f. That it permit complete control of its operation by the person using it.
- g. That it be easily identifiable in a program library index both as to type and possible usefulness.

First with regard to identification, some systematic yet simple classification method is a distinct aid to a person consulting the index of a library of programs in the hope of finding assistance in the solution of his particular problem. The idea of using the original programmer's initials followed by a sequence number to identify general purpose programs is a worthy one from the standpoint of individual credit but very confusing to a person trying to quickly discover which programs might be of assistance to him. A straight all numeric code number is convenient from a library reference point of view and relatively simple to use once the code is learned. The addition of an alphabetic

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 9
CHECK						
APFD						
APFD						
					BOEING AIRPLANE COMPANY	
					SEATTLE 14, WASHINGTON	

prefix serves to key the person consulting the index as to the general type of the program, thus narrowing the group of program numbers he must actually decode to find the one best suited to his needs; especially if the alphabetic prefix has a phonetic connotation such as A for "arithmetic" program, C for "card" program, and etc. The classification system for general purpose programs in use at Boeing identifies the type of program with an alphabetic prefix followed by a numerically coded number system identification and writing sequence. The writing sequence was included to identify the degree of sophistication the user might reasonably expect from the program since it is to be hoped that this will increase as machine operating and programming experience are obtained. The details of this classification system are as follows:

Programs written for the IBM Model 701 computer are classified at Boeing as a sequence of four alphabetic and numeric characters. The first character defines the purpose of the program. It is always a letter.

1st Letter

Purpose

- A      Arithmetic programs - general purpose.
- C      Card reading or punching programs - general purpose.
- D      Drum read or write programs - general purpose.
- F      Function determination programs - general purpose.
- J      Job programs - special purpose.
- M      Matrix programs - general purpose.
- N      Numerical analysis programs - general purpose.
- P      Printing programs - general purpose.
- S      Statistical analysis programs - general purpose.
- T      Tape read or write programs - general purpose.
- U      Utility programs - general purpose.

<table border="1"> <tr> <td>CALC</td> <td>Porter</td> <td>3-54</td> <td>REVISED</td> <td>DATE</td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APFD</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APFD</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	CALC	Porter	3-54	REVISED	DATE	CHECK					APFD					APFD					<p>ENGINEERING COMPUTING CONSIDERATIONS</p> <hr/> <p>BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON</p>	<p>PAGE 10</p>
CALC	Porter	3-54	REVISED	DATE																		
CHECK																						
APFD																						
APFD																						



The second character defines the number system used in the program for all except the "J" purpose classification. In this latter case the second character is a letter indicating the Boeing Engineering Unit or Company Division which originates the job. In all other cases the second character is a number.

<u>2nd Number</u>	<u>Number System</u>
0	Decimal - stated point.
1	Decimal - floating point.
2	Binary - stated point.
3	Binary - floating point.
4	Unassigned.
5	Unassigned.
6	Combination of number systems.
7	Number system not relevant.
8	Octal - stated point.
9	Octal - floating point.

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 11
CHECK						
APPD						
APPD						
					BOEING AIRPLANE COMPANY SEATTLE 14 WASHINGTON	

For "J" classification only:

<u>2nd Letter</u>	<u>Engineering Unit or Company Division</u>
A	Aerodynamics.
B	Pilotless Aircraft Division.
C	Processes and Standards.
E	Acoustics and Electrical.
F	Flight Test.
G	Armament.
I	Industrial Engineering Division.
J	Project.
M	Mechanical Equipment.
P	Power Plant.
R	Physical Research.
S	Stress.
T	Industrial Products Division.
V	Structural Dynamics.
W	Weights.
X	Preliminary Design.

The third and fourth characters are a two digit sequence number to distinguish programs within the classifications of the first two characters. The first program in a given sequence is designated "00."

All general purpose programs written at Boeing begin with a standard basic linkage entry. An additional entry point may be provided in addition to the basic linkage entry if the program is designed to read a control card from which it will obtain the necessary control data to

CALC. <u>Porter 3-54</u> REVISED    DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 12
CHECK APPRO APPL		

direct the program's operation. Binary control cards are used whenever possible to minimize storage requirements. The control card permits control of the program from outside the electrostatic storage while the basic linkage entry permits internal program control. Immediately following the basic linkage entry, Boeing programs check the condition of the overflow indicator at the time the sub-program was entered. Should the overflow indicator be "ON" at the time of sub-program entry, a "locked in" machine STOP occurs which can only be cleared by manually transferring around the STOP. Such an "OVERFLOW STOP" indicates that the programmer has violated a basic rule by entering a sub-program while an overflow condition set up by his main program still exists. The rule that the overflow indicator must be "OFF" when entering a sub-program was set up to permit legal use of the overflow condition by a general purpose program without disturbing, or being disturbed by, an overflow condition set up by the main program. The Boeing general purpose programs do not use either the sense lights or switches. These are left free for use in the main program where both solution progress and external control are likely to be required.

A standard entry point for all general purpose program control data minimizes the clerical burden imposed upon a person trying to use it. Boeing general purpose programs are written so that the fifth half word location of the sub-program's instruction sequence is assigned to hold the first item of control data, the sixth half word the next item of control data, and etc., until all control data items are entered. This saves the user from the rather grim chore of having to stuff his control data into the program at whatever point the person who wrote the sub-program decided he needed that piece of data. The entry section of a typical Boeing general purpose program is illustrated below:

<u>Location</u>	<u>Operation</u>	<u>Address</u>	<u>Explanation</u>
1000	Store A	1234	Usually used to store point at which main program was left, in "Error Stop" location.
1001	TR OV	1003	Check overflow condition.
1002	TR	1007	Transfer around overflow STOP and constants if overflow is "OFF."
1003	STOP	1003	"Locked in" stop if overflow was "ON."
1004	(1st Control No.)	}	Control Constants.
1005	(2nd Control No.)		
1006	(3rd Control No.)		
1007	Add	1500	Basic Linkage--word 1500 contains 2 in this example.
1008	Store A	1245	Store "exit transfer" address.
1009	-----		Continue with sub-program.

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 13
CHECK						
APPD						
APFD						
					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	

Since a general purpose program would not be used in cases where an unchecked simple program is sufficient for the user's purpose, most Boeing sub-programs provide for at least two tries plus a comparison which must check before machine control is returned to the main program. When possible without the sacrifice of too much electrostatic storage space, an independent check is used. In any case the original data supplied to the sub-program is not altered by the action of the sub-program. This latter provision makes repetition possible in case random machine errors occur. Usually at least one more try at correct completion is made by the sub-program in case an error is detected, before the programmed machine Error STOP is activated, thus minimizing the time lost because of random machine error. Should an error be again detected on the second try, the machine is presumed to be malfunctioning and stopped to permit the initial or control data to be checked for conformity to the sub-program's limitations. The storage of the main program's exit location as the address part of an Error STOP instruction is a help in keying the operator as to which of perhaps several uses of that sub-program is subject to error.

Programs which perform the transfer of blocks of data from one unit of the Model 701 to another require a check sum against which the check sum calculated from the transferred data may be compared. It is confusing, and therefore one more source of error, for the problem planner to have to remember to add two to the half word count of his actual data every time he transfers it from one place to another. It is particularly annoying to be forced to leave room in electrostatic storage at the end of each block of data for a check sum, especially if the data blocks represent instruction sequences which must work as a complete unit at computing time. The Boeing general purpose programs therefore use the "invisible" check sum approach, that is, check sums being read from cards, magnetic tapes or magnetic drums are kept in the erasable storage of the transfer program and not affixed as the last full word of the data block itself in electrostatic storage. This permits blocks of data or instructions to be assembled in electrostatic storage without leaving space for check sums between blocks. When data are to be transferred from electrostatic storage to cards, magnetic tapes or magnetic drums, the check sum of the entire block of data to be transferred is calculated and stored in the erasable storage of the transfer program. The data are transferred, and then the calculated check sum is written as the last full word of the transferred record. Only when writing on the magnetic drums does the programmer have to plan for storage space for the check sum. Card check sums are punched in specified cards or locations on a card and are therefore fixed in location so the programmer does not have to consider them.

<table border="1"> <tr> <td>CALC</td> <td>Porter</td> <td>3-54</td> <td>REVISED</td> <td>DATE</td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	CALC	Porter	3-54	REVISED	DATE	CHECK					APPD					APPD					<p>ENGINEERING COMPUTING CONSIDERATIONS</p>	
CALC	Porter	3-54	REVISED	DATE																		
CHECK																						
APPD																						
APPD																						
		<p>BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON</p>	<p>PAGE 14</p>																			

## V. The Assembly Program

The ability of the IBM Model 701 itself to combine general purpose programs from its internal library with other instructions read from any of its forms of input media enables it to materially assist in the assembly of the complete set of machine instructions required to solve large problems. Since the total number of instructions for a large problem may not all fit into the available electrostatic storage space at assembly time, provision must also be made in an assembly program for recording logical portions of the complete program on a suitable output media such as IBM cards or IBM magnetic tape for future re-entry in the Model 701 at computing time. An assembly program must in addition be able to provide the problem planner with partial listings of the assembled instructions to permit him to monitor the work done by the machine under his direction. The basic assembly program functions stated above, those of assembling, recording and providing a trail of the work done, must be accomplished by the machine with a minimum of clerical detail required of the problem planner to be of maximum assistance to him while at the same time providing him with complete machine control at all times.

Since the majority of the engineers using the IBM Model 701 at Boeing are not familiar with either the octal or binary number systems, Boeing assembly programs are controlled by IBM cards punched in the decimal number system. The first five digit field in each decimal control card is interrogated by the assembly program to determine which of its functions to perform. The succeeding five digit fields in the card are punched with the minimum control data needed by the assembly program to activate the general purpose program actually performing the desired function. When the assembly program senses the code to bring a library program into electrostatic storage for example, it knows that the next control card field will contain the library number of the desired program, the following field the electrostatic storage location in which the first word of the library program is to be placed at assembly time and the field after that the electrostatic storage location in which the first word of the library program will be at computing time. These two latter locations may or may not be the same. Using the program library number as a guide, the assembly program then searches its library index to find the magnetic drum location, number of half words and the number of instructions affected by relocation for the general purpose program desired by the problem planner. These control data together with the specified electrostatic storage locations are inserted in a relocation program by the assembly program which then transfers machine control to the relocation program via basic linkage. After the library program is successfully installed in electrostatic storage, the relocation program automatically returns machine control to the assembly program

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 15
CHECK						
APFD						
APFD						
					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	

which then reads another control card to find out what to do next.  
 Typical functions performed by Boeing assembly programs are:

- a. Punch a binary card self loading program and its associated control card to perform automatic re-entry of assembled programs recorded in binary card form.
- b. Write a tape self load program as the first unit record of a tape selected for subsequent recording of assembled programs.
- c. Read regional instruction cards and convert them into an actual instruction sequence in specified electrostatic storage locations.
- d. Bring a general purpose program from the machine library and relocate it to work from specified electrostatic storage locations.
- e. Read a sequence of instructions from previously recorded binary cards and place them in electrostatic storage.
- f. Read a sequence of instructions from previously recorded IBM magnetic tape and place them in electrostatic storage.
- g. Read a program comment and place it in electrostatic storage as a card image beginning at a specified location.
- h. Record a specified number of consecutive electrostatic storage location contents in binary punched card form.
- i. Record a specified number of consecutive electrostatic storage location contents as the next unit record on an IBM magnetic tape.
- j. Print a specified number of consecutive electrostatic storage location contents as machine instructions in decimal and octal form.
- k. Transfer machine control from the assembly program to the program just assembled.

<table border="1"> <tr> <td>CALC</td> <td>Porter</td> <td>3-51</td> <td>REVISED</td> <td>DATE</td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	CALC	Porter	3-51	REVISED	DATE	CHECK					APPD					APPD					<p>ENGINEERING COMPUTING CONSIDERATIONS</p> <hr/> <p>BOEING AIRPLANE COMPANY          SEATTLE 14, WASHINGTON</p>	<p>PAGE          16</p>
CALC	Porter	3-51	REVISED	DATE																		
CHECK																						
APPD																						
APPD																						

## VI. The Program Write Up

A general purpose program write up disseminates pertinent information used by a program planner in the selection, control, and usage of machine library programs. Unless these write ups follow a systematic outline and are edited for consistency, a rather heterogeneous collection of program information results. It is particularly important that this not be allowed to happen when engineering personnel not intimately familiar with all the details of machine operation are trying to discover which general purpose programs will assist in the solution of their problems. It is also important that these personnel not be overwhelmed by a mass of detail, such as the list of actual machine instructions, at the time they consult the library of program write ups. The Boeing Engineering Computing Facility has adopted the policy of placing the general purpose write ups exclusive of the program block diagram and list of instructions in one document and the program details in another. The document containing the write ups is given wide circulation throughout the engineering department. The detail document is restricted in circulation principally to the mathematical services group itself where it may be consulted by any engineer actually interested in programming techniques. A typical Boeing general purpose program write up is given on the following pages and is self explanatory.

CALC	Porter	3-54	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 17
CHECK						
APPD						
AND						
					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	

I. PURPOSE

This program reads V half words into consecutive locations in electrostatic storage from a drum or writes V half words on a drum from consecutive locations in electrostatic storage.

II. METHOD

A. Drum to electrostatic storage

After preparing the drum for reading, V half words are copied into electrostatic storage. A check sum is then copied from the drum into the erasable storage of this program and used to check the data read from the drum.

B. Electrostatic storage to drum

Prior to beginning writing, a check sum of the V half words is formed and stored in the erasable storage of this program. After preparing the drum for writing, V half words are copied onto the drum from electrostatic storage. The check sum is then copied into the next full word location on the drum.

C. Program Limitations

Data can be successfully transferred by this program provided the starting location plus the number of half words does not exceed  $10(4096) = 9(8000)$ , i.e.,

$$M + |V| \leq 4096$$

where: M is the first word address in E. S.  
and provided that no part of this program lies between the first and last locations of data, i.e.,

$$M + |V| \leq t \text{ or } 10(t+84) \leq M$$

where t is the starting location of this program. Note that V does not include the check sum.

D. Checking

The full word check sum used by this program for checking is the standard check sum and is equal to the sums of the absolute values plus  $2^{17}$  times the number of negative half words of data to be transferred. In writing on the drum, this check sum is formed and written in the full word drum location immediately following the last word of data transferred. In reading from the drum a second check sum is formed and compared with the check sum copied from the drum. They must agree exactly for the transfer to check.

CA: D.I. Cook 11-53 REVISED: DATE

ENGINEERING COMPUTING CONSIDERATIONS

BOEING AIRPLANE COMPANY  
SEATTLE 14, WASHINGTON

PAGE 18



### III. USAGE

#### A. Program Entry

<u>TYPE</u>	<u>LOCATION</u>	<u>OPERATION</u>	<u>ADDRESS</u>		<u>REMARKS</u>
			<u>Dec.</u>	<u>Octal</u>	
Basic Linkage	r	R ADD	r	r	r = location in main program.
"	r+1	TR	t	t	t = first half word location of this program.
"	r+2 . . . . .				control returned here.
Binary Control Card	r	TR	t+29	t+35	when entered at this location the program will read a control card.
Self Load	(Does not apply)				

#### B. Control Data - Basic Linkage Entry

<u>ITEM</u>	<u>LOCATION</u>		<u>EXPLANATION</u>
	<u>Decimal</u>	<u>Octal</u>	
M	t+4	t+4	The first positive even location in E.S. of data to be transferred.
±V	t+5	t+5	The even number of half words to be transferred exclusive of the check sum. If V is negative, the transfer is from drum to E.S. If V is positive, the transfer is from E.S. to drum. Note that $0 \leq M +  V  \leq 4096$ .
N	t+6	t+6	The positive drum number where $128 \leq N \leq 131$ .
B	t+7	t+7	The first positive even drum address of data to be transferred.

CALC	D.I. Cook	11-53	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 19
CHECK						
APPD						
APPD						
BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON						

III. USAGE (cont'd)

E. (cont'd)

Control Data - Binary Control Card Entry

<u>ITEM</u>	<u>FORM</u>	<u>CARD COLUMNS</u>	<u>CARD ROW</u>	<u>EXPLANATION</u>
M	STOP M	1-18	9	The first positive even location in E.S. of data to be transferred.
±V	±STOP V	19-36	9	The even number of half words to be transferred exclusive of the check sum. If V is negative, the transfer is from drum to E.S. If V is positive, the transfer is from E.S. to drum. Note that $0 \leq M +  V  \leq 4096$ .
N	STOP N	45-62	9	The positive drum number where $128 \leq N \leq 131$ .
B	STOP B	63-80	9	The first positive even drum address of data to be transferred.
S	STOP* S	1-18	8	The location in E.S. to which control is to be given after successful completion of the data transfer.

	<u>LEFT</u>			<u>RIGHT</u>	
Row 8	STOP* S				
Row 9	STOP M	±STOP V		STOP N	STOP B
Card Cols.	1-18	19-36	37-44	45-62	63-80

\*This instruction can be replaced by "TR S" if a successful stop is not desired.

Transfer to location  $(t+29) = (t+35)$  of this program to cause it to read the above binary control card.

CARC	D.I. Cook	11-53	REVISED	DATE	ENGINEERING COMPUTING CONSIDERATIONS	PAGE 20
CHECK						
APPD					BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON	
APPD						

III. USAGE (cont'd)

C. Input Data

1. This program can transfer any number of half words of data which can be contained in electrostatic storage exclusive of the locations occupied by this program. The number of half words to be transferred (V) must be at least 2.
2. The form of the input data is irrelevant.
3. Data to be read from a drum must have a check sum in the full word drum location immediately following the last word of data, i.e., in drum location B + |V|.

D. Auxiliary Programs Required (None)

E. Program Storage Requirements

<u>NO.</u>	<u>ITEM</u>	<u>STARTING LOCATION</u>		<u>NO. OF HALF WORDS</u>	
		<u>Decimal</u>	<u>Octal</u>	<u>Decimal</u>	<u>Octal</u>
1	Program including erasable storage	t	t	84	124
2	Program without erasable storage	t	t	76	114
3	Instructions affected by Relocation	t	t	72	110
4	Program Constants	t+72	t+110	4	4
5	Erasable Storage	t+76	t+114	8	10

Program Constants

<u>DECIMAL</u>			<u>OCTAL</u>		
<u>Loc.</u>	<u>Op.</u>	<u>Address</u>	<u>Loc.</u>	<u>Op.</u>	<u>Address</u>
t+72	00	0000	t+110	00	0000
t+73	00	0001	t+111	00	0001
t+74	00	0002	t+112	00	0002
t+75	00	0003	t+113	00	0003

F. Regional Constants

<u>REGIONAL CONSTANT</u>	<u>NO. OF HALF WORDS IN REGION</u>	
	<u>Decimal</u>	<u>Octal</u>
F0000	72	110
B0000	4	4
E0000	8	10

CAIC D.I. Cook 11-53 REVISED DATE

CHECK

APPD

APPD

ENGINEERING COMPUTING CONSIDERATIONS

BOEING AIRPLANE COMPANY  
SEATTLE 14, WASHINGTON

PAGE

21

III. USAGE (cont'd)

G. Program Stops

LOCATION  
Decimal    Octal

MEANING

CORRECTIVE ACTION

STOP	t+3	t+3	Overflow on at time of entry to program.	Programming must be corrected before proceeding.
STOP	t+32	t+40	End of file stop. No control card.	Put control card in reader and "ready." Pressing start key causes control to return to location $_{10}(t+28) = \rho(t+34)$ .
STOP	t+38	t+46	Successful stop for control card.	Pressing start key causes control to be given to location "S" punched in control card.
STOP	t+61	t+75	Too many words transferred.	Pressing start key causes program to try again. If stop is repeated, determine cause of machine failure.
STOP	t+70	t+106	Too many attempts required to transfer data.	Pressing start key will cause control to return to location R of basic linkage to try again. If stop is repeated, determine cause of machine failure.
TR	t+66	t+102	Successful exit.	$W_{10}(t+66)$ is "TR r+2."

CALC	D.I. Cook	11-53	REVISED	DATE
CHECK				
APFD				
APFD				

ENGINEERING COMPUTING CONSIDERATIONS

BOEING AIRPLANE COMPANY  
SEATTLE 14, WASHINGTON

PAGE 22

VII. Acknowledgements

The installation and successful operation of the IBM Model 701 Electronic Data Processing Machine by the Boeing Engineering Computing Facility as a powerful new engineering tool was made possible by information and assistance from several sources. The machine operating characteristics described so well by the IBM engineers from the Poughkeepsie laboratory gave us an excellent start toward understanding the equipment from the design standpoint. The discussions of programming techniques and the preparation of utility programs by the IBM Applied Science Department personnel equipped us to begin successful operation of the machine with a minimum of delay and confusion. Although neither the New York nor Poughkeepsie types of IBM utility programs were suitable for the type of operation assigned to the Boeing computing facility, both of these forms were invaluable as "bootstraps" to set up the system now in use before Boeing programs were operable. The IBM Speedcoding system is being used at Boeing for problems suitable for solution with abstract coding. Finally, the week spent by a number of Boeing personnel on the Model 701 installed at IBM World Headquarters in New York straightened out our thinking and prepared us for the immediate use of our own machine.

Information received from the Los Alamos Scientific Laboratory was also greatly appreciated by the Boeing facility. We expect to place the "Dual" system in operation for any engineer requesting it and have derived a great deal of benefit from the T-1 Utility Program Manual. Although our method of operation is slightly different, we are encouraged by the success at Los Alamos achieved by "outsiders" programming their own problems.

Lastly, we desire to express our appreciation to the IBM Customer Engineering Department for their excellent performance which has resulted in such satisfactory operation of the Model 701 installed at Boeing Seattle plant. Without in any way minimizing the initial contributions of all others, the continuing effort required of IBM service engineers to keep a machine as complex as the IBM Model 701 in good operating condition is the largest single factor in its success, for without trouble free operation the best planning personnel are helpless.

<table border="1"> <tr> <td>CALC</td> <td>Porter</td> <td>3-54</td> <td>REVISED</td> <td>DATE</td> </tr> <tr> <td>CHECK</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APPD</td> <td></td> <td></td> <td></td> <td></td> </tr> </table>	CALC	Porter	3-54	REVISED	DATE	CHECK					APPD					APPD					<p>ENGINEERING COMPUTING CONSIDERATIONS</p> <hr/> <p>BOEING AIRPLANE COMPANY SEATTLE 14, WASHINGTON</p>	<p>PAGE 23</p>
CALC	Porter	3-54	REVISED	DATE																		
CHECK																						
APPD																						
APPD																						

## SpeedCo at Work

The United Aircraft Corporation Computation Laboratory has been operating an IBM 701 since early in October of 1953. Practically all of the work done on the machine during this period was performed through the medium of the Speedcoding System, a multiple address floating point interpretive routine prepared and distributed by the IBM Scientific Computing Service in New York City.

SpeedCo operates upon numbers which are expressed, inside the machine, in floating binary. The fractional part (or mantissa) of each of these numbers occupies one full 701 word. The corresponding exponent occupies the left half of the next full 701 word. The right half of this word is not used.

Primary input to, and final results from the calculator are expressed in floating decimal. Each piece of input data consists of a ten digit decimal fraction accompanied by a three digit exponent. All results printed by the calculator are also expressed in this form. Card reader input and printer output are both at the rate of 750 pieces of data per minute. Conversion between binary and decimal notation and between powers-of-two and powers-of-ten is completely automatic.

SpeedCo instructions each occupy two full 701 words. Each instruction provides for two distinct operations. First, a three address arithmetic, trigonometric or exponential operation may be performed upon the floating point numbers handled by SpeedCo. (This three address portion of the SpeedCo instruction is also used to control blockwise transfers of information within electrostatic

*over 100 people use it  
at United*

and between electrostatic and tapes, drums, or the printer). Second, an entirely unrelated single address operation may be included in order to provide conditional or unconditional transfer of control, fixed point modification of addresses of the program, or control of certain built-in checking features. These compound two operation four address instructions are interpreted and executed at the rate of about 250 per second, which is roughly one-twentieth of the rate at which machine language solutions proceed.

Our almost exclusive concentration on such a relatively slow coding system would appear to indicate that we feel that we can be rather prodigal of machine time. In a sense this is true. We are much more interested in minimizing the total time a problem spends in the Computation Laboratory than in minimizing the time it spends on the 701. And for the general range of our problems the coding and debugging time represents by far the biggest portion of the total elapsed time. Hence considerations of overall efficiency of our operation require that we adopt coding procedures which are easy to use and which result in coded sequences which are easy to debug. It has been our experience that SpeedCo satisfies these requirements.

Our use of SpeedCo during this six month period has, however, suggested the desirability of certain changes and modifications. Some of these we have already made; others are still in the planning stage. These changes are described below. (No special importance attaches to the order in which these changes are described. The presentation is in fact, more or less chronological).

#### 1. Additional operation READ

SpeedCo, as originally written, assumed that all data and instructions for a problem were loaded into the calculator and stored in electrostatic or on tapes or drums before the calculation began. Under these circumstances there

was no need for a SpeedCo instruction which would cause the card reader to resume operation (under control of the program) after some calculations had been performed. Hence SpeedCo provided no such instruction.

There are, however, numbers of problems at United Aircraft which can be much more efficiently handled if the programmer is not restricted to loading all of the input information for the entire problem before the calculation begins. In many of these problems, for example, a much better mode of operation is as follows. The instructions and only as much data as is needed for the first case are read into the calculator and stored. The first case is calculated and the results are printed. Then, under program control, the card reader resumes operation and the data for the second case is read in. The second set of results is then calculated and printed. The card reader then starts up again and reads in the third set of data. And so on.

In order to facilitate the solution of such problems the operation READ was added to SpeedCo. Whenever this instruction is executed, the card reader initiates the reading of the next card in the hopper. If this card is (as it should be) the self-loading SpeedCo 99 99 card the cards following will be loaded and stored in the usual fashion.

## 2. Additional operations LEAVE and RETURN

In order to facilitate the use of SpeedCo language subroutines, the operations LEAVE and RETURN were added to SpeedCo.

Whenever LEAVE is executed the current contents of the SpeedCo program counter are stored in a special electrostatic register reserved for this purpose and control is then unconditionally transferred to the location given as the D address associated with LEAVE.



Whenever RETURN is executed control is unconditionally transferred to the SpeedCo instruction located in that memory cell whose address exceeds by one the contents of the special register referred to above.

### 3. Additional operation OUT

Most of the problems encountered at United Aircraft are of such nature that the "balance" provided in SpeedCo (no more than one single address logical operation for each three address arithmetic operation) is quite satisfactory. There are a very few problems, however, where the number of single address logical operations required greatly exceeds the number of arithmetic operations to be performed. Typical of these is the problem of solving a set of linear simultaneous equations. Here the number of arithmetic operations is very small compared to the number of logical operations. Hence a direct SpeedCo solution would be quite inefficient, since most of the instructions would consist of single address logical operations bracketted with time-wasting NO OPERATION three address arithmetic operations.

In order to permit more efficient solution of this sort of problem the operation OUT has been added to SpeedCo. Whenever this instruction is executed control is transferred to that address which is four times the D address used with OUT. At this location the programmer will have stored a machine language routine which will carry out the necessary logical operations at full machine language speed. These logical operations having been completed, this machine language program will then transfer control back to the SpeedCo interpretive loop in order to permit resumption of floating point arithmetic calculations.

### 4. Additional operations WRITE NEW FILE TAPE X

Some of the problems encountered at United Aircraft can be more easily coded if one can write more than one file of information on a single tape.

It has been discovered that this can be done successfully if the following procedure is used. After writing the last record of a file, do not write the end of file gap. Instead, the tape must be left undisturbed until the first record of the next file is to be written. At this time the end of file gap should be written, followed immediately by the writing of the first record of the next file. The timing here is important. The tape must not be permitted to stop after writing the end of file gap and before writing the first record of the next file. If it is permitted to stop, the machine will not be able to read the tape because of noise pulses at the end of the end of file gap.

The operations WRITE NEW FILE TAPE X have been written with this in mind. Whenever one of these instructions is encountered an end of file gap is written on the designated tape and the first record of the next file is written. Since only one machine language instruction (a transfer) intervenes between the machine language instructions "Write End of File" and "Write Tape", the tape has no time to stop and hence the machine has no difficulty in reading the tape thus written.

##### 5. Additional operations WRITE and READ ELECTROSTATIC TWO

The United Aircraft 701 was equipped with a second electrostatic memory frame during February of this year. It has been integrated into SpeedCom merely as a very fast form of auxiliary storage. The two instructions involved are WRITE ELECTROSTATIC TWO and READ ELECTROSTATIC TWO.

Whenever WRITE ELECTROSTATIC TWO is executed a block of information is transferred from E.S.F. 1 to E.S.F. 2. The given block extends from A to B in E.S.F. 1 and the relocated block extends from C to C+B-A in E.S.F. 2.

Whenever READ ELECTROSTATIC TWO is executed a block of information is transferred from E.S.F. 2 to E.S.F. 1. The given block extends from C to C+B-A in E.S.F. 2 and the relocated block extends from A to B in E.S.F. 1.

It is interesting to note that problems which made extensive use of drum storage have benefitted to a surprising extent by recoding them to place the most frequently consulted information in E.S.F. 2. In one case this cut the running time for a moderate size problem in half; in another the running time for a rather long problem was cut by one-third.

#### 6. DPBC and sign checks on data cards

SpeedCo was originally so written as to check data cards for double punches and blank columns. In addition, it also checked that every fractional part and exponent read into the machine was identified either by a twelve punch for plus or an eleven punch for minus.

It does not appear to us that the restrictions inherent in these checks are justified by the advantages obtained. Operating experience with the United Aircraft machine seems to indicate that errors of the sort caught by these checks are quite infrequent. We have therefore changed SpeedCo to permit the loading of data cards on which

- (a) blank columns represent zeros.
- (b) x punches represent minus signs.
- (c) no x's represent plus signs.

#### 7. Accumulating storage

SpeedCo was originally so written that the result computed by any one of the arithmetic, trigonometric, or exponential operations was sent to storage where it replaced the previous contents of the register C. In many cases, however, it would clearly be preferable to have this result add itself to the previous contents of C. We have changed SpeedCo in order to make this possible.

In order to permit the programmer to exercise his choice as to whether the result is to replace the previous contents of C or whether it is to be added to the previous contents of C, a quantity K has been included in each SpeedCo instruction. K, which is punched in column 14 of each instruction card, can have either of the two values zero or one. If K is zero, the result replaces the previous contents of C. If K is one, the result is added to the previous contents of C.

Programs which make frequent use of the operation ADD benefit in two ways from this change. First (and most important) such programs are shortened considerably, since many of the additions required need not be separately programmed but may be obtained merely by giving K for the preceding instruction the value one. Second, there is also a timewise advantage since the interpretation time for this "accumulating storage" addition is very much less than the interpretation time for the regularly coded operation ADD. In the case of the combination multiply and add, for instance, a time saving of just under one-fifth is achieved.

#### 8. Decimal punching

A decimal punching subroutine has been added to SpeedCo. Five floating point numbers are punched per card. Punching proceeds at the rate of 100 cards per minute.

#### 9. Additional operation TABLE LOOK UP

Problems arising at United Aircraft frequently require the representation, within the machine, of empirical bivariate functions for which no analytic expressions exist. Usually we replace such functions by bivariate polynomials, making use of standard orthogonal polynomial approximation techniques. Occasionally, however, it turns out that the given function cannot be satisfactorily approximated in this fashion. In such cases we resort to tabulating the function using

quadratic interpolation in both directions. The resultant tables are then written on tape with each table comprising one unit record. A tape searching and interpolation routine called TABLE LOOK UP has been added to our SpeedCo, and this routine is used in connection with this tape. The A address gives the location of  $x$ , the B address gives the location of  $y$  and the C address gives the table number. When the instruction TABLE LOOK UP is encountered the machine first searches the tape for the correct table (unit record). It then searches this unit record for the correct section of the table. The interpolation coefficients for this section of the table are then read into electrostatic and the interpolation is performed.

#### 10. SpeedCo "One-And-One-Half"

In a very few cases problems have been encountered at United Aircraft which have required greater precision than is afforded by the 35 bit fractional part provided by SpeedCo. In order to take care of these unusual problems, we have written a special SpeedCo which provides 52 bit precision and which is called "SpeedCo One-And-One-Half". In this SpeedCo the data word occupies two full 701 words and is arranged as follows. The first full word contains the most significant 35 bits of the fractional part. The left half of the second full word contains (as before) the exponent. The right half of the second full word (which was previously unused) now contains the least significant 17 bits of the complete 52 bit fractional part. The arithmetic operations have been rewritten to provide 52 bit precision. The additional electrostatic space needed for these more precise subroutines has been obtained by deleting the trigonometric and exponential operations. In all other respects SpeedCo One-And-One-Half is the same as the United Aircraft SpeedCo. In particular, the decimal input and output remain at the same ten digit level of precision. The read in conversion process automatically gives the value zero to the least significant 17

bits of each 52 bit fractional part. The print routine ignores these 17 bits entirely. Throughout the entire calculation, however, these 17 bits are of course correctly handled by the arithmetic operations available in this special SpeedCo.

#### 11. Assembly

A SpeedCo assembly program is presently being prepared. This program follows the general pattern of IBM Assembly Program S 02. It will provide in the case of SpeedCo problems, essentially the same features that S 02 provides for machine language problems. Regional programming will be used. The output of this program will usually be by decimal listing and binary punching, with eleven SpeedCo instructions punched on each binary card.

#### 12. Projected changes

In order to realize more fully the advantages inherent in the doubled electrostatic memory capacity, it is obviously desirable to rewrite SpeedCo so that it uses the entire electrostatic as working memory instead of (as at present) using half as working memory and half as fast auxiliary storage. This is of course a very far-reaching change since the interpretive loop and all subroutines which in any way use or operate upon SpeedCo addresses must be rewritten to conform to whatever new address numbering system is adopted for the larger working memory. This project is now under way at United Aircraft. In the new system, as in the old, each instruction and each data word will occupy two full 701 words. The first of these words will be located in a 36 bit cell in E.S.F. 1 and the second will be located in that 36 bit cell in E.S.F. 2 whose address exceeds by one the address of the E.S.F. 1 cell containing the first word.

Since the modifications entailed in making this change are so extensive as to require the complete rewriting of large sections of SpeedCo, it has been decided that certain other desirable changes may as well be made at the same time. These changes are as follows.

(A) Tracing

1. Change the interpretive loop so that tracing will begin only after the instruction in location F has been executed n times. In all other respects tracing will be controllable by sense switches, as at present.
2. Make such changes to the tracing routine itself as may be needed to at least double the speed of tracing. Among these changes are the following.
  - (a) Decrease the quantity of information printed to such extent that only one type stroke per instruction is necessary.
  - (b) Delete the extensive system of checks, so that the routine itself will no longer encroach upon programmer storage.
  - (c) In order to retain reasonable printer efficiency, continue the present system of executing ten instructions, printing the tracing information for these ten instructions, executing the next ten instructions, printing their tracing information, etc. In order to speed this process up, however, do not use drum storage to accumulate the ten line block of tracing information to be printed. Instead, store this information in electrostatic.

(B) Checking

During the execution of each SpeedCo instruction approximately one-third of a millisecond is devoted to set up time for the built in checking routine,

even when this routine is not being used. But experience with the United Aircraft 701 shows that the machine makes very, very few calculation errors of the sort detected by the checking routines built into SpeedCo. Hence it appears advisable to change this feature so that checking is under sense switch control. This will have the advantage that only the occasional program which requires SpeedCo checking will be burdened with any significant amount of checking set up time.

(C) Tracing during checking

Change the tracing routine so that tracing during the second pass through a checking loop is under sense switch control.

(D) Starting point control

SpeedCo is so written that the calculation always begins with the execution of the instruction stored in location 300. This is to be changed in order to permit the programmer to specify any register he chooses as the starting point of the calculation.

(E) Tape and drum identifications

SpeedCo is so written that it is not possible to operate upon an instruction such as WRITE TAPE M in order to change it to, say, WRITE TAPE K. However, instructions of this sort would be very helpful in certain problems which transfer information back and forth between tapes and such instructions will therefore be added.

(F) READ BACKWARD TAPE X

These instructions will be deleted, since they have never been used.

(G)  $R_A$ ,  $R_B$  and  $R_C$  Counters

SpeedCo provides three fixed point counters which are useful for a variety of purposes such as stepping addresses, counting the number of times some loop has been traversed, etc. The contents of these counters are usually controlled by means of TRANSFER AND INCREASE XYZ and TRANSFER AND DECREASE XYZ instructions. Whenever one of these operations is



executed the contents of the designated counter or counters are increased or decreased by one and control is unconditionally transferred to the D address given in the same instruction. Programming experience with SpeedCo indicates that these counters would be much more useful if some convenient means existed for incrementing their contents by amounts other than one. In order to accomplish this it has been decided to eliminate the transfer function from these instructions and make them merely counting instructions, with the D address being used to specify the desired increment. A skip feature will be incorporated into the DECREASE operations which will cause the SpeedCo program to skip one instruction when the contents of the designated R counter are decreased to zero. This will permit the coding of two-instruction loops which are terminated by means of a count-down-to-zero-and-skip procedure.

(H) Error skips

SpeedCo signals the occurrence of certain errors such as tape check sum discrepancies by executing a program skip of one or two instructions. This arrangement is going to be reversed so that the skip occurs if the operation succeeds and does not occur if the operation fails.

(J) Data card forms

Two data card forms are to be incorporated into the rewritten system. The one form will be the same as the present form with 5 ten digit fractional parts and 5 exponents per card. The other, for use with jobs where a large volume of data of low precision has to be read in, will provide for 10 five digit fractional parts and 5 exponents per card. Each exponent will apply to both of the five digit fractional parts to its left.

(K) LEAVE and RETURN

These instructions presently provide for convenient transfers only between the main program and its subroutines. It has been decided that this

same principle should be extended in cascade fashion at least enough to also permit similar transfers between subroutines of the main program and subroutines of these subroutines.

(L) Numbering unit records on tape

It has been decided that the C address of the tape reading and writing instructions will be used to identify tape unit records. During the execution of WRITE TAPE the C address will be written on the tape at the beginning of the record. During the execution of READ TAPE a tape search will take place which will locate and read the desired unit record. It would appear that this can be accomplished without incurring any significant time penalty.

TELETYPE COMMUNICATIONS:  
ITS APPLICATIONS IN A COMPUTING SERVICE

BY: JOHN B. HUGHES

April 26, 1954

The Numerical Analysis group at General Electric in Evendale, Ohio is presently engaged in the operation of an IBM 701 calculator. Relatively early in our operating history we found that a considerable portion of our problems were originating at remote points. Examples of these problems are test data reduction problem where the test facilities are located in Lynn, Massachusetts and design problems originating in the Steam Turbine Division of General Electric in Lynn and Schenectady.

Due to the urgency of the solution of these problems, a rapid communication system between these remote points and the Evendale 701 was necessary. In order to establish this communication a teletype network was installed. This network presently is made up of four stations connected to a single circuit. These stations are:

1) Evendale, 2) Schenectady, 3) and two stations in Lynn. These stations time share the wire facility with practically all traffic being to and from Evendale.

The circuit facility and terminal machinery are standard Western Union rental equipment. All stations are equipped with a model 15 sending-receiving teleprinter, a model 39 typing reperforator, a distributor transmitter, and selector equipment for the control of remote stations. This equipment allows the transmission of either taped data or data sent via a keyboard. The receipt of data is either or both a printer copy and a punched paper tape. In order that the data may be converted to and from punched cards automatically, the IBM type 043 and 063 machines are used. The 043 will read standard teletype five hole tape and punch cards. The coding and card form are under control of a plugboard. The 063 performs the inverse operation of reading cards and producing teletype tape.

The two different types of problems, i.e., data reduction and design problems, impose somewhat different requirement upon the data transmission. In the case of design problems the amount of input data is relatively small but an error of a single digit in the transmission may produce disastrous results in the running of the problem. Furthermore, since the design problem may well fall into an area of investigation for which the designer has no intuitive feeling of what the answer should be, errors in the

results may go unnoticed. Data reduction on the other hand usually implies large volumes of data with somewhat less stringent requirements on accuracy. Generally the test data is taken for a multitude of similar operating points. If an error occurs it is detected relatively easily by plotting, examining trends, etc. For the above reasons two separate and distinct methods of data transmission and checking are used for the two different types of problems.

The design problems and analytical studies being transmitted to us from the Steam Turbine Division in Lynn and Schenectady are processed through the 701 as shown in Figure 1. The data is initially keypunched from a source document originated by the engineer specifying the problem. The cards are then check summed on a 407 and the computed check sum is placed at the end of the deck of cards to be transmitted. The 407 also produces a copy of the input data so that a neat permanent record of data is available at the sending station. The cards are then processed to tape using the 063. Control information is automatically punched into the tape so that the teletype tape produced will cause the teleprinters to print formats comparable with the 407. The tape is then transmitted by means of the tape reader. While the tape is being transmitted the teleprinter monitors the transmission. This gives the operator a visual check of the progress of the outgoing information. At the receiving station the data is again punched onto tape and interpreted on the tape. The teleprinter also monitors the transmission. The tape is then processed through the 043 which produces a deck of cards. If all is well, this deck of cards is a duplicate of the original at the remote point. The cards are then checked by recomputing the check sum and comparing this with the original. As a by-product of this check summing a copy of the input data is produced by the 407. Given on this copy of the input data is a program number which specifies the deck of binary program cards to be used for the calculation and refers the operator to a set of operating instructions for the 701. Normally the output of the 701 is decimal cards. These cards go through a similar process to return the results to the problem originator. In the event of an

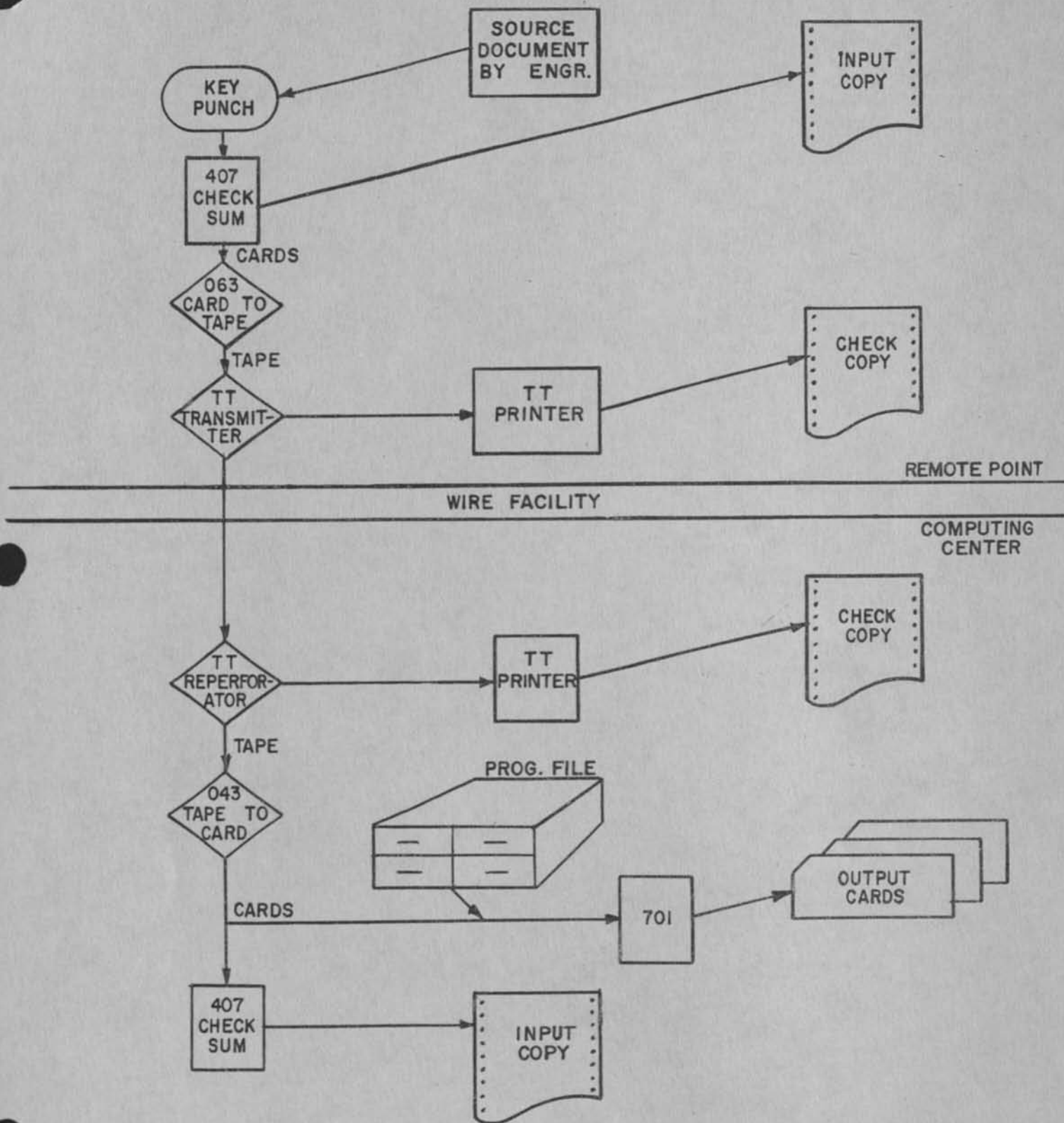


FIG. 1. PROBLEM FLOW DIAGRAM

error the receiving operator checks through the input data to see if there are any erroneous characters or cards obviously in error. The teletype copy is then checked to determine if the error was in transmission or conversion. If it occurred in conversion, the error can be fixed by reprocessing the tape or keypunching the error card. If the mistake is not easily found then the deck containing the check sum error is retransmitted. The Keyboard facilities are used by the operators to give verbal instruction for reruns etc.

The processing of data reduction problems follows the same flow diagram for the conversion and transmission. The recording of data is originally done in the test cell as the test progresses. The data is recorded on standardized log sheets which have been designed to fit given instrumentation layouts and are not the most logical for transmission or programming. In order to simplify the recording of data onto cards the operator merely punches the log sheet data and appropriate identification numbers. The identification numbers are fixed for any given test but the amount of data taken may vary. The cards are then transmitted to Evendale. As the cards are made they are checked for proper format and for erroneous characters. The cards are then checked on a collator for proper identification numbers. The identification numbers are later converted to addresses by means of a table look-up in the 701. When the cards are read into the 701, parts of the data are checked to see if they are within specified ranges and if sufficient data has been transmitted to perform the calculation specified. If all tests are passed the 701 computes and prints the results with appropriate alphabetic headings. In some cases the results are punched on decimal cards so that graphs can be produced on automatic plotting equipment.

One additional field of work which we are embarking upon is the processing of test data which has been automatically recorded. Our test facility in Lynn has installed an automatic recording analog to digital converter. This device was built by the A. D. Little Co. It has the capacity for reading 50 channels of input and recording

the information by means of a Flexowriter. The Flexowriter also produces paper tape which can be transmitted for processing on the 701. This system places additional requirements on the transmission of the data. The volume of data produced by such machinery may be tremendous and, therefore, require maximum speed and efficiency in communication with the data reduction center.

Several economic factors must be considered in the installation of a teletype system. The saving of time is the major advantage to be gained. In the case of test data reduction we are dealing with test facilities which have operating expenses in the order of \$500 per hour. It is imperative that the results of these tests be available at the earliest possible time so that the progress of the test may be determined. The calculation which we are doing would require a prohibitive amount of hand computing at the test site and mail service would probably introduce a two day delay. Our total rental for the wire service and terminal equipment is about \$1500 per month for the four stations. This includes about 750 miles of wire facilities. At present the cost of transmitting decimal digits over this wire facility is the same order of magnitude as reading them into the 701. Then costwise, we are supplying these remote customers with a hypothetical 701 which has half speed input and output for data. We feel that there is a class of problems which are economical on this hypothetical machine. Of course the economics will vary with the length of line, type of problem, and alternate methods of solving the particular problems.

In the future we hope to see improved, cheaper, and more reliable methods of transmitting data. It would be a considerable advantage to have the remote customer in more direct communication with the machine. At present it does not appear practical to tie a machine such as the 701 directly to telegraphic facilities. Greater bandwidths and special equipment would be necessary. The IBM 065 card to card machine will allow us a more direct communication since there will be no intermediate medium such as tape and the transmission is self checking.



In conclusion I would like to acknowledge the effort of Mr. R. A. Butterworth of the Steam Turbine Division in planning the problem transmission system described.

A NOTE CONCERNING THE ORGANIZATION OF  
AN IBM TYPE 701 INSTALLATION

John D. Madden

P-505

7 April 1954

---

The **RAND** Corporation

1700 MAIN ST. • SANTA MONICA • CALIFORNIA

---

Summary

This is a brief report on three projects in the RAND Numerical Analysis Department. The first is a survey of the organizational setups used at various IBM Type 701 installations. The results give an insight into the philosophies influencing operation of each of eleven computing groups. The second is the development and validation of a test which will help to distinguish good prospects for computing jobs from bad ones. The last is the installation of a camera in the 701 room to expedite checking of machine programs. It is felt that the camera will allow divorcing the programmers from machine operation and that this will effect an increased efficiency in machine utilization.

A NOTE CONCERNING THE ORGANIZATION OF AN  
IBM TYPE 701 INSTALLATION

This is a report on three projects in the RAND Numerical Analysis Department — one virtually completed, another in process, and the last about to begin. The Organization Survey section is completely consistent with the title of the paper whereas the Test Project and Camera Project sections bear only indirectly on the organization of an installation.

Organization Survey

Early this year we decided to survey computing groups that use IBM Type 701 machines. We were interested in ascertaining how each was set up organizationally. Accordingly, questionnaires\* were sent to twelve companies that had 701's installed and "on rental." The participants are listed below.

Boeing Airplane Company  
Douglas Aircraft Company, Inc. (El Segundo)  
Douglas Aircraft Company, Inc. (Santa Monica)  
General Electric Company  
General Motors Corporation  
International Business Machines Corporation  
U. S. Naval Ordnance Test Station (Inyokern,  
China Lake, California)  
Lockheed Aircraft Corporation  
Los Alamos Scientific Laboratory  
North American Aviation, Inc.  
The RAND Corporation  
United Aircraft Corporation

We believe that there are benefits for all in the

---

\*A copy of the questionnaire appears at the end of the paper.

unrestricted sharing of information of this kind. This provides the basic motivation for our undertaking the survey. Admittedly, our attitude is that of an organization which is not engaged in competition with like organizations.

Specifically, we were interested in collecting data on three points. We realized that our programming and coding group was under-staffed. This was evidenced by the fact that these people were frantically working to process the jobs and keep the machine busy on useful work while devoting too little time to the important job of preparing and revising general routines. We wondered if this situation obtained at other installations. Also, we were interested in comparing our proportions of the people in the various job classifications with the norm. We felt that we might gain by seeing these figures. Also, we wanted to ascertain if other installations were hiring people with specialist classifications which we had not considered.

The first five questions on the questionnaire are intended to frame the mode of operation of the reporting group. A compilation of the answers to these questions follows.

1. How many hours per day is your 701's power on?

36, 20, 17, 16, 16, 11, 11, 10, 10, 9, 9

mean: 15

This requires little comment although the "thirty-six" figure is obviously a two-machine answer.

2. What percentage of your machine's production time\* is logged on problems coded and programmed by members of your Computing Department?

100, 100, 100, 100, 80, -80, 80, 75, 65, 20, 0

mean: 73

It is interesting to note that installations tend to have a completely "closed shop" or a completely "open shop" and the larger number of groups prefer Computing Department programming.

3. What percentage of your machine's production time is logged on problems which use abstract code and interpretive routines?

100, 95, 90, 90, 80, 70, 30, 25, 25, 15, 0

mean: 56

Here, there appears the same tendency to go to one end of the scale or the other. However, in this case, installations are about evenly divided between abstract code emphasis and de-emphasis.

4. What percentage of your machine's production time is logged on problems of a one-shot nature (as distinguished from problems which recur week after week or month after month)?

100, 90, 80, 70, 20, 20, 20, 10, 5, 0

mean: 42

\*Production time is used in the sense of good machine time excluding assembly and code-check.  
29-1013-0(R)

Here the same tendency appears although one would suspect that some groups are answering one question and some groups another. This question was intended to determine the percentage of an installation's machine time used on codes which are run, filed away, and forgotten. However, we feel that some of the small answers reflect a different interpretation of the question.

5. How much production time is logged on an average-length problem?

25.0, 10.0, 8.0, 2.5, 2.0, 1.0, .8, .3, .2, .2  
mean: 5.0

Here, again, we suspect that these are not all answering the same question. In this case we were attempting to determine "the number of hours logged on completed jobs" divided by "the number of completed jobs." The small ones look too small. This is a difficult question to answer for many groups since they have very few "completed" jobs.

Entries in the table which constitutes item 6 on the questionnaire show that total personnel numbers range from fewer than fifteen to greater than fifty with a mean of thirty-three. In order to obtain the numbers of people in various job classifications, it was necessary to make these classifications broader since many persons occupy positions which fit in two or more of the suggested categories. Consequently, 701 operators and dispatchers are considered as one group; punched-card machine operators including keypunch operators comprise a second group; and

all programmers, coders and numerical analysts make up the third group. The mean proportion ratios for these three groups are approximately 1:2:7. It is curious to note that this approximates closely the ratios selected by many groups.

The following results pertain to the specific points we wanted to investigate.

1. Our "programming, coding, and numerical analysis" staff is slightly larger than the average. We do not feel, however, that the fact that we are understaffed implies that groups with fewer people are also understaffed. There is too much variance in the manpower demands of different modes of operation of a computing department.
2. Our proportions of people in the various job classifications are quite close to the norm.
3. The specialist classifications (different from those suggested on the form) were "analog operators" in one group and people engaged in "business applications" at another installation.

#### Test Project

For some years we have been concerned about how we are to select good prospects for our department out of a group of applicants. It is difficult to single out good people from a group of recent graduates particularly when they have no work experience. Unfortunately this is the category of more than half



of those who apply. Judgment has been based on whether or not the man "talks" a good interview and on his college grades. Neither of these is a certain criterion. In fact, it is not clear that we want the people with the best marks.

We have experienced the situation in which two persons with virtually identical backgrounds are hired, are given similar opportunities, and one proves to be considerably more valuable than the other. Also, we have cases of persons who were almost rejected who have become useful additions to the department. One man, in particular, had lower than usual grades and "talked" an average interview. He has since developed into a leader in our group. It is discouraging to contemplate the number of good men who have gotten away because we failed to recognize their worth.

In the interest of getting a better indication of the aptitude of persons before they are hired or rejected, we asked psychologists in the RAND Social Science Division to aid us in obtaining a test which would help discriminate between the likely and unlikely applicants. Dr. Robert L. Chapman consented to undertake this.

We started the project by looking over many tests and trying to predict whether or not each test measured attributes which were desirable or undesirable to us. We were interested in tests which would give a negative correlation with what we considered desirable as well as those which would show a positive correlation. A group of tests was selected and administered to all of the people in our department and to people in the Inyokern and Douglas, Santa Monica installations. This gave us a sample

of about 100. The test scores were correlated with ratings of the subjects made by supervisory people in each installation.

The final, finished results of the study are not complete. However, it appears that three "factors" are important. They are called "reasoning," "space," and "stability." Somewhat simplified definitions of these qualities follow.

- "reasoning" - the ability to solve logical problems  
— to foresee and plan.
- "space" - the ability to think about objects  
in two or three dimensions.
- "stability" - the ability to remain calm in a crisis,  
to disregard distractions while working,  
to avoid becoming irritated if interrupted  
when concentrating.

In order to determine the amount of each of these attributes which an applicant has, we have been using two tests — the Thurstone Primary Mental Abilities test and the Thurstone Temperament Schedule test. Evidence as to the usefulness of the tests is, of necessity, pretty much of a "testimonial" nature. All persons who were hired after testing well, have worked out well. We did hire two men who had low "stability" scores. Both of these men had records of frequent job changes and both have since left us. It is not too difficult to extrapolate "chronic dissatisfaction with job situations" from a lack of stability qualities.

It should be emphasized that we do not use the test as an absolute determiner. We believe that it should be considered no more than an indication of the potential of the individual. Apparently, some people enjoy taking tests whereas

others are upset by a testing situation and fail to do as well as they should. However, subjects do not score better than they are capable of, hence, high scores on the tests indicate an abundance of the quality being measured.

One attribute which the tests do not sense is "drive." We have not found a way to select persons with this characteristic and we feel that it is important. By "drive" we mean an attitude which will not allow the individual to drop a problem until he has completely solved it. We have the case of two men both of whom had almost identically high test scores — one is good and the other is very good. This is due to a tremendous amount of "drive" in the latter man. We are still searching for a way to determine this characteristic.

#### CAMERA PROJECT

We are building a camera for installation in our 701 room to photograph the machine's console panel. We expect that this will be a help in code-checking and an aid toward trying out a pair of theories concerned with the efficient operation of a computing department.

For some time we have felt that a degree of job specialization in a computing installation is desirable. In our punched card work, a programmer writes a detailed outline of the work that must be done and submits it to the machine room where operators process the cards in accordance with the written instructions. We think that this system has advantages over one in which an individual does all of the work on one job. It assures our having experienced people working with the machinery; it allows us to process jobs on an around-the-clock basis; and it tends to force

the programmer to think "real hard" about the job before it gets to the machine rather than while it is on the machine or after a difficulty has appeared.

We theorize that the advantages of this system realized in punched-card work are also obtainable in 701 work. Further, it would appear that the saving is increased due to the relatively larger machine rental of the 701. Most of our 701 "production" work (as distinguished from code-checking and assembly) is done by 701 operators from "procedures" written by the programmers. Some of the code-checking is done in this manner.

The other theory which we would like to test is the following: We believe that much code debugging may be done by merely observing the 701 console panel when the machine stops rather than indiscriminately resorting to such props as memory print-outs (MPO's) and tracing. Also, we believe that the average cost of discovering a coding error will be smaller if the programmer first tries to find his difficulty using clues available on the console panel and then goes to other schemes where necessary.

We have gone through at least two stages in developing a code-checking technique. At the beginning we used tracing to a great extent. This was in large part due to habits developed over a period of years in checking CPC codes and, before that, in checking-out 604 and 605 control panels. When we determined the relative merits of tracing and MPO's and compared these with their relative costs, we placed emphasis on MPO's. Now, we are tending toward sparing use of MPO's and greater use of the "stop information" only.

In order to divorce the programmers from operation of the machine and at the same time to encourage a test of our code-check theory, we are installing the camera. Whenever an unpredicted stop occurs, either during a production run or during code-check, a photograph of the console panel will be taken, time-stamped, and returned to the programmer along with the rest of his materials. The programmer will try to determine the reason for the stop using the panel clues and take appropriate action. In some cases this will involve requesting the operator to run to the same place and get an MPO.

It is true that the operator could read and record the information on the panel instead of photographing it. And, he might, in this way, be able to record the pertinent information faster, or at least cheaper, than we can with the camera. However, if the contents of A, MQ, and/or Memory Register are desired, it is clear that the picture is more efficient. Further, the photograph will make it more difficult to overlook any information available on the panel. For instance, the picture will indicate whether or not the operator has set the Sense Switches correctly.

The camera will be mounted on the wall across the room from the console. The pictures will require a time exposure of about one second. For this interval, enough lights in the room will be turned out (automatically) so as to avoid losing the console lights and at the same time to record the switch settings. A time exposure is required because the orthochromatic Polaroid film is relatively insensitive to the red light of the neons.

This camera idea is not new with us. For some time the people at MIT have been using a camera to photograph the Whirlwind I control panels. Their purpose is somewhat different from ours, however. The device is used as an aid in tracking down intermittent machine errors. The pictures give them a permanent record of the condition of each toggle and of each control switch setting at the time an error occurs. We, too, shall have the information for this purpose but our primary aim is to expedite code-checking.

7 open  
9 closed  
3 mixed

701 Facility Questionnaire

1. How many hours per day is your 701's power on? \_\_\_\_\_
2. What percentage of your machine's \*production time is logged on problems coded and programmed by members of your Computing Department? \_\_\_\_\_
3. What percentage of your machine's production time is logged on problems which use abstract code and interpretive routines? \_\_\_\_\_
4. What percentage of your machine's production time is logged on problems of a one-shot nature (as distinguished from problems which recur week after week or month after month)? \_\_\_\_\_
5. How much production time is logged on an average--length problem? \_\_\_\_\_
6. How many of your Computing Department people fit, primarily, in each of the slots in the table below? Add appropriate categories if they do not already appear.

cover  
GE  
NSA

	<u>Supervisory</u>	<u>Non-Supervisory</u>
701 operators	_____	_____
701 dispatchers	_____	_____
Punched-card machine operators (not keypunch)	_____	_____
Keypunch operators	_____	_____
701 programmers and coders	_____	_____
Punched-card programmers and coders	_____	_____
Numerical Analysts	_____	_____
Administrators	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

\*"Production time" is used in the sense of good machine time excluding assembly and code-check.

QUICK AND DOUBLE QUICK:  
FLOATING POINT ABSTRACTIONS

Donald W. Gantner  
Computing Engineer  
Douglas Aircraft Company, Inc.  
El Segundo Division

QUICK and DOUBLE QUICK are single address, floating quaternary point abstractions, employing single precision and double precision arithmetic respectively. The companion abstractions utilize identical codes and hence a problem, once programmed, may be run in either system at the discretion of the analyst.

Originally, QUICK was conceived to fill the need for a 'quick and dirty' programming system. The last part of the phrase was dropped when development and refinement produced an abstract system which was obviously useful for other than one-shot jobs. The system may still be described as QUICK, since refinements were not allowed to submerge the original requirements which follow:

1. That the system be straightforward so as to preclude virtually all programming errors.
2. That it be simple, requiring a minimum of training and experience as a prerequisite to successful programming.
3. Most important, and as a result of the first two requirements, that the elapsed time between receipt of the problem and the delivery of correct solutions be minimized.

An abstract system was clearly needed, for despite the admirable flexibility of the machine operations, fixed



binary arithmetic can be considered neither simple nor straightforward to those educated in the decimal system. Furthermore, if the abstract instruction is tailored to the specific application, a single abstract instruction can initiate execution of a series of instructions correctly, thus reducing programming and programming errors.

The requirements immediately dictated a floating point system with its inherent relief from scaling, shifting, record keeping and assorted apprehensions. Quaternary point was indicated since it affords faster floating than binary point, and yields more significance for any given word size. Finally, the basic single address system was chosen as best satisfying the requirements of simplicity and straightforwardness.

DOUBLE QUICK was developed when it was discovered that QUICK was sufficiently fast to warrant its use over a large range of problems. DOUBLE QUICK serves two purposes. Primarily, it is a double precision system, yielding twice the accuracy obtained from QUICK. In addition, it most adequately stifles the cries of 'deception' frequently heard from the fixed point proponents, and occasionally from the customer. They claim, and rightly so, that one is lulled into a false sense of security by the appearance of eight or more digits in the final result. By the simple expedient of running a representative case or cases in both abstractions, fears are allayed or substantiated by observing the significant digit agreement. Should the fears be substantiated, one has the option of running the whole problem in DOUBLE QUICK. Since it is considerably slower than QUICK, its use would normally be restricted to emergencies seldom found in engineering problems.

The principal features of QUICK are enumerated below:

#### DATA INPUT

Input is in decimal, eight significant digits with sign

and an exponent whose range is  $\pm 38$ . Data are converted to binary and stored in the location designated on the data card. In continuous reading, input is accomplished at the rate of 600 floating point numbers per minute.

#### DATA OUTPUT

Output is in decimal, eight significant digits with sign and an exponent whose range is  $\pm 38$ . Results may be printed in any of six print positions, up to six results per line. In addition, a four-digit identification number may be printed in the left margin. In continuous writing, 900 floating point numbers may be printed per minute.

#### ARITHMETIC

All arithmetic operations, except those on addresses, are carried out in floating quaternary point arithmetic. Calculations are carried out to thirty-five bits of significance, rounded to twenty-eight, and the result stored with its associated seven bit exponent. Exponents too large or too small for the range cause the machine to stop or the result to be made exactly zero respectively. The internal range of the exponent is the same as that for input and output.

#### STORAGES

Only electrostatic is accessible. At present, 2560 half-words are available for the programmer to divide as he will between instructions (half-words) and data (full-words). It is expected that this will be reduced to approximately 2000 half-words upon completion and insertion of the proposed function programs.

#### SPEED

Arithmetic operations requiring floating are performed at an average rate of more than four hundred per second. All other operations are performed at rates between 1060 and 2700 per second. See table below for more complete break-down.

CHECKING

Memory-print-out at the rate of 1200 words per minute, and tracing at the rate of 900 instructions per minute are available. Tracing yields the location of every sixth instruction and the contents of the A register in floating decimal notation.

OPERATIONS

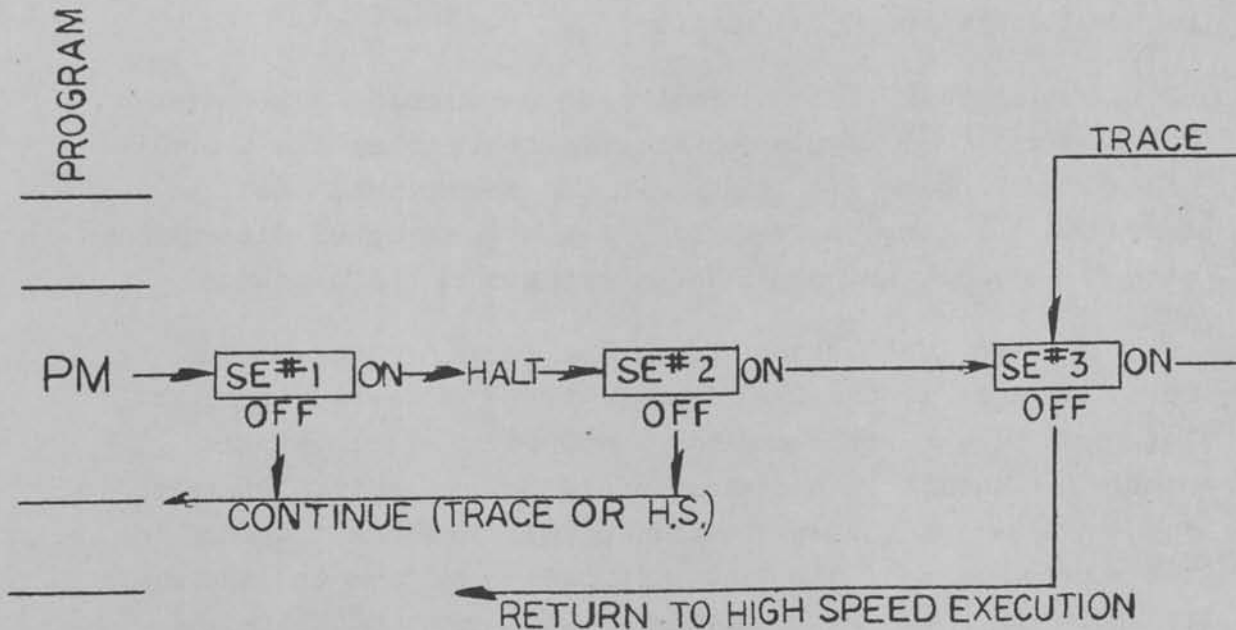
The table below lists the possible operations, with codes, times, and result of the operation. Times are approximate and will be increased by excessive floating. Programming is done in alphabetic code.  $\alpha$  is the address part of the abstract instruction. A and M are permanent registers;  $c(\alpha)$  means the contents of electrostatic location  $\alpha$ ; and  $a(\alpha)$  means the address part of the contents of location  $\alpha$ .

CODE	OPERATION	RESULT	TIME M.S.
AD	Add	$c(A) + c(\alpha) \rightarrow A$	2.57
RA	Reset Add	$c(\alpha) \rightarrow A$	.66
MA	Multiply Add	$[c(M) \times c(\alpha)] + c(A) \rightarrow A$	3.15
SU	Subtract	$c(A) - c(\alpha) \rightarrow A$	2.57
RS	Reset Subtract	$-c(\alpha) \rightarrow A$	.66
MS	Multiply Subtract	$-[c(M) \times c(\alpha)] + c(A) \rightarrow A$	3.15
MP	Multiply	$c(M) \times c(\alpha) \rightarrow A$	2.27
DV	Divide	$c(A) \div c(\alpha) \rightarrow A$	2.28
AV	Absolute Value	$ c(A)  \rightarrow A$	.66
DA	Add Address	$c(A) + a(\alpha) \rightarrow A$	.94
AR	Reset Add Address	$a(\alpha) \rightarrow A$	.94
US	Subtract Address	$c(A) - a(\alpha) \rightarrow A$	.94
SR	Reset Subtract Addr.	$-a(\alpha) \rightarrow A$	.94
TS	Store Address	$a(A) \rightarrow \alpha$	.66
LM	Load M	$c(\alpha) \rightarrow M$	.66
AM	A in M	$c(A) \rightarrow M$	.73
ST	Store	$c(A) \rightarrow \alpha$	.66
NO	No Operation	No operation	.49
HT	Halt and Transfer	Halt, Transfer to $\alpha$	.37
TR	Transfer	TR to $\alpha$ Unconditionally	.37
TZ	Transfer on Zero	TR to $\alpha$ if $A = 0$	.71
TP	Transfer on Plus	TR to $\alpha$ if A is positive	.71
TN	Transfer on Neg.	TR to $\alpha$ if A is negative	.66
PM	Post Mortem	(see below)	
RD	Read	(see below)	
PR	Print	(see below)	
RT	Square Root	$\sqrt{c(\alpha)} \rightarrow A$	3.77
FU	Function	$f [c(A)] \rightarrow A$ (see below)	
AZ	"almost zero"	addr. gives size of $\epsilon$ .	

Most of these operations are self-explanatory. An easily memorized two letter alphabetic code is used. Note also that the codes for fixed-point operations on addresses are identical to their floating-point counterparts with the letters interchanged. Mixing fixed and floating-point operations should be avoided unless the internal operation of QUICK is thoroughly understood. The several operations Post Mortem, Read, Print and Function, require explanation and are interesting from the development standpoint.

Post Mortem (PM) is the code which controls tracing, when executed in conjunction with three sense switches. A high speed tracing system with only partial display of register status was considered to be more useful than a complete display with much slower tracing. Consequently, QUICK traces at six times the conventional rate but prints only the contents of the A register, plus every sixth location for orientation.

A valuable feature of the tracing procedure is the ease with which it may be initiated or discontinued. PM codes are sprinkled judiciously through the program or may be keyed in manually. With sense switches off, execution of a PM is identical to execution of a No Operation.



Inspection of the diagram and the possible sense switch settings discloses the flexibility of the arrangement. When a PM is reached, the operator may at will continue high speed calculation indefinitely, continue high speed to the next PM, initiate tracing, discontinue tracing and return to high speed, continue tracing indefinitely, continue tracing to the next PM, etc. In addition, tracing may be discontinued at any time by turning the sense switches off, whereupon high speed computation is resumed. During tracing, all normal printing is suppressed.

A simple example may clarify the incomplete diagram. Suppose a program contains ten PM operations and it is desired to trace between the fourth and sixth Post Mortems only. Placing sense No. 1 on, No. 2 and No. 3 off, the program is started. Upon reaching the first PM, a program stop occurs. The operator depresses the start button and notes that the first PM has been passed. Counting the stops encountered, he starts the machine two more times, arriving at the fourth Post Mortem. Senses No. 2 and No. 3 are now turned on, the machine is started, tracing is initiated and continues to the fifth PM. The start button is depressed once more and tracing continues to the sixth PM. Placing senses No. 1 and No. 3 off and starting the machine causes return to high speed calculation.

Briefly then, one may desire to continue or discontinue either tracing or high speed calculation. Sense No. 1 provides for stopping the program at the appropriate position, sense No. 2 allows the decision of continuation or discontinuation to be made, and sense No. 3 implements the decision.

The Read instruction causes the floating decimal numbers, on cards in the Card Reader, to be converted to floating quaternary numbers and to be stored in electrostatic memory. A data card contains four floating decimal numbers, each with an associated location. This location is added to the address part of the Read instruction to form the address at which the floating-point number will be stored. This

feature enables the programmer to use symbolic coding, meanwhile maintaining the same simplicity of data input as is possible in absolute programming. Reading continues without further instructions until a blank card is reached or an appropriate punch is detected, at which point execution of the abstract program is resumed.

Execution of a Print code produces various results, dependent upon the address part of the instruction. Addresses 0 - 6 are storing addresses and 10, 20 and 40 are printing addresses. Specifically, an address of zero stores the address part of the contents of the A register for future printing at the left margin as an identification. Addresses 1 - 6 store the contents of the A register for future printing at the print positions 1 - 6 (left to right). The printing addresses 10, 20, and 40 cause (1) single, double or quadruple spacing before printing, (2) conversion of the numbers stored by addresses 0 - 6 to decimal, (3) printing of these converted numbers. Where duplicate storing addresses are given between printing addresses, the last number stored will be printed. Where a storing address is not used, that print position will be blank. The printing addresses 0 - 6 may be programmed in any order. Continuous printing (150 lines per minute) is possible where computing time between printing addresses does not exceed approximately 100 milli-seconds.

The Function code is another whose address determines the specific operation to be performed. Dependent upon this address, some function of the contents of A is computed and stored in A. The instruction is run through a multi-way switch, similar to the main switch, and thence to the particular section of electrostatic memory containing the pertinent program.

The functions are not yet completely programmed. In order to increase the usefulness of QUICK and to further accelerate programming, plans call for inclusion of about twenty functions. Preliminary work indicates that this can

be accomplished with less than four hundred instructions, placing the emphasis on speed for six or eight most frequently used functions, and emphasizing conservation of storage for the others. In addition, several codes will be available for insertion of unusual functions not included in the twenty above. Similar to a library subroutine, a simple binary card loading process will make their utilization in a program as easy as the standard functions.

It may be observed that four of the possible thirty-two operations have not been used. Although many additional useful operations occur to one almost immediately, it was deemed wise to let experience dictate their form. Had the codes been used and later changed to represent more desirable operations, earlier programs would be invalidated.

QUICK programs are written in symbolic coding and are assembled by a slightly modified version of a standard assembly program developed at El Segundo. The assembly program, QUICK, and the alteration instructions for tracing, are stored semi-permanently on one drum, providing fast and handy access. Assembly of a QUICK program produces a binary program deck, parallel listing of the program (before and after assembly), and stores the assembled program in electrostatic memory, ready for running.

It was intimated in the opening paragraphs that the area of economical application of QUICK is greater than might be suspected. Versus non-abstract, fixed point symbolic programming, QUICK offers the following economic advantages:

1. Fewer re-assemblies, through fewer programming errors and hence less machine time for check-out.
2. No time spent in tape search for subroutines, nor in reading subroutines from tape.
3. Reduced programmer's time through faster initial programming and less correction time.

Against these advantages must be weighed the disadvantages of greater machine time for production runs. The ratio is of

the order of twelve to one. This appears to be an overwhelming argument against QUICK until one considers the actual rather than the percentage increase.

The solution of many problems can be effected by execution of less than a million abstract instructions. In these cases, QUICK will almost invariably produce the result more economically. Consider that the abstract instructions are executed at the rate of approximately 60,000 per minute, that programming time is cut from fifty to ninety percent, that five minutes is the minimum machine time required for additional assembly and check-out, that tape searches of from half a minute to two minutes are completely eliminated, that time for reading data and printing results is not increased.

Suppose now that one has a recurring problem which has been previously checked out. If less than 100,000 instructions are to be executed, QUICK will generally do it in less time.

QUICK is clearly useful in cases where the range of parameters is such that overflow or loss of accuracy occurs for certain parameters at the extremes of the range. Scaling and re-scaling are expensive operations, especially so when the trouble was not predicted and a re-run is required.

The above are clear cut areas in which QUICK excels, despite the training of the programmer. Superimposed upon them, and making feasible the use of QUICK for much larger programs, are the considerations of programmers' inexperience and mental attitude. Where the degree of arithmetic skill is not high, QUICK is a crutch. For the able mathematician, QUICK affords relief from the tedium of bookkeeping, freeing the mind for ingenious programming and clearer analysis.

In addition, the advantages of having DOUBLE QUICK for proof of accuracy and as a double precision tool, available without re-programming, must be evaluated in any decision regarding a choice of programming systems.



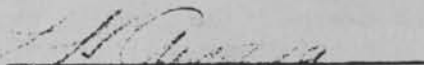
In conclusion, it appears that for all but the largest or best behaved engineering problems, use of QUICK may be advantageous. The inherent properties of abstractions, symbolic coding, and floating point arithmetic should combine to simplify and accelerate programming in the average computing installation.

MATHEMATICAL ANALYSIS DEPARTMENT STUDY NO. 28

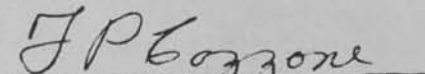
March 17, 1954

AN IBM TYPE 701 MATRIX ABSTRACTION  
AND ITS ENVIRONMENT

Prepared by:

  
H. Amaya  
LELAND  
MATHEMATICAL ENGINEER

Approved by:

  
F. P. Cozzone, Manager  
Mathematical Analysis Department

Prepared	NAME L. H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	TEMP.	FORM.
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT	Model		1
Approved				MAS 28		

The purpose of this paper is to present an IBM Type 701 matrix abstraction, with emphasis on the environment in which it was developed and the one in which it now operates. Environment in this case means the reasoning behind the matrix abstraction, its evaluation, its particular function, programs used in conjunction with it, applicable problem types, programming techniques, and the philosophy of its operation. In addition, examples will be given to illustrate the effect of the abstraction on such aspects as economy and efficiency when performing particular matrix operations. Emphasis is not placed on the detailed coding of the routine, number of commands used, portion of electrostatic memory occupied, and other pertinent facts because the coding must usually fit a very particular set of restrictions, whereas the environment is of a much more general nature and consequently of more interest at large.

At the outset of developing the matrix abstraction (January, 1953) we at Lockheed had no experience whatsoever with actual problems on the IBM Type 701 or on any machine of similar type and magnitude. In fact, at this time we were still in a state of ecstasy with the thought of tremendous speed and a wealth of memory, as compared to the IBM Card-Programmed Calculators and IBM Type 604's which we were then operating. The task before us was to prepare ourselves and the 701 for the solution of problems involving matrices. One of the first steps was to consider the matrix operations which were being done on the existing equipment. The bulk of matrix additions, multiplications, inversions, and simultaneous linear equation solutions were run on the 604. The majority of the eigenvalue problems, that is those requiring the solution of all roots and vectors, were solved using both the 604 and CPC. The high order eigenvalue problem which required the solutions of only the dominant roots and vectors and the low order problem requiring all roots were done entirely on the CPC along with such problems as low order simultaneous linear equations resulting from least square problems. The operations being done on the 604 were generally of the three address type, that is, two operands were fed into the machine and a resultant matrix emerged punched in cards. The resultant matrix was in a form which could be either an operand in the next operation or be printed. The flexibility, the third address, of such a system is due to the manual and mechanical operations performed between the 604 calculations. The disadvantage, however, is that the handling has always been the greatest source of error.

The question then: How do we program matrix operations for the 701 so that they can be used on various order matrices, in any sequence, and still retain the present flexibility? The direct approach might be, since we have a machine which will do the job of both the 604 and the CPC, to use the 701 in the capacity of either the 604 or the CPC with a stored program which is equivalent to the corresponding plugboard. As absurd as it is to use the 701 in this manner, the actual cost of doing the job would be less than on 604's as will be illustrated later. Assuming we have 701 routines to perform

Prepared	NAME L.H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	TEMP.	FORM. 2
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT	Model MAS 28		
Approved						

each of the operations, the next question considered was how to eliminate all the manual handling required between two 604 matrix operations. It looked as though we needed a special routine to load the data, other routines to perform the operations, routines to do the handling mentioned above which includes storing intermediate results, and still others to cause output of results. With the routines mentioned we can solve the handling problem as it was associated with 604 and CPC operation. However, we have magnified the task corresponding to that of placing the plugboard into the 604 or the CPC, i.e., the manual handling of all the routines and their linkage. Also, manual juggling of routines is considerably more susceptible to error than is the handling of plugboards.

How can we eliminate the handling and manual linkage of the routines. One possibility is to keep the routines in symbolic form punched in binary cards and have a 701 assembly program which is capable of arranging the routines in a prescribed sequence prior to the running of each problem. In addition, we must have a special program or have incorporated in the operational routines, a scheme for locating the result of one operation in position for the next command to operate on.

The next step in the direction of the complete abstraction might be to locate all of the routines in a specified portion of the machine memory, and assemble them, as required, from the memory rather than from cards.

Finally, of course, there is the complete matrix abstraction which performs matrix operations and logic automatically upon command. The decision to be made between the last two phases of automation is: Shall we retain the flexibility and assemble the program for each problem, or shall we forfeit some flexibility and gain from simplicity? At Lockheed we chose the latter.

The matrix abstraction, through IBM Type 701 machine coding, alters the machine to one which performs, as such, operations and logic required in the general matrix problem. The basic routine accommodates 24th or lesser order matrices with real or complex elements. To provide for matrix storage, sixty portions of the total memory have been assigned abstract addresses which will store a maximum of sixty 24th order complex number matrices.

The arithmetic operations are performed double precision floating point, with the number word consisting of a sign bit for the characteristic, a sign bit for the exponent, a 27 bit characteristic, and a 7 bit exponent. The matrix number word is exactly the same composition as the number used in FLOP. (FLOP, a contraction for 'floating octal point, is Lockheed's general floating point abstraction for the 701.) Because of the exactness of the number words, the systems may be used in conjunction as one very complete abstraction. Together the two abstractions provide for accomplishing both scalar and matrix arithmetic in floating point, and for

IS A MODIFIED THREE ADDRESS SYSTEM WHICH

Prepared	NAME L. H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	FORM 3
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACT- ION AND ITS ENVIRONMENT	Model	MAS 28
Approved					

performing logic either in machine commands or in one of the abstractions. Both the matrix abstraction and FLOP use full words for commands as they do for numbers; however, the command word structures are somewhat different.

### MATRIX OPERATIONS

#### COMMAND WORD

In the command word, the following components are specified as required for each instruction.

- OP = Operation
- A = Storage address of [A]
- $\ell$  = Number of rows in [A]
- m = Number of columns in [A]
- B = Storage address of [B]
- n = Number of columns in [B]
- $\beta$  = As explained in commands

The number of rows in [B] is specified by  $\ell$  or m depending on the command. Each component is 2 octal digits except  $\beta$  which is four. A and B can be any of the 60 matrix storage addresses.

#### ARITHMETIC OPERATIONS

- ADD [A] + [B]      Computes the sum of the matrix in storage A and the matrix in storage B.
- SUBTRACT [A] - [B]      Subtracts the matrix in storage B from the matrix in storage A.
- MULTIPLY [A] · [B]      Computes the product of the matrix in storage A and the matrix in storage B.
- SCALAR MULTIPLY  $\beta \cdot [A]$       Multiplies the matrix in storage A by the scalar in electrostatic address  $\beta$ .
- INVERT [A]<sup>-1</sup>      Computes the inverse of the matrix in storage A.
- FORM CHECK SUMS      Computes a row and column of check sums for the matrix in storage A.
- CHECK      Checks the sum of the elements of the matrix in storage A against its check sums to a specified number of figures. If the matrix does not check, the machine stops and the difference between the check asked for and the actual number of figures it does check to is displayed in the accumulator.

Prepared	NAME L.H. Amaya	DATE 8-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	TEMP.	PERM. 4
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT	Model		
Approved				MAS 28		

The result of the operations ADD, SUBTRACT, and MULTIPLY may be stored in any of three specified matrix storage locations. For the SCALAR MULTIPLY and INVERT operations, the result location is fixed. Check sums are automatically calculated on the commands ADD, SUBTRACT, MULTIPLY, and SCALAR MULTIPLY.

#### LOGICAL OPERATIONS

**TRANSPOSE**                      Forms the transpose of the matrix in storage A.

**STORE**                              The contents of matrix storage B is replaced by the matrix in storage A.

**STORE  $\beta$**                           The matrix in storage A replaces the contents of a portion of electrostatic memory where  $\beta$  specifies the electrostatic address of the (1, 1) element. The command also can reverse itself and store any portion of electrostatic in any matrix location. The command is used for expanding and diminishing the order of a matrix.

**TRANSFER**                          Causes the machine to take its next matrix command from electrostatic address  $\beta$ .

**FLOP**                                  Causes the machine to interpret all subsequent commands as FLOP commands.

**WRITE END OF FILE AND REWIND**      Causes the tape addressed to either Write EOF and rewind or just rewind depending on whether the tape is in write or read status.

#### INPUT - OUTPUT OPERATIONS

**LOAD**                                  Loads the matrix from cards into matrix storage A.

**PRINT**                                  Prints the matrix in storage A. The elements are shown as a characteristic and decimal exponent, and are printed in matrix form. A matrix identification number is printed, as well as row and column identification which is printed down the left side and across the top of the page.

**PRINT (SENSED)**                      Prints the matrix in storage A or skips the command depending on the position of a sense switch. The command is used for printing intermediate results, usually while a problem is being checked out.

**PUNCH**                                  Punches the matrix in storage A. Each card contains a matrix identification number and matrix row and column identification of the first element in the card, a card check sum, and a maximum of 11 complex elements.

Prepared	NAME L. H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	TEMP.	PERM. 5
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT	Model MAS 28		
Approved						

The descriptions given with each command are not necessarily complete, although the basic function is indicated.

The time (to the nearest second) required to perform the basic arithmetic operations is tabulated below. The time shown is for complex number operations, and for both operands of the same order with no zero elements.

<u>ORDER</u>	<u>ADD and SUBTRACT</u>	<u>MULTIPLY</u>
2nd	2 <i>su</i>	3
6th	3	8
10th	4	30
16th	7	106
24th	14	304

Since the system is a modified three address, the operation times include access of the two operands and time for storing the result. For real number operations the time is slightly less than one-third of that shown for complex number elements. The reason complex operations were incorporated into the routine rather than using double order real operations is apparent when one compares the time involved. The 24th order maximum in the routine was selected because it is a practical one for the 701 and also seemed satisfactory for the anticipated problems. If the problems exceed 24th order the abstraction may still be used since it readily lends itself to partitioning methods <sup>1</sup> for all arithmetic operations.

Having brought the matrix abstraction through its development stages, let us now consider the environment in which it operates. The digital group at Lockheed is purely a service group involved primarily with the solution of design and production type engineering problems. These problems vary in magnitude from large to very small with respect to programming time, machine time, and combinations of both. Therefore the computing group must be capable of solving almost any problem. Not only must the problem be solved but it must be solved within a specified time which may be anything from immediately to a year hence. With such a wide range of possibilities it is absolutely necessary that some preparation be done in advance if the customer is to be satisfied. We consider the matrix abstraction as part of this preparation.

Everyone should be and usually is interested in the economy and efficiency of the computing operation. As insignificant and inefficient a job as could be chosen for the IBM Type 701 is the single multiplication of two matrices where both operands are in cards and the results are to be printed. For illustration purposes, let us perform the operation of multiplying two 10th order matrices to form a 10th order product, and compare the time and cost of doing this job by hand, on the IBM Type 604 and auxiliary equipment, and on the IBM Type 701 using the matrix abstraction.

Prepared	NAME L.H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION		Page	TEMP.	PERM. 6
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT			Model MAS 28	
Approved							

	HAND	IBM TYPE 604	IBM TYPE 701
Elapsed time required for the job not including punching.	8.5 hrs.	50 min.	25 sec.
Time ratio for doing the same job with respect to doing it on the 701.	1044	109	1.0
Cost ratio to do the job with respect to the cost of doing it on the 701.	33	15	1.0

*for 10<sup>th</sup> order matrix*

The time quoted for each method is that to do only the stated matrix operation (or continuously repeat the same operation) at a production rate of speed. Again, the example chosen is of the worst type for the 701 because of its small percentage of computing as compared to the input and output time; however, it does illustrate, contrary to common belief, that it is at least practical to do the smallest of matrix problems on a large machine. Another point of interest is the fact that with the program in the machine it takes only four matrix commands to perform the illustrated matrix multiplication. The high ratio of machine to abstract commands is obvious and is of course one of the prime reasons for the utility of the abstraction.

The first few problems attempted with the matrix abstraction were of the ideal type. They were the formation of aerolastic matrices<sup>2</sup> and contained approximately fifty serial matrix operations to be performed for three sets of data. The program was coded and checked practically error free. The 701 also cooperated, doing the problem without any machine errors. You can imagine that by now we had the impression that the abstraction was the alleviator of all the 604 and CPC procedure ills. Then came the task of programming the Myklestad method for dynamic structural analysis. The method is full of matrix operations and looked ideal for the abstraction. The structural analysis problem is what we call a routine problem, one which may use the same program for solution thousands of times with varied data over a period of years. Since the program will run hours on the machine it behooves us to make the program as efficient as possible at a cost of programming time which is eventually more than saved in machine time.

Well, the matrix abstraction did not achieve the utmost in machine speed on this problem due to the fact that the program left the matrix abstraction frequently between matrix commands for non-matrix logic and calculations of single elements. Since it was necessary to reload the matrix abstraction from the magnetic drum to electrostatic after each non-matrix operation, it soon became evident that about 15% of the machine time could be saved if the abstraction were not used. In connection with the problem mentioned it should be emphasized that it was of the routine type and additional time was available for coding. The job stated is perhaps an extreme because it



Prepared	NAME L. H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	TEMP.	FORM. 7
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT	Model		
Approved				MAS 28		

could not be planned in such a way as to avoid the drum time involved; however, many problems may be restated and the operations done in a sequence which will allow the matrix operations to be done without referring to other means for non matrix logic and computations between the matrix commands.

A routine job such as the  $|A-B\lambda|$  eigenvalue problem which consumes a large percentage of the 701 machine time at Lockheed is another example of the type problem which should definitely not be programmed in the matrix abstraction even though floating point is used. As explained, if the job is routine and will eventually run for possibly hundreds of hours, the programming time spent on the routine to save machine time will be paid for manyfold. There is a point of diminishing return, however.

Opposed to the aforementioned routine type problems, we recently had a problem from the preliminary design group which had to do with the sweep angle and aerolastic design criteria of a wing. The details are of a classified nature; however, to be of real value, the customer needed answers immediately. The problem involved approximately 100 matrix operations and used almost every command in the matrix abstraction. There were very few cases run, which, including check-out time, only consumed 4.5 hours of machine time. The programming time was a minimum and required only one man full time and one more part time, a total of about 25 man hours for the job. Needless to say, the job probably could not have been done in less elapsed time any other way, and because of the number of different matrix operations required and the relatively few cases run, there is much doubt that the program could have been checked out and run in the amount of machine time used even if there had been unlimited elapsed time available.

The two extreme usages of the matrix abstraction have been brought forth in the previous examples. Somewhere between the two is a point of diminishing return which, of course, will vary appreciably with the computing organization. For example, if you are caught short for coders, or with inexperienced coders, the point of diminishing return will be far removed from where it will be if you have a large staff of very experienced coders. Thus the matrix abstraction assumes its place in the organization.

Another very important usage of the matrix abstraction is in research of matrix methods, such as any number of methods for solution of the eigenvalue problem which can be broken down into a sequence of standard matrix operations. In the search for a method, we again have a consideration: Shall we spend weeks coding the method and find it is inadequate after 15 minutes running time on the machine or shall we code it in a few days and find the method doesn't work after 20 minutes run time? If the method works the first time, there are always refinements to try which may be programmed with many times fewer changes than if the program were in absolute coding. Also in trying to develop a method, one is perplexed with the programming,

Prepared	NAME L.H. Amaya	DATE 3-17-54	LOCKHEED AIRCRAFT CORP. CALIFORNIA DIVISION	Page	TEMP.	PERM. 8
Checked			TITLE AN IBM TYPE 701 MATRIX ABSTRACTION AND ITS ENVIRONMENT	Model MAS 28		
Approved						

perhaps the inadequacy of the method, machine errors, methods of checking, and many others which are at least somewhat reduced when using an abstraction. If you have ever found yourself in the above predicament you know that assistance from any direction is more than welcome.

A speaker in Los Angeles once commented that the 701 enables him to find that his method is inadequate faster than by any other means. If he is not using one, it might be added that with the assistance of a matrix abstraction he could probably find the same results even faster.

In conclusion, let us summarize a few of the more interesting points mentioned: A history of development was presented; some flexibility was forfeited for simplicity; the matrix abstraction should be used with other scalar abstractions; the abstraction is best utilized, in general, on short one-time jobs, and is indispensable on a long program job which consumes little running time and requires a minimum of elapsed time; very small matrix jobs are practical on the 701; for the most part, no routine job should be coded with the matrix abstraction; there is a place for the matrix abstraction in research of matrix methods; in evaluating a matrix abstraction, attempt to locate the exact role it will assume with respect to the type of problem to be solved, philosophy of operation, personnel in the organization and economy desired; make the abstraction fit your restrictions as well as your requirements.

*(50 to 125  
orders for one  
abstract command)*

*151 full words available for orders  
200 " " on drum*

References:

1. Louis A. Pipes "Matrices in Engineering"  
Mathematics for Modern Engineering; Lecture IV.
2. NACA Technical Report 1000.

# Los Alamos Debugging Programs and Techniques

As Used on the IBM 701

by Edward A. Voorhees, Jr.

## Introduction

If the experience at other IBM 701 installations coincides with our experience at Los Alamos, I believe we may agree that the main bottleneck in the course of a problem is the period beginning after the coding of the problem has been assembled on cards and ending with the successful calculation of the first results, i.e., the debugging period. Also, in some problems, the code will never take on a fixed form, for with the entry of new parameters it is often necessary to modify the code and, in some cases, to actually recode portions of the problem. Frequently, this situation will require the use of debugging programs and techniques.

At present, there seem to be two main general debugging methods: (a) memory print-out and (b) tracing. Memory print-out may be defined as the listing of a stored program (or a selected part thereof) whose instructions are not being executed concurrently with the execution of the listing program. A tracing program is one that lists the instructions and certain additional information as the instructions of the stored program are actually being executed. It would seem that, in general, the memory print-out method makes for more efficient use of the machine, whereas, the tracing method makes the detection of coding errors easier for the individual at the expense of the machine. There are quite a few exceptions to this statement which arise because of the particular nature of the error being sought.

At Los Alamos a large majority of the coding (and debugging) is performed by persons who are not full-time coders. Many of these people are very capable coders, but for them coding is only a tool of their profession - a tool not unlike operating a slide rule or hand calculator before the advent of large scale computers. As a result, our debugging programs and techniques have been developed with these people in mind, and the trend has been to somewhat favor the person instead of the machine in the developing of new debugging programs. It has been our experience that the beginning coder will rely almost exclusively on tracing programs and that, as he gains experience, he will make increasing use of memory print-outs. The experienced individual will use either tracing or memory print-out, making his choice on the basis of the nature of the suspected error and the difficulty he anticipates in finding it.

Our method of time scheduling serves, however, as a check against idle or excessive wasted time during the debugging phase of the problem. This is accomplished by scheduling short periods of time, of the order of 10 minutes, in general, for debugging during the daytime and longer periods of time during the evening and night for "production" running.

The particular programs described below are used for debugging programs coded in machine language. Our two interpretive coding systems, the single-address Dual system and the three-address Shaco system, each has its own peculiar debugging program and technique. Since 80% of current problems are coded in machine language, (and the figure is gradually increasing) no further development of debugging routines for these systems

is anticipated.

All memory print-out programs print information in octal, and all tracing programs print information in octal and decimal. Electrostatic storage will be referred to as memory in the remainder of the paper.

#### Memory Print-out Programs

a. 186 is a program to print in octal the contents of electrostatic storage except for the 151 half-words which it, itself, occupies. The program will search memory, beginning at the first half-word following itself, for the first half-word neither plus zero nor minus all ones. It will then print the location of that half-word, the half-word, and the following ten half-words, regardless of the composition of these ten words. It then continues the search and prints whenever the above condition is satisfied. 186 may be located anywhere in electrostatic storage.

186 is commonly employed when the problem stops unexpectedly and 151 consecutive half-words of storage are available. After noting the location of the stop and inserting the print board, the operator loads a properly located 186 deck of cards and the listing is issued automatically. Occasionally, 186 is used at an intentional stop to obtain memory listing. A memory malfunction is occasionally detected through 186, in which case the particular memory drawer at fault can be determined. At present, we are revising 186 to print  $n$  ( $8 \leq n \leq 11$ ) instructions to a line to accommodate those people who find an octal print-out with 8 instructions to the line easier to read.

b. 982 is essentially a 186 program that will print all of memory except for the two full-words with location -0000 and -0002. This is accomplished through a self-loading program that transfers all of memory except for these two full-words to a drum. This program is intended for use with those programs which are so large that 151 half-words are not available for storing 186. After the listing is complete, memory is restored to its original form with the exception of the two full-words destroyed by the self-loading program. The listing is identical with that obtained from 186.

c. 784 is a program which lists all references made by a specified consecutive range of instructions to a specified consecutive range of half-words. The program operates in the following manner. The first half-word of the specified range of half-words, (viewed as an instruction) and its location are printed and marked with an asterisk. Then the address of each instruction of the set of instructions is examined, and, whenever the address is equal to the location of the half-word being considered, the instruction is printed. When the set of instructions has been completely searched in this manner, the next consecutive half-word is printed and the search is begun again. This continues until the set of half-words is exhausted. A control card specifies the location of the first and last instructions and the location of the first and last half-words. When the search indicated by a control card is complete, the program stops and pressing the START button causes the program to read the next control card. The ranges indicated may be of unit length.

This program is quite useful after having the machine COPY CHECK with control being where it was never intended to be. The offending transfer order can readily be detected by searching the code for references to this half-word.

d. 785 is a program to compare original ordinary binary program cards with the program stored in the machine. 786 is identical, except that it compares regional binary cards with the stored program. The program first transfers all of memory, except for the contents of -0000 and -0002, to a drum. It then reads the first card of the coders program, reads the corresponding set of words from the drum, and then prints all half-words that do not agree. The print-out consists of the location of the half-word, the half-word from the program card, and the half-word from the drum. This process continues until all program cards have been checked. The listing is double-spaced after the printing for each card.

This program has been very widely used, not only in debugging codes, but also in the detection of machine errors. If the code has been mutated by a memory malfunction, this will appear in the listing along with the instructions with variable addresses. The incorrect instruction can be spotted almost immediately by a person familiar with the program. Usually, the appropriate action can then be undertaken without delay. Occasionally 785 is used to check a corrected binary deck against an old binary deck where the listed discrepancies are assumed to be known.

#### Tracing Programs

a. 794 is a tracing program that can be used in one of two ways: (a) To list all instructions with a specified operation, and (b) to list all instructions whose addresses lie within a specified range. All listing is done during the execution of the program. A sense switch determines whether the program should or should not stop on negative transfer orders. A second sense switch is used to inform the program as to whether operation or address tracing is desired. It is also possible, by means of a third switch, to have the instructions READ, WRITE, and READ BACKWARD "dummy" executed. If these orders are not "dummy" executed, control is relinquished by the tracing program and given to the input-output program involved. Information printed includes the status of the overflow indicator, the overflow bits, the location of the instruction (octal), the instruction (octal), the sign and first 17 bits of the accumulator (octal), the sign and complete contents of accumulator, MQ, and storage referred to in the address part of the instruction (all as decimal fractions).

b. 796 is a tracing program that lists all transfer and sense orders as they are being executed, i.e. 796 traces logic. A control card determines the location of the first instruction to be traced. A conditional transfer or sense that is not executed, i.e., has no effect on control, is not printed. No sense switches are used by 796. The program has been coded for a two electrostatic frame 701. The application of the program is readily apparent.

c. 795 is a high-speed tracing program which can be used with either a single or double electrostatic frame machine. Provision is made to accommodate any number of "traps", a "trap" being defined to be a portion

of the coder's program which is to be traced. If a portion of the program is not being traced it is executed at full speed. After the coder's program has been loaded, 795 is loaded with  $n+1$  control cards. The first  $n$  cards designate the range of the  $n$  desired traps and the  $n+1$ st card contains the location of the first instruction ( $R$ ) of the coder's program. On each of the  $n$  trap cards is the location of the first instruction of the trap ( $M_i$ ) and the location of the last instruction of the trap ( $N_i$ ),  $M_i \leq N_i$ . As 795 reads the  $i$ th trap control card, it replaces the instruction  $M_i$  with a transfer to a portion,  $D_i$ , of the tracing program. Then 795 stores  $M_i$ ,  $N_i$  and the contents of  $M_i$  in the  $D_i$  block. It then reads the  $i+1$ st control card and repeats the procedure in the  $D_{i+1}$  block, continuing to read control cards until an "R" card is reached, whereupon control is transferred to  $R$ . Each  $D_i$  block is 6 half-words in length, hence the number of traps which may be specified is limited to the amount of space which is available in the machine for the trap table ( $D_i$ ) block.

When the coder's program reaches the instruction  $M_i$ , control is transferred to 795 and tracing begun, with or without printing depending on the position of a sense switch. When instruction  $N_i$  is reached, it is traced and afterward control is relinquished by 795 and given to the coder's program at instruction  $N_i+1$ .

When a trap is encountered, the paper is spaced, and the contents of the accumulator, MQ, and overflow bits before the execution of the first instruction are printed, provided the "print" switch is on. The paper is spaced whether printing occurs or does not occur. If the instruction READ, WRITE, or READ BACKWARD is encountered while tracing, it is listed but not executed. All COPY orders are "dummy" executed by loading the MQ with the contents of the memory location referred to in the COPY instruction. All other instructions, including  $\pm$  SENSE 40<sub>8</sub>, and  $\pm$  00 and  $\pm$  01 transfers between the first and second banks of memory, are executed.

If, while a particular trap is being traced, the coder no longer wishes to trace this trap, he may "erase" this trap by depressing a sense switch which will replace the "transfer to tracing" order in  $M$  with the original instruction.

The information printed consists of the location of the instruction; the instruction; the status of the overflow indicator; the overflow bits; and the sign and contents of the accumulator, MQ, and storage location being referred to in the address part of the instruction in octal and decimal.

### Conclusion

These programs constitute the major portion of our library of programs used in debugging problem programs. Any of these programs are available to IBM 701 installations. We at Los Alamos would be interested in any suggestions for the improvement of these programs. We would also be interested in any ideas for debugging programs that you have utilized. We feel that only through the exchange of ideas can each installation benefit from the experience gained at other installations.

# Los Alamos Coding System and Assembly Program for the IBM 701

by Dora W. Sweeney

The process of coding, coding assembly, and check-out of the assembled code is the most demanding upon the coder's attention and, in many cases, his time. Therefore, it is necessary to furnish the coder with a simple but highly versatile coding system and assembly program, and a variety of utility programs to aid him in the detection and correction of coding errors.

As a result of our self-service system at Los Alamos, many people with virtually no experience with internally programmed machines have undertaken the coding of large problems for the 701. The first such coders are now quite adept, but more people are still starting to learn to code. This has required a versatile coding system with as few restrictions and conventions outside of the use of the thirty-two operations as possible, because of two reasons. First, that both the new and experienced coder use a system based as closely as possible upon the machine language, and second that the new coder be unhampered by restrictions so that he may be free to devise new coding techniques. Another requirement is that the assembly program for this system must be easily usable and virtually fool-proof.

## Regional Programming

Regional programming is one of the coding systems which fits these requirements and is used almost exclusively at Los Alamos. A brief description of the system follows.

The coder uses three main divisions of electrostatic storage. They are storage for problem constants, storage for numbers calculated by the code, and storage occupied by the code itself. Any or all of these divisions may be sub-divided. For example, the storage for numbers calculated by the code may be sub-divided into erasable storage used in certain coding loops and storage used to contain numbers for later printing. Using regional programming, the coder assigns a letter to distinguish each variety of storage. These letters are in the range from A to H and from J to Q. The letters A, B, E, and F are the most commonly used and denote respectively storage for universal constants, storage for problem constants, erasable storage, and storage for code.

The coder makes a flow diagram or breaks the problem into small logical calculations. Then each block of the flow diagram or logical part of the problem is assigned a number in the range from zero to ninety-nine.

This number and letter are combined and called the regional index, and the coder may distinguish at a glance the part of the problem and type of storage referred to. Each regional index is followed by a regional location which is in the range from 0000 to 4095. Coding proceeds with the coder filling these various blocks of storage starting from regional location zero.

The coding form used is divided into columns which correspond to card columns. One column contains a digit to designate whether this is a control

card or a card containing coding. The next seven columns contain the regional location index and regional location. The next three columns contain the sign and operation parts of the address. The next seven columns contain the regional address index and regional address. Then, twenty columns are allowed for comments about the code followed by eight columns which contain the alphabetic abbreviation of the operation. Each line of the coding form contains one half-word instruction and corresponds to one punched card.

When the coding is finished, the coder prepares a control card for each regional index. This control card indicates the absolute location in electrostatic storage which the assembly program will assign to the regional location. The coder may assign these locations so that there are no gaps in his code or may locate different regions with gaps between them to be filled with other sections of code. In the course of coding, it is necessary to use operations whose address parts do not refer to electrostatic memory, such as shift orders. This type of address is characterized by the regional index, OOR, and no control card is necessary for this index, as the address is already absolute and does not need to be located. After the control cards are prepared, the coder is ready to use the assembly program.

#### Regional Assembly Program

Regional Assembly Program, 607, has been written at Los Alamos to replace a similar program, called 802, written by IBM. 607 uses the Card Reader, Printer, Punch and Electrostatic Storage only. Normally, all control cards are read into the 701 prior to the reading of the cards containing the regional code. Each card is read, assembled, checked, and stored in electrostatic storage before the next card is read. After each card is assembled, its location is checked against the previous card's location. If the locations are in consecutive order, card reading is resumed until forty-four cards (forty-three cards in the case of regional binary punching) are assembled. If non-consecutive locations are encountered or if forty-four cards have been read, card reading stops, one binary card is punched and a page is printed showing the original code and the assembled code in both decimal and octal notation. Then card reading is resumed.

607 allows three types of control cards. One type is used to assign absolute locations to regional locations and regional addresses; one type is used to expand or contract regional locations and addresses; and one type is used to change one regional index to another. Up to two hundred control cards may be entered.

607 also allows one other type of card which is used to introduce half-word or full-word constants into regional locations before assembly. This type of card contains the usual regional location index and regional location, but the operation and regional address are zero, and the regional address index is OOR. The sign of the operation specifies whether the information punched in the comments columns is to be converted to a half-word or a full-word. The constant is considered a fractional number consisting of twelve digits and a sign. It is followed by a two digit decimal scaling factor and a two digit binary scaling factor. The decimal scale factor is in the range 00 to 11 to indicate the number of integers in the constant. The binary scale factor is in the range 00 to 35 to indicate the number of binary integers in the converted constant. In the



case of a half-word constant, the constant is converted to binary and stored in the operation and address part of this card's location. In the case of a full-word constant, the left half-word is stored in this card's operation and address part, then another card image is formed in electrostatic storage with location one greater than the previous location, and the right half-word is stored in this card image's operation and address part.

607 checks each card for double punching and blank columns. It checks each original regional location index, regional location, operation, regional address index and regional address to see if they are within the prescribed ranges. It checks each assembled regional location index, location, regional address index, and address to see that they are still within the prescribed ranges. It also checks that there is a control card giving an absolute location for each regional location and address. 607 also checks each constant for double punching or blank columns and checks that the binary scaling is large enough to accommodate the decimal scaling, and that the rounding of the converted twelve digit number to thirty-five bits (or seventeen bits) does not overflow.

All printing and punching is controlled by Sense Switches. 607 contains one print program and three different punch programs. The first punch program punches binary cards containing up to forty-four half-words in the eight through the twelve-row of the card. The nine-row contains the check sum, the half-word count, and the location of the first word. The second punch program punches new regional decimal cards. The third program punches what we term regional binary cards.

Let me digress here, briefly, to explain regional binary cards and their importance to the 701 user.

#### Regional Binary Cards

Every half-word of code consists of an operation and address. This address must either refer to a location in electrostatic, or, in the case of operations such as Read, Write, Sense, or shift orders, not refer to any electrostatic location. We will define all addresses which refer to electrostatic locations as variant addresses. All others will be defined as invariant addresses. The significance of this definition is that a variant address will change if the program is relocated in electrostatic, but an invariant address will not.

A regional binary card contains a designation as to the variance or invariance of the address of each half-word on the card. Regional binary cards contain up to forty-three half-words per card instead of the forty-four on ordinary binary cards.

This designation of variance or invariance of addresses now leads to several important new applications. All suitable utility programs, such as print, punch, tape and drum programs, have been assembled in regional binary form so that the coder may locate these programs where he pleases in the machine without having to reassemble them as a part of his own code using 607. A self-loading, card-reading program, called 025, for loading regional binary cards has been written which allows the user to enter an increment to be added to all locations and variant addresses on succeeding regional binary cards. Several sets of regional binary cards, each with a different increment may be loaded successively. Probably

the most important application is the regional binary assembly program, 620.

#### Regional Binary Assembly Program

Regional binary assembly program, 620, will expand a program and insert new orders, contract a program and delete unwanted orders, replace incorrect orders with correct orders, and permanently relocate programs.

620 reads forty-three half-words per card compared to 607's one half-word per card. 620 also prints seven times as much octal information per page as 607 does. Only one control card is allowed in 620 at one time, but this card may contain a lower cut-address, and increment, and an upper cut-address. This allows the user to add an increment to any location or variant address greater than or equal to the lower cut-address, but less than the upper cut-address.

All half-words are read, assembled and stored as full words, with location as the address part of the left half-word and the sign, operation, and address in the right half-word. The operation part of the left half-word is "STOP" if the address is variant and "TR" if the address is invariant. These full words are all positive. Drums 128 and 129 are reset to the full words, minus zero. Now each location is doubled and used as the Set Drum address for Drum 128, if it is in the range 0000 to 4094, and for Drum 129, if it is in the range 4096 to 8190; then the full word is stored in that drum location. This gives a full-word drum location corresponding to each half-word electrostatic location. After this "sorting" operation, the drums are searched, new binary cards are punched, and octal listings printed.

#### Correction of Errors

Los Alamos' procedure for the detection of errors is explained in another article; the remainder of this paper will be devoted to methods of correction of such errors.

If the coder is interested in the correction of an instruction, he may use one of several methods. The direct correction of the binary card is the most commonly used. The coder may block out incorrect punches in the binary card and then punch the correct bits, or he may only correct the instruction itself and use programs 925 or 926 which will read the corrected card into electrostatic storage, recompute the check-sum and punch a correct card. (925 operates on ordinary binary cards, and 926 operates on regional binary cards.) Two other methods of correction are available, but both require more 701 time than the previous method described. The coder may punch a correct regional decimal card, and reassemble a correct binary card using 607; or he may punch a card similar to a regional binary card containing the correct instruction, place this behind the incorrect regional binary card, and, using 620, obtain a correct regional binary card.

The most difficult part of code correction is the insertion of several additional orders in the middle of the code. If there are very few such corrections, the coder replaces the instruction following the place where the insertion is to be made by a transfer order to some unused portion of electrostatic storage. There he locates the orders to be inserted and the replaced order and transfers back to the main code.

If many of these so called "patches" are inserted, it becomes very difficult to follow the sequence of the program.

Both 607 and 620 allow the coder to form gaps in his code in which the inserted orders are to be located. In this case, as well as the case in which the coder deletes unwanted orders and closes up a gap, the 701 does the "bookkeeping" so that increments are properly added to the locations to be changed, and so that all addresses are also properly changed. This method, using either 620 or 607, allows the coder to keep his program in sequential electrostatic locations without unnecessary transfers. In actual practice, the coder uses 620 to correct his program, then, if necessary, makes a final assembly with 607, after inserting the regional decimal correction and control cards, to obtain a final correct listing showing the program comments and absolute decimal and octal program.

S I N G L E   A D D R E S S   A S S E M B L Y   P R O G R A M

Date written  
July, 1953

Written by  
Ed Law

With suggestions from  
Jack Strong  
Charles Davis

Report written  
March, 1954

Written by  
Ed Law

With suggestions from  
Jack Strong  
Frank Wagner  
Florence Anderson  
Roger Mills

# CONTENTS

	Page
GENERAL DESCRIPTION	1
Objectives	1
Philosophy	1
Capacity	4
Speed	4
Components Used	5
Checks	5
Input	6
Output	6
Assembled Program Location	6
Electrostatic Storage Use during Assembly	7
Block Diagram	8
SYMBOLIC INPUT	10
Location	10
Operation	10
Alphabetic Operation Codes	11
Alphabetic Type Codes	11
Type	12
Address	12
Remarks	12
Examples	13
MACHINE ASSEMBLY	15
Symbolic Decimal Card Form	15
Card Order and Reader Board	15
Sense Switches	15
Programmed Stops and Sense Lights	16
Printer Board and Alteration Switches	17
Printer Board Wiring	18
Description of the Listing	19
Punch Panel	19
Binary Card Form and Loading	19
DETAIL LISTING PART I	21
Coding	21
Data	25
Temporary	25
DETAIL LISTING PART II	26
Coding	26
Data	32
Temporary	32

## SINGLE ADDRESS ASSEMBLY PROGRAM

### GENERAL DESCRIPTION

#### OBJECTIVES:

- 1 To provide a means of converting a program written in a symbolic decimal and alphabetic form to an absolute binary form.
- 2 To keep the symbolic form used as simple and direct as possible.
- 3 To avoid having to break any but the very largest program into blocks for assembly.
- 4 To produce a listing of the assembled program showing the symbolic and absolute forms as well as a description of the steps taken.
- 5 To punch binary cards which are ready for loading and execution. Each binary card to contain word count, initial address, check sum, and a signal as to whether or not it is the last card of a group.
- 6 To approach as nearly as possible the speed of continuous card reading and printing regardless of the size of the program being assembled.
- 7 To provide certain checks on the program being assembled.

#### PHILOSOPHY:

The reason for wanting to write a program in a symbolic form rather than in an absolute form is found in the programmer's desire for "elbow room" in developing the program. The analysis of even the best organized problem may be found in fault and changed after the program has been partially completed. The need for additional instructions in one part of a program often becomes apparent as another part is written. This sort of change is disastrous when the program has been written in absolute form.

More often than not, it is convenient to write the core of a program first, and then build out from that toward the beginning and end of the program. It is impossible in this case to know what absolute location each instruction will eventually occupy when the program is complete. But we do know that when the program is complete, successive instructions will occupy successive absolute locations. Hence we may adopt an arbitrary sequence of digits to stand for the location of the instruction. Call this the symbolic location. Then the only regularity demanded of the set of symbolic locations is that they be in order. No other restrictions need be placed on the symbolic locations used.

It is convenient, though not necessary, to adopt some convention as to the use of the sequence of digits making up the symbolic location. The coding usually falls into several logical sections. It is common practice to call them regions. The order of the instructions within a region is called the sequence number. Finally, at least one digit is needed for insertions between adjacent sequence numbers. Two decimal digits for region number gives the possibility of one hundred regions within a program. Up to one hundred instructions can be written in each region by using two more decimal digits for sequence number within a region. Finally, one decimal digit used for insertions gives the possibility of nine insertions between each instruction. Thus a total of five decimal digits used for symbolic location would seem ample for easy flexible programming.

Now since five decimal digits will convert to, at most, seventeen binary bits, a symbolic location will occupy one half word. This is important because the maximum size of the program which can be assembled depends directly on the space required for each symbolic location.

In general, the data required by a program fall into two classifications. There are the data required by the program every time it is executed. Call this permanent data. And there are the data which change each time the program is executed. Call this variable data. There will be data which fall in between the two classifications somewhere, but the following will still hold. In the interest of minimum loading time, the permanent data should probably be converted and punched on binary cards. This might also be done to variable data if it is going to be loaded more than once, but even if it is loaded from decimal cards, there should be as many numbers per card as possible. Either of these alternatives strongly suggests a compact data arrangement. This consists of a single area for a whole program with pieces of data assigned to consecutive locations within the area. The area itself can be assigned by the assembly program to the absolute locations immediately following the last instruction.

The need for freedom to make insertions in the instruction area all during the preparation of a program stems from the sequential way in which the machine executes the instructions. This is not the case with data, so no difficulty arises from assigning the data compactly within the data area.

A certain amount of erasable or temporary storage is also needed by the program during its execution. In general, a temporary location is used only briefly by the program and can be reused several times during the course of the program. As in the case of data, there is no programming objection to assigning the temporary locations compactly in a single temporary area. This facilitates an efficient use of a minimum number of temporary locations. The temporary area can be assigned by the assembly program to the absolute locations immediately following the data area.

The program has been broken, then, into three distinct areas in electrostatic storage—the instruction area, the data area, and the temporary area. Addresses referring to the instruction area will be

called symbolic type addresses. Those referring to the data area will be called data type, and those referring to the temporary area will be called temporary type. There is a fourth type of address which overlaps the other three. This is the absolute type by which reference is made directly to an absolute location anywhere in electrostatic storage or to an input-output unit, switch, light, etc.

Each of the four types of address requires different handling during the assembly process in order to correctly assign the corresponding absolute address. A symbolic address necessitates a search for the corresponding symbolic location. Since any instruction in a program may refer by way of its address to any other instruction in the program, it is necessary to have in electrostatic storage a file of all the symbolic locations used in the program being assembled. If each symbolic location requires only one half word in the file, the limit on the number of instructions which can be assembled at one time is equal to 4096 minus the number of half words required by the assembly program itself.

This file of symbolic locations can be built up by retaining in electrostatic storage each symbolic location as the instructions are read (to one card). The remaining information on each card can be written on tape so that the cards will not have to be read again. When all of the instructions have been read and the file of symbolic locations is complete, the instructions can be read back from tape one at a time, and the symbolic type addresses used as the arguments in a search of the file of symbolic locations. The absolute file location in which the correct symbolic location is found is linearly related to the absolute program location. In fact, since the symbolic locations each occupy a half word in the file, the difference between the file origin and the assigned program origin is of course the difference between any file location and the corresponding program location. This difference can be stored and applied against each absolute file location which results from the search of the symbolic locations to yield the program location as assembly proceeds.

At the time the instructions have all been read into the machine and the file of symbolic locations completed, the number of instructions in the program being assembled can be calculated. Now, if the programmer has specified the absolute location at which his program should begin and the number of half words of data used in the program, the data area origin and the temporary area origin can be calculated. The data origin becomes the first even address following the last instruction, and the temporary origin becomes the first even address following the last data location. Then the absolute equivalent of a data type address or a temporary type address can be calculated by adding the relative address to the respective origin. If the number of half words of temporary storage is also specified, a check can be made to be certain the program is not located too high in memory.

The three values which the programmer has been asked to specify, the program origin, the number of half words of data, and the number of half words of temporary, are written in absolute as the addresses of the last three instructions.



When the tape is read back record by record and each instruction is assembled into absolute form, immediate printing saves both time and storage space. Tape reading and symbolic searching can be completed quite easily between continuous print cycles. Since the program listing should be in direct order, the instructions have to come off the tape one by one in the same direct order. But it would be advantageous to avoid having to wait for the tape to rewind between writing and reading. This can be done if the instructions are written on the tape in reverse order and then the tape is read backwards. The cards can be sorted into reverse order as easily as into direct order. And the file of symbolic locations used can be set up starting at half word 4095 and working back. The last file location entered then becomes the file origin.

An alphabetic operation code is convenient to avoid having to memorize the absolute operation codes. It can be short and still be sufficiently mnemonic. It is desirable to have some kind of a check that the operation code written is allowable and that it has been key-punched correctly. The latter check demands some regularity in the codes; for example, that they all be two letter alphabetic codes. This is far from sufficient as a check, but it will catch a great many errors.

With regard to the first check, if the alphabetic codes are chosen so that the numeric underpunching is unique, the underpunching can be used as the argument in a table lookup operation to obtain the corresponding absolute code. Along with the absolute code, a test amount can be picked up to indicate that the underpunching is an allowable combination. This type of operation lookup also has the advantage of a readily accessible table if changes are desired. Additional operations for an interpretive scheme may be added for assembly.

The sign of an instruction is important when that instruction is to be modified, but otherwise it has a meaning which does not follow directly that of a full- or half-word reference. Many of the instructions are never modified, and when one is, the programmer's attention is directed to that instruction specifically. Consequently, the use of a 1 or a 2 instead of a sign in the symbolic instruction to mean one half word or two half words (a full word) is perhaps slightly more direct. This might not matter to the experienced programmer, but it may make a difference to the part-time man.

**CAPACITY:** Will assemble a program containing up to 3700 instructions.

**SPEED:** Requires approximately 850 milliseconds per instruction assembled. Most of this time is used in card reading and printing. Tape writing takes place between cards without slowing card reading. Then tape reading, assembly, and drum writing\* take place between printing the assembled instructions one per line at full printing speed. Finally

\*When the program being assembled contains more than 1850 instructions

drum reading\* and binary punching take place. This speed can almost be doubled if a listing of the assembled program is not required. Putting sense switch #5 down eliminates printing.

## COMPONENTS USED

Makes use of the card reader, all of electrostatic storage, tape #1 or tape #2 (under the control of sense switch #2), drum #4\*, the printer if a listing of the assembled program is desired, and the punch.

## CHECKS

The symbolic instructions are sequence checked as they are read. Any instruction out of sequence will result in a programmed stop.

The punch pattern on the cards is checked as the cards are read. Any punching which is contrary to the numeric-alphabetic pattern will result in a programmed stop.

The operation code is checked as the cards are read. Underpunching which does not match one of the standard codes will result in a programmed stop.

In these three cases, the offending card is the third one from the end if the cards are fed out. In the last two cases, running a corrected card followed by the remainder of the cards into the reader and pressing the start button will continue the process where it was interrupted. This is not possible in the case of the sequence error because the 701 tapes cannot be stepped back a unit record while they are in write status.

The number of instructions is checked during card reading, and a programmed stop will occur if an attempt is made to read in more than 3700 instructions.

A check sum is carried with each unit record on tape and will result in a programmed stop in the event of a tape error.

If printing is to take place, a check is made to be certain that the correct printer board is in. If ~~it is not~~, a programmed stop will occur. If the start button is pressed after the correct board has been put in, the assembly process will continue.

A check is made to be certain that every symbolic address has a corresponding symbolic location. If this is not the case, a programmed stop will occur. If the start is pressed, the assembly process will continue with an indication being printed beside the instruction with the bad address. This address check is applied only to those instructions having symbolic addresses.

\*When the program being assembled contains more than 1850 instructions

*The correct board  
is not in the  
machine*

The total amount and location of electrostatic storage used for instructions, data, and temporary storage is checked to be certain that the high order end of electrostatic storage is not exceeded. If it is, a programmed stop will occur.

In all of the above stops, one of the sense lights (see section on sense lights) is turned on to help identify the cause of the stop.

## INPUT

Input to the card reader consists of symbolic instructions punched one per card. See the section on machine assembly for the exact card columns. The principle reason for reading only one instruction per card is to maintain the ease of inserting or deleting single instructions.

The input must include three dummy instructions for the purpose of specifying three pieces of information about the program to be assembled. The three highest instructions must have as their address parts respectively the program origin, the number of half words of data, and finally the number of half words of temporary storage as shown in the following example:

99997	ST A1	2000	Program origin
99998	YE A1	48	Half words data
99999	YE A1	20	Half words temporary

## OUTPUT

Output consists of both printing and punching. The program is listed one instruction per line in both symbolic decimal and absolute octal. Remarks are printed to help in following the course of a program. The absolute is printed in octal rather than decimal because octal is the form most useful in using the console, memory printouts, etc.

The assembled program is binary punched forty-four instructions per card. The nine-left row of each card is used for control information for that card. Columns 10-14 contain the full word count on that card. Columns 15-26 contain the address for the first instruction on that card to read into. Columns 33-44, when other than zero, signal that the card is the last of a group and give the address to which control is to be transferred when the card has finished reading. The 12-right row, or the last half row punched if the card is not full, contains the check sum, a simple sum of all the other half rows on the card, including the 9-left control word.

## ASSEMBLED PROGRAM LOCATION:

There is no restriction on the area of electrostatic storage for which a program may be assembled. However, if a program origin is specified so high that instructions, data, and temporary will exceed the remainder of storage, an error stop will occur. (See the section on checks).

The instruction area begins at the program origin which has been specified by the programmer. The data origin is assigned by the assembly program to the first even location following the last instruction (the third of the three final dummy instructions). The origin of temporary storage is assigned by the assembly program to the first even location following the last data location. This has been determined by the programmer's specification of the number of half words of data.

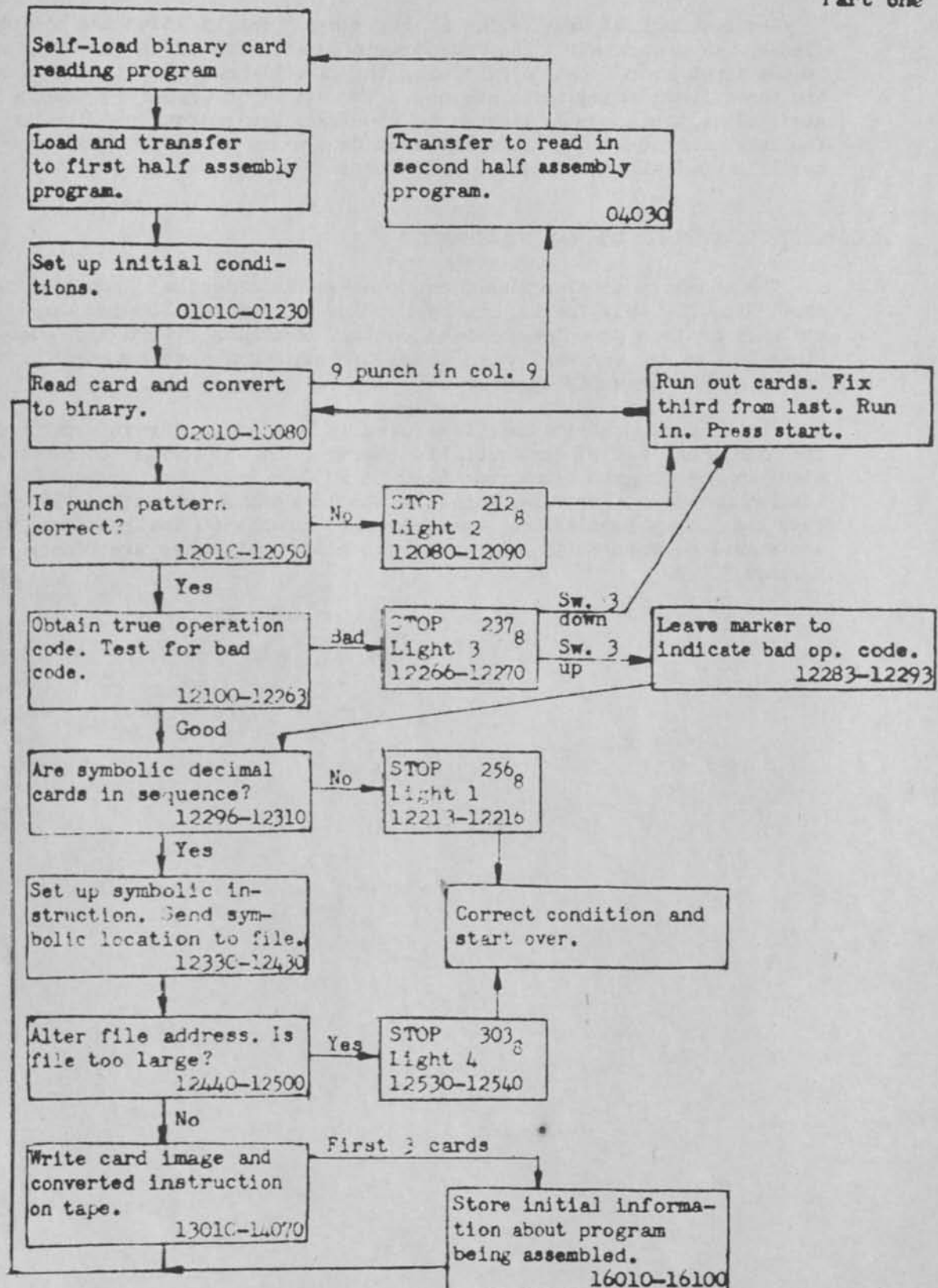
#### ELECTROSTATIC STORAGE USE DURING ASSEMBLY

The assembly program itself occupies absolute decimal locations 40 thru 395. The self-loading binary reading program for loading the assembly program occupies absolute decimal locations 0 thru 35. Locations 36 thru 39 are used as an absolute transition area between the two halves of the assembly program.

The file of symbolic locations used is located in direct order against the high order end of electrostatic storage. If the number of instructions in the program being assembled is 1850 or less, the assembled instructions are stored beginning in absolute decimal location 396 until they are binary punched. If there are more than 1850 instructions, they are stored on drum #4 in pairs after assembly until they are binary punched.

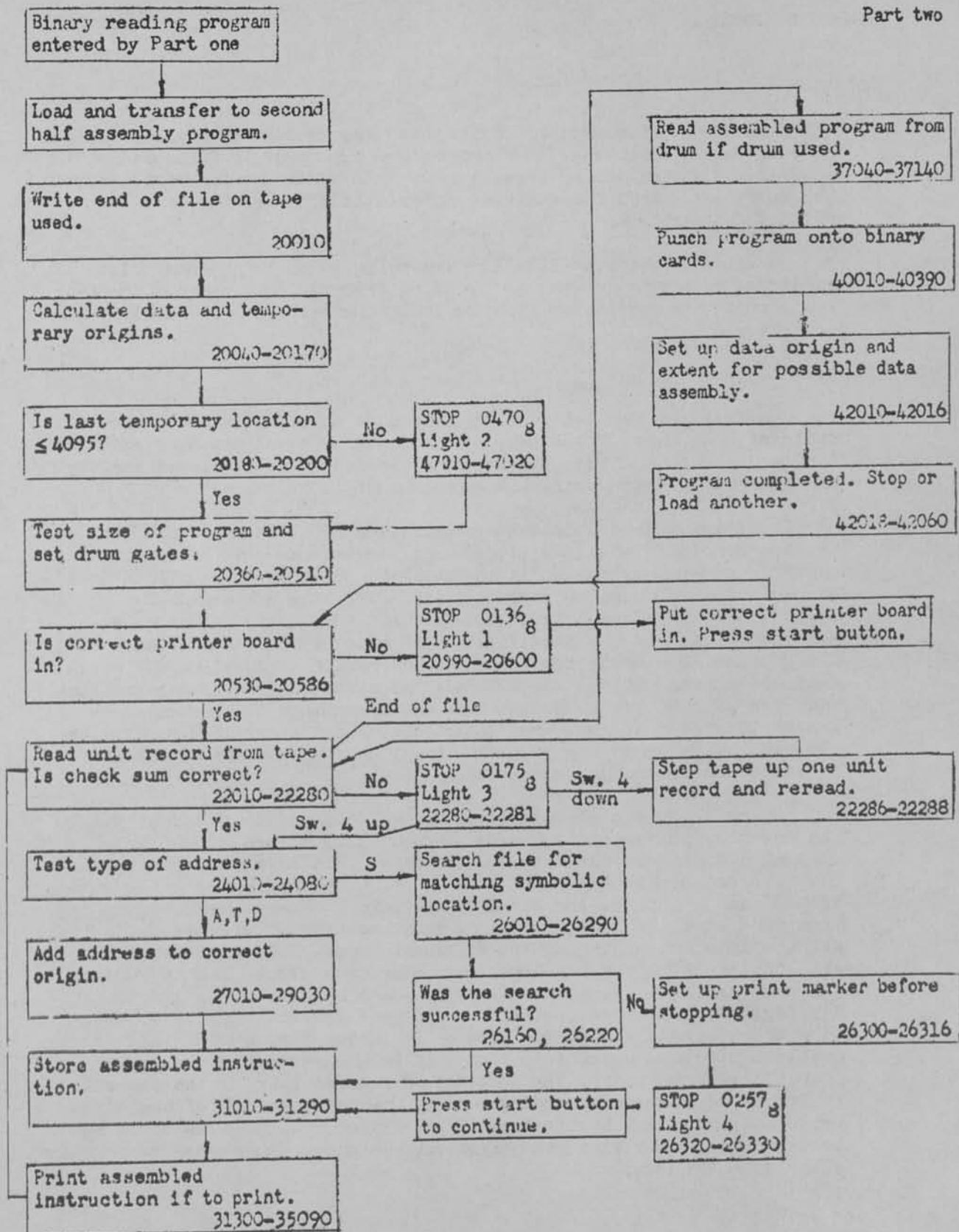
Block Diagram

Part one



Block Diagram

Part two



## LOCATION

Consists of five decimal digits which may be used in any way desired as long as the locations for a program are monotonically increasing. A suggested division of the five is to use the first two for region number, the third and fourth for sequence number within the region, and the fifth for insertions.

Consecutive absolute locations beginning with the program origin specified by the programmer as the third from the last instruction are assigned to the successive symbolic locations during the assembly process.

## OPERATION

Consists of a two-letter alphabetic code as shown in the attached table of operations. These have been chosen for their mnemonic value and for unique underpunching. The absolute operation codes are assigned by the assembly program during the execution of its first part.

There are two operations shown which are not standard. The first is ET for "Extract," This is a programming convenience only, since the assembly program assigns to it the absolute operation code of the Send Address order. ET must be used only with full-word addresses. The second is YE for "Your Entry." This is used in calling sequences when the address part is the specification of some absolute amount such as a magnitude. The reason for using this operation is found in the North American change routine. The YE operation part tells the change routine that the address part is an absolute amount and not to be changed as though it were an address when insertions or deletions of instructions are made elsewhere in the program. This in no way affects anyone who is not using the change routine.

Any or all of the alphabetic codes may be changed quite easily. The assembly program binary cards punched 0111201 thru 0111203 in columns 1 thru 7 are the data for the first part of the assembly program. The data origin for this first part is 354<sub>g</sub>. The operations table runs from D1 11 thru D1 99. The absolute equivalent of any alphabetic operation code should be in the relative data location D1 xx where xx is the alphabetic underpunching of the alphabetic code. This location should also contain 2525<sub>g</sub> for the legitimate operation check. In other words, the entries in the operations table are each a half word in length. The first or high order twelve bits are used for the legitimate operation check amount, 2525<sub>g</sub>. The last or low order five bits contain the absolute operation code. Note that this is the reverse of the usual operation-address order. The location of a given entry in the operations table can be computed by adding octally the underpunching of the alphabetic operation code to the base of the table, 354<sub>g</sub>. Any new codes must be two-letter codes since the punch pattern check tests these card columns along with the rest.

## ALPHABETIC OPERATION CODES

<u>S</u> top and <u>T</u> ransfer	ST
<u>U</u> nconditional <u>T</u> ransfer	UT
<u>O</u> verflow Transfer	OV
<u>P</u> lus Transfer	PT
<u>Z</u> ero Transfer	ZT
<u>S</u> ubtract	SU
<u>R</u> eset and <u>S</u> ubtract	RS
<u>S</u> ubtract as if <u>P</u> lus	SP
<u>N</u> o <u>I</u> nstruction	NI
<u>A</u> dd	AD
<u>R</u> eset and <u>A</u> dd	RA
<u>A</u> dd as if <u>P</u> lus	AP
<u>S</u> end <u>R</u> esult	SR
<u>S</u> end <u>A</u> ddress	SA
<u>S</u> end <u>M</u> Q	SQ
<u>L</u> oad <u>M</u> Q	LQ
<u>M</u> ultipl <u>Y</u>	MY
<u>M</u> ultipl <u>Y</u> and <u>R</u> ound	MR
<u>D</u> ivide	DV
<u>R</u> ou <u>N</u> d	RN
<u>L</u> ong <u>L</u> eft Shift	LL
<u>L</u> ong <u>R</u> ight Shift	LR
<u>A</u> ccumulator <u>L</u> eft Shift	AL
<u>A</u> ccumulator <u>R</u> ight Shift	AR
<u>R</u> ea <u>D</u>	RD
<u>R</u> ea <u>D</u> in <u>R</u> everse	RR
<u>W</u> ri <u>T</u> e	WR
<u>W</u> ri <u>T</u> e <u>E</u> nd of <u>F</u> ile	EF
<u>R</u> ewind	RW
<u>S</u> et <u>D</u> rum <u>A</u> ddress	DA
<u>S</u> ense and <u>S</u> kip or <u>C</u> ontrol	SS
<u>C</u> opy and <u>S</u> kip	CS
<u>E</u> xtract	ET
<u>Y</u> our <u>E</u> ntry	YE

## ALPHABETIC TYPE CODES

<u>A</u> bsolute	A
<u>S</u> ymbolic	S
<u>T</u> emporary	T
<u>D</u> ata	D



## TYPE

Consists of a two-column description of the address on that instruction. The first column is alphabetic and tells whether the address is written in absolute or is a reference to the symbolic area, data storage area, or temporary storage area. The letters for these are A, S, D, or T respectively.

The second column is a one if the address refers to a half-word location and a two if the address refers to a full-word location. This column is the equivalent of the sign when the instruction is in absolute form.

An advantage of using this type column is that the address column is then left completely free of arbitrary restrictions on the choice of regions.

## ADDRESS

Consists of five decimal digits. If the address is a reference to a symbolic location, the five decimal digits must be identical to the five decimal digits of the symbolic location referred to. An address with type "S" which does not have a corresponding symbolic location will result in an error stop.

If the address is a reference to a data or a temporary location, the absolute address will be obtained by adding the written address to the respective origin. The programmer usually assigns data or temporary locations compactly beginning with the respective origin, but this is not a requirement. When either or both of these areas have not been assigned compactly, special care should be taken that the specified number of half words of data or temporary covers both used and unused words in the area.

If the address is a reference to an absolute location, the type is "A" and the absolute address is exactly as written.

If the address is a modifiable address, it should be written as type "A" with brackets in the address columns. The brackets are then keypunched as a numeric X followed by four zeros. It is possible to include absolute, data, or temporary addresses within the brackets, but not symbolic addresses. A reference to full-word data location 2 (later to be modified) would be written D2 [ 2 ] and keypunched D2 x0002.

## REMARKS

Anything which can be keypunched and for which there are characters on the 716 printer can be written in the remarks column. Thirty-five card columns are reserved for these remarks. They are written onto and

read from the tape without check sum. Only the left side of the card image is included in the tape check sum.

An adequate description of each instruction written in the remarks column as the program is prepared will prove very useful when the program is being tested or changed. These remarks on the listing of the assembled program will be of assistance to the coder during the checkout period.

#### EXAMPLES

The attached form ED-622-3 serves as an example of the form used at North American for symbolic programming and contains some isolated instructions from the assembly program to illustrate the manner in which the symbolic decimal input is written.

The first instruction has symbolic location 02020 and is a Read operation with absolute address 2048, the address of the card reader.

Instruction 02070 is a Store operation in temporary full word 56. Instruction 08040 is a Transfer on Overflow to symbolic half-word location 08050. Instruction 08250 is a Subtract operation with the address data half-word location 1.

Instruction 12180 is a Divide by data full-word location 6. Instruction 12240 is a Reset and Add operation with address being a modifiable half word. Instruction 22120 is a Copy operation with address being a modifiable full word. Both of these modifiable addresses are initially setup by the program itself and altered by instructions in the same loop.

Finally, instruction 26150 is a Subtract temporary half-word location 58.

PROBLEM: EXAMPLES OF SYMBOLIC INPUT CHECKED BY: \_\_\_\_\_ DECK NO. \_\_\_\_\_ DATE: \_\_\_\_\_

JOB NO. \_\_\_\_\_ PAGE \_\_\_\_\_ OF \_\_\_\_\_

TRANSFER FROM	SYMBOLIC LOCATION	OP.	TYPE	ADDRESS	REMARKS (Limit to 35 columns)	BINARY MAGNITUDE
Do not key-punch	10	1415	17	19		80
	0 2 0 2 0	R D	A 1	2 0 1 8	PREPARE TO READ CARD	
	0 2 0 7 0	S R	T 2	5 6	CLEAR CK SUM COUNTER	
	0 6 0 4 0	O V	S 1	0 6 0 5 0	RESET OVERFLOW INDICATOR	
	0 6 2 5 0	S U	D 1	1	ALTER DOUBLE SUM ADDRESSES	
	1 2 1 6 0	D V	D 2	6	DIVIDE BY TEN	
	1 2 2 2 0	R A	A 1	7	OBTAIN OUR OPERATION CODE	
	2 2 1 2 0	C S	A 2	7	COPY LEFT WORD	
	2 6 1 5 0	S U	T 1	5 6	SUBTRACT NEW JONES	

## SYMBOLIC DECIMAL CARD FORM

See "SYMBOLIC INPUT" for more detailed description of symbolic instruction.

<u>Card column</u>	<u>Contents</u>
1-8	Indicative information if desired
9	Must be left blank
10-14	Symbolic location. Must have single numeric punch in each column.
15-16	Alphabetic operation code. Must have an alphabetic punch in both columns.
17	Alphabetic A, S, T, or D for type address. Must be alphabetic.
18	Numeric 1 or 2 for half or full-word address, respectively. Must be numeric.
19-23	Address. Must have a single numeric punch in each column. Numeric X in first column will give bracket. In this case, must <u>not</u> have digit punch also.
24-45	Should be left blank
46-80	Remarks. May be mixed alphabetic, numeric, or blank.

## CARD ORDER AND READER BOARD

The symbolic decimal cards must be sorted into reverse order on columns 10 thru 14, the symbolic location. This puts the three program specification cards first. The binary cards for the assembly program are then split into two parts. Cards 01 100 01 and 01 112 01 thru 01 112 08 are put on the front of the reversed symbolic decimal deck. Cards 01 112 09 thru 01 112 19 are put on the back of the reversed decimal deck. The whole deck is then ready for the card reader.

A standard reader board reading card columns 9 thru 80 straight into calculate entries left and right will read both the binary and the decimal cards correctly.

## SENSE SWITCHES

Switch #1: Not used by assembly program

- Switch #2: Tape selection switch  
 Up - Tape #1 used for intermediate storage  
 Down - Tape #2 used for intermediate storage
- Switch #3: Bad operation correction switch  
 Up - Pressing the start button after a non-allowable operation stop will cause the assembly process to continue after a marker has been left indicating that the instruction on which the stop occurred has a bad operation code.  
 Down - Pressing the start button after a non-allowable operation stop will cause the card on which the stop occurred to be reread. This assumes that the operator has run the cards under the brushes out, corrected the third from the last, and run it along with the following cards into the reader again.
- Switch #4: Tape reading error reread switch  
 Up - Pressing the start button after a tape check sum stop will cause the assembly process to continue after a marker has been left indicating that the instruction on which the stop occurred had a tape error.  
 Down - The unit record on which the stop occurred will be reread if the start button is pressed. If the original stop was caused by a reading error, a second attempt to read the unit record may very well be successful.
- Switch #5: Print switch  
 Up - A complete listing of the program being assembled will be printed out at the rate of 150 instructions per minute.  
 Down - No listing of the program being assembled will be printed.
- Switch #6: Program following switch  
 Up - When the assembly process has been completed and the binary cards punched, a program stop will occur.  
 Down - When the assembly process has been completed and the binary cards punched, the programmed equivalent of the load button will occur for the purpose of reading in any self-loading program which follows the assembly program.

## PROGRAMMED STOPS AND SENSE LIGHTS

For the purpose of being able to quickly ascertain the nature of a stop during the assembly process, one of the sense lights is turned on for each of the stops. The assembly process is broken into two parts, reading cards and printing or reading tape. The part being executed, plus one of the four lights will uniquely identify any one of the eight stops. The stops, associated lights, printing marker if there is one, and recommended course of action are shown below:

While reading cards:

<u>Octal Loc.</u>	<u>Light</u>	<u>Marker</u>	<u>Meaning and Course of Action</u>
0256	1	None	Decimal cards out of sequence. Put in sequence and restart from beginning.

## MACHINE ASSEMBLY (Continued)

17.

<u>Octal Loc.</u>	<u>Light</u>	<u>Marker</u>	<u>Meaning and Course of Action</u>
0212	2	None	Punch pattern bad. Run cards out, fix third from last, run it and following cards into reader, and press start button.
0237	3	8	Alphabetic operation code bad. Continue or reread bad card. See description of sense switch #3 use.
0303	4	None	File too large. Attempting to assemble more than 3700 instructions. Break up into blocks to assemble.

While printing or reading tape\*:

<u>Octal Loc.</u>	<u>Light</u>	<u>Marker</u>	<u>Meaning and Course of Action</u>
0136	1	None	Wrong printer board. Put correct board in and press start button.
0175	2	6	*Tape check sum error. Continue or reread unit record. See description of sense switch #4 use. Check punched binary card for error.
0470	3	None	*Upper limit of electrostatic storage exceeded. Reassign program origin and begin over again.
0257	4	7	*No such location in file for this symbolic address. Press start button to continue.

\* In the event one of the last three error stops occurs while assembling without printing (sense switch #5 down), put sense switch #5 up before pressing the start button and let one or two lines print before putting it down again. This will give a record of the instruction being assembled on which the stop occurred.

## PRINTER BOARD AND ALTERATION SWITCHES:

## Alteration Switch #1

Normal - Spacing across the printed sheet is wider and more open. Somewhat easier to read. Occupies approximately 8 inches.  
 Transferred - Spacing across the printed sheet is narrower in order to fit on a narrow form. Occupies approximately 7 inches.

## Alteration Switch #2

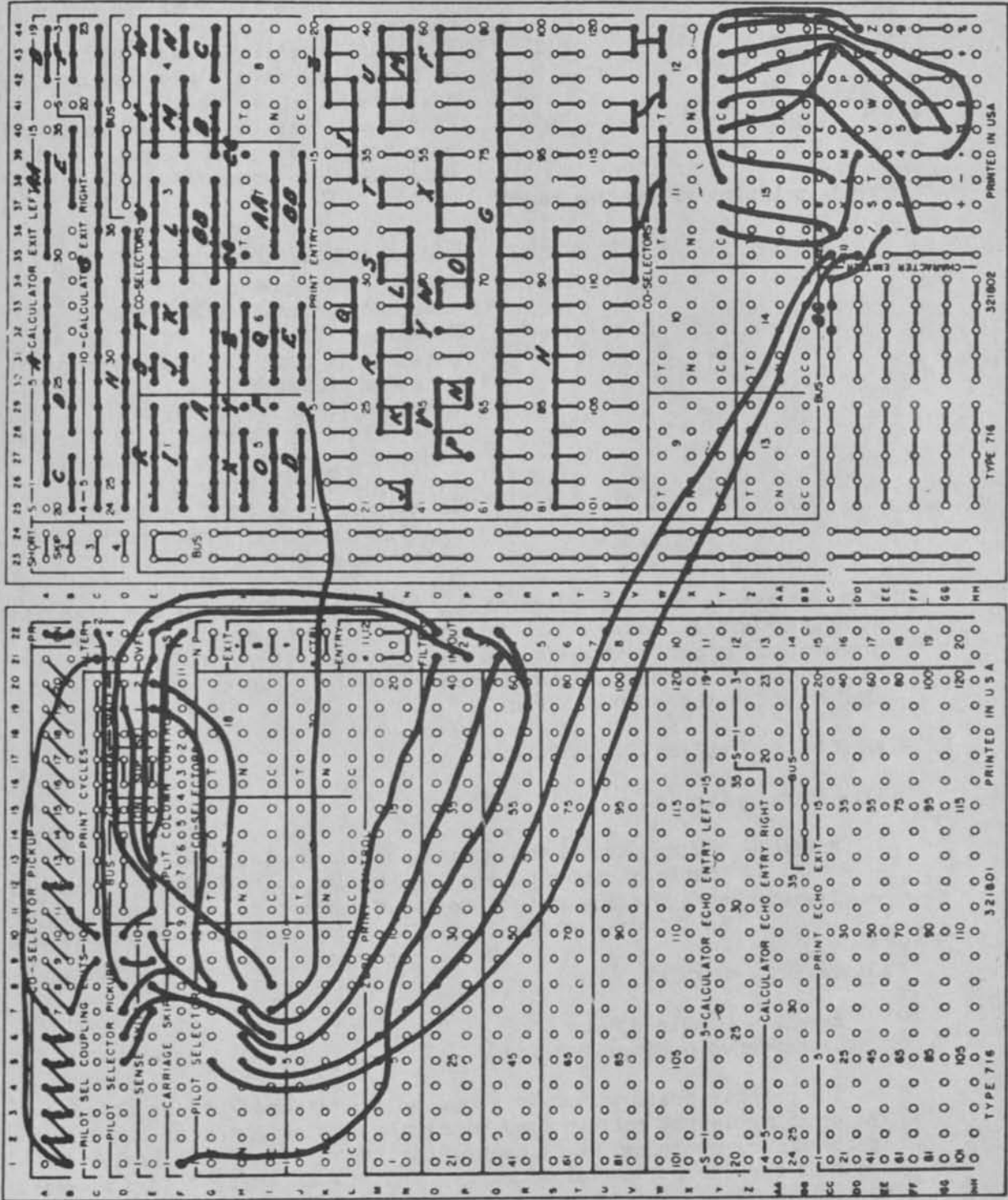
Normal - Double spacing  
 Transferred - Single spacing

The printer board wiring is shown on the attached wiring diagram. The printing is not echo checked since the critical part of the output is the binary punching. Zero control is not wired. The co-selectors are used for the horizontal spacing. The date is emitted thru a selector controlled by the carriage overflow impulse for printing on the first line of each sheet.

PRINTER BOARD WIRING

TYPE 716 PRINTER

Form 24-4137-0



PRINTED IN USA

321802

TYPE 716

PRINTED IN USA

321801

TYPE 716

## DESCRIPTION OF THE LISTING

The attached listing is the assembly program assembled by itself. Any program being assembled will list in this same form if sense switch #5 is up.

The symbolic decimal coding is printed to the left in exactly the same form as written. Brackets are printed as two slashes.

The absolute octal form of the program is printed in the center. Absolute octal was chosen rather than absolute decimal because of the necessity of using octal in connection with the console, memory print-outs, etc. The idea of printing both was discarded because of its tendency to confuse.

The remarks or description of the instruction is printed to the right exactly as punched.

If a marker has been left by one of the error stops, it will print to the left of the symbolic decimal on the line with the instruction in error.

## PUNCH PANEL

The punch panel is a standard seventy-two column board with calculate exits left and right being punched straight into card columns 9 thru 80.

## BINARY CARD FORM AND LOADING

The binary cards punched by the assembly program are in the following form. The nine left row is used as a control word and contains (1) the number of full words on the card, excluding the control word itself and the check sum, in columns 10-14, (2) the address into which the nine right word on the card is to read in columns 15-26, (3) the signal as to whether or not the card is the last of a group, and if it is, to what location control is to transfer when the card has been read. The last is in columns 33-44. It is zero on all except those cards on which a transfer of control away from the reading program is desired, i.e., the last card of a program.

The last half row punched (the 12 right row in the case of a full card) is the check sum for that card. It is the simple sum of all the other half rows on the card with the overflows disregarded. The nine left row is included in the check sum.

The binary cards punched may be loaded for execution with a copy of the first card of the first part of the assembly program. This card is identified with an 01 100 01 in columns 1 thru 7. It is a self-loading binary card-reading program which occupies and makes use of absolute octal locations 0 thru 43. When already in storage it can be entered



for the purpose of reading more binary cards by transferring to absolute location 0006. A check sum error while reading in binary cards will cause a stop at octal location 0036. The difference between the card check sum and the calculated check sum will show in the accumulator at the time of the stop. The card in error should be reread by running it into the reader and transferring to 0006<sub>g</sub> or by reloading card 01 100 01 ahead of it.

Any binary data cards being loaded should precede the program cards since the program will be executed as soon as the last program card has been read.

This binary card form was chosen because it was felt that it would lend itself to the most compact punching and reading programs while at the same time maintaining the advantage that each card is an independent unit record.

Symbolic Decimal			Absolute Octal			Remarks
Loc.	Op.	Type Addr.	Loc.	Op.	Addr.	
01010	RA	D1	00009	0050	+12 0365	SET STARTING ADDRESS FOR FILE
01020	SA	S1	12430	0051	+15 0272	X
01030	RA	S1	19001	0052	+12 0334	SET ENTRANCE TO INITIAL 3 REGION
01040	SA	S1	14070	0053	+15 0321	X
01050	RA	S1	19008	0054	+12 0343	SET SR ADDR IN INITIAL 3 REGION
01060	SA	S1	16020	0055	+15 0323	X
01080	RA	D1	00006	0056	+12 0362	CLEAR INFORMATION AREA
01090	SR	A2	00036	0057	-14 0044	X
01100	SR	A2	00038	0060	-14 0046	X
01110	RA	D1	00010	0061	+12 0366	SETUP FIRST VALUE FOR SEQUENCE CK
01130	SR	A1	00001	0062	+14 0001	X
01140	SS	A1	00070	0063	+36 0106	CHANGE TAPES IF 2 DOWN
01150	UT	S1	01230	0064	+01 0070	X
01160	RA	S1	19012	0065	+12 0347	X
01170	SA	S1	01230	0066	+15 0070	X
01180	SA	S1	12320	0067	+15 0257	X
01230	RW	A1	00256	0070	+34 0400	INSURE THAT TAPE IS REWOUND
02010	WR	A1	02052	0071	+32 4004	INSURE THAT MQ NOT IN USE
02020	RD	A1	02048	0072	+30 4000	PREPARE TO READ CARD
02030	RA	D1	00006	0073	+12 0362	CLEAR CARD CONVERSION COUNTERS
02040	SR	T2	00050	0074	-14 0602	X
02050	SR	T2	00052	0075	-14 0604	X
02060	SR	T2	00054	0076	-14 0606	X
02070	SR	T2	00056	0077	-14 0610	CLEAR CK SUM COUNTER
02080	RA	S1	19002	0100	+12 0335	RESET INITIAL COPY LOOP ADDRESS
02090	SA	S1	04010	0101	+15 0107	X
02095	SA	S1	04020	0102	+15 0110	X
02100	SA	S1	14010	0103	+15 0313	X
02110	RA	S1	19003	0104	+12 0336	RESET TRANSFER ADDRESS
02120	SA	S1	08180	0105	+15 0160	X
02130	SS	A1	00064	0106	+36 0100	TURN OFF SENSE LIGHTS
04010	CS	A2	/ /	0107	-37 0000	COPY LEFT WORD
04020	RS	A2	/ /	0110	-06 0000	TEST SIGN OF WORD JUST COPIED
04030	PT	A1	00006	0111	+03 0006	END OF FILE IF PLUS
04040	RA	S1	04010	0112	+12 0107	LEFT COPY ADDRESS
04050	SA	S1	05010	0113	+15 0133	ALTER LQ ADDRESS
04060	SA	S1	10010	0114	+15 0174	STORE LEFT WORD ADDRESSES
04070	SU	D1	00001	0115	+05 0355	FOR CK SUM
04080	SA	S1	10020	0116	+15 0175	X
04090	SU	D1	00001	0117	+05 0355	X
04100	SA	S1	04120	0120	+15 0121	ALTER RIGHT WORD COPY ADDRESS
04120	CS	A2	/ /	0121	-37 0000	COPY RIGHT WORD
04125	SU	D1	00002	0122	+05 0356	X
04130	SA	S1	04010	0123	+15 0107	ALTER LEFT WORD COPY ADDRESS
04135	SA	S1	04020	0124	+15 0110	X
04140	RS	S1	02040	0125	+06 0074	RESET ADDRESSES IN CARD
04150	SA	S1	08160	0126	+15 0156	CONVERSION LOOP
04160	SA	S1	08170	0127	+15 0157	X
04170	AD	D1	00001	0130	+11 0355	X
04180	SA	S1	08190	0131	+15 0161	X
04190	SA	S1	08200	0132	+15 0162	X
05010	LQ	A2	/ /	0133	-17 0000	SPACE LEFT ROW IMAGE

05020	LL	A1	00005	0134	+24	0005	X	
05030	AL	A1	00001	0135	+26	0001	X	
05040	LR	A1	00006	0136	+25	0006	X	
08010	RA	S1	04120	0137	+12	0121		TEST FOR ZERO ROW
08020	SU	S1	19004	0140	+05	0337	X	
08030	ZT	S1	08120	0141	+04	0152	X	
08040	OV	S1	08050	0142	+02	0143		RESET OVERFLOW INDICATOR
08050	RA	D1	00008	0143	+12	0364		SET END OF GROUP INDICATOR
08060	LL	A1	00001	0144	+24	0001		DIGIT INTO ACCUMULATOR
08070	OV	S2	08150	0145	-02	0155		LOOP END IF LAST DIGIT
08080	SR	T2	00048	0146	-14	0600		CONVERSION TO BINARY
08090	AL	A1	00002	0147	+26	0002	X	
08100	AD	T2	00048	0150	-11	0600	X	
08110	UT	S1	08060	0151	+01	0144		RETURN FOR NEXT DIGIT
08120	RA	S1	19005	0152	+12	0340		ALTER TRANSFER FORK ADDRESS
08130	SA	S1	08180	0153	+15	0160	X	
08140	UT	S1	08040	0154	+01	0142		RETURN TO CONVERSION
08150	AL	A1	00018	0155	+26	0022		POSITION CONVERTED GROUP
08160	AD	A1	/ /	0156	+11	0000		ADD SINGLE SUM CONVERSION CNTRS
08170	SR	A1	/ /	0157	+14	0000	X	
08180	UT	A1	/ /	0160	+01	0000		AVOID DBLF SUMS IF ROW 0 THRU 12
08190	AD	A1	/ /	0161	+11	0000		ADD DOUBLE SUM CONVERSION CNTRS
08200	SR	A1	/ /	0162	+14	0000	X	
08210	RS	S1	08170	0163	+06	0157		ALTER SINGLE SUM ADDRESSES
08220	SU	D1	00002	0164	+05	0356	X	
08230	SA	S1	08160	0165	+15	0156	X	
08240	SA	S1	08170	0166	+15	0157	X	
08250	SU	D1	00001	0167	+05	0355		ALTER DOUBLE SUM ADDRESSES
08260	SA	S1	08190	0170	+15	0161	X	
08270	SA	S1	08200	0171	+15	0162	X	
08280	SU	S1	02070	0172	+05	0077		LOOP END TEST
08290	PT	S2	08040	0173	-03	0142		CONVERT NEXT GROUP
10010	RS	A1	/ /	0174	+06	0000		MINUS LEFT HALF WORD
10020	SU	A1	/ /	0175	+05	0000		MINUS RIGHT HALF WORD
10030	AR	A1	00018	0176	+27	0022		
10040	AD	T2	00056	0177	-11	0610		PREVIOUS PARTIAL CK SUM
10050	SR	T2	00056	0200	-14	0610		STORE NEW PARTIAL CK SUM
10060	RS	S1	10010	0201	+06	0174		TEST FOR END OF CARD
10070	AD	S1	19006	0202	+11	0341	X	
10080	PT	S1	04010	0203	+03	0107	X	
12010	RA	T1	00050	0204	+12	0602		ADD SINGLE SUM CONVERSION CNTRS
12020	AD	T1	00052	0205	+11	0604	X	
12030	AD	T1	00054	0206	+11	0606	X	
12040	SU	D1	00003	0207	+05	0357		SUBTRACT CONVERSION TEST SUM
12050	ZT	S2	12100	0210	-04	0213		TRANSFER IF CORRECT
12080	SS	A1	00066	0211	+36	0102		PUNCH PATTERN ERROR STOP
12090	ST	S1	02010	0212	+00	0071	X	
12100	RA	D1	00006	0213	+12	0362		CLEAR HIGH ORDER OF CONVERTED
12110	SR	T1	00050	0214	+14	0602		SYMBOLIC OPERATION AND ADDR
12120	SR	T1	00052	0215	+14	0604	X	
12130	SR	T1	00054	0216	+14	0606	X	
12140	LQ	T2	00052	0217	-17	0604		OBTAIN SYM OPER AS DIVIDEND
12150	DV	D2	00006	0220	-22	0362		DIVIDE BY TEN

12160	SR	A2	00002	0221	-14	0002	FIRST REMAINDER IS SYM SIGN
12170	RA	D1	00006	0222	+12	0362	CLEAR ACCUMULATOR
12180	DV	D2	00006	0223	-22	0362	DIVIDE BY TEN
12190	SR	T2	00048	0224	-14	0600	SECOND REMAINDER IS TYPE
12200	SQ	A2	00004	0225	-16	0004	CODED OPERATION AS FULL WORD
12210	RA	A1	00005	0226	+12	0005	CODED OPERATION AS HALF WORD
12220	AD	S1	19009	0227	+11	0344	ADD BASE OF ASSMPLY DATA
12230	SA	S1	12240	0230	+15	0231	ADDRESS OF TRUE OPERATION CODE
12240	RA	A1	/ /	0231	+12	0000	OBTAIN TRUE OPERATION CODE
12245	LQ	D1	00006	0232	+17	0362	CLEAR MQ
12250	LR	A1	00023	0233	+25	0027	SHIFT OPERATION TO MQ
12260	SU	D2	00004	0234	-05	0360	SUBTRACT OPERATION CK CONSTANT
12263	ZT	S1	12296	0235	+04	0247	AVOID ERROR INDICATION IF OPERATION
12266	SS	A1	00067	0236	+36	0103	ERROR INDICATION NO SUCH OPERATION
12270	ST	S1	12273	0237	+00	0240	X
12273	SS	A1	00071	0240	+36	0107	CONTINUE OR REREAD BAD OPERATION
12276	UT	S1	12283	0241	+01	0243	CONTINUE
12280	UT	S1	02020	0242	+01	0072	REREAD BAD OPERATION CORRECTED
12283	RA	D1	00001	0243	+12	0355	LEAVE INDICATION OF BAD OPERATION
12286	AR	A1	00017	0244	+27	0021	X
12290	AD	T2	00004	0245	-11	0524	X
12293	SR	T2	00004	0246	-14	0524	X
12296	RA	T1	00051	0247	+12	0603	SEQUENCE CHECK OF LOCATIONS
12300	SU	A1	00001	0250	+05	0001	X
12303	PT	S1	12313	0251	+03	0255	SEQUENCE ERROR IF PLUS
12305	RA	T1	00051	0252	+12	0603	REPLACE LAST WITH CURRENT LOCATION
12307	SR	A1	00001	0253	+14	0001	X
12310	UT	S1	12320	0254	+01	0257	CONTINUE
12313	SS	A1	00065	0255	+36	0101	SEQUENCE ERROR
12316	ST	S1	01010	0256	+00	0050	X
12320	WR	A1	00256	0257	+32	0400	PREPARE TO WRITE ON TAPE
12330	LL	A1	00035	0260	+24	0043	SHIFT OPERATION BACK TO ACCUM
12340	AD	T1	00049	0261	+11	0601	ADD TYPE OF ADDRESS
12350	AD	T2	00054	0262	-11	0606	ADD SYM ADDR TO OPER AND TYPE
12360	SR	T2	00048	0263	-14	0600	STORE OP AND TYPE AND ADDR
12370	RA	A2	00002	0264	-12	0002	SYMBOLIC SIGN
12380	AR	A1	00001	0265	+27	0001	SHIFT FOR 1 OR 2 ODD OR EVEN
12390	ZT	S1	12420	0266	+04	0271	AVOID MINUS ACTION
12400	RS	T2	00048	0267	-06	0600	ATTACH MINUS SIGN IF NEGATIVE
12410	SR	T2	00048	0270	-14	0600	X
12420	RA	T1	00051	0271	+12	0603	SYMBOLIC LOCATION
12430	SR	A1	/ /	0272	+14	0000	STORE IN FILE
12440	RA	S1	12430	0273	+12	0272	FILE ADDRESS
12450	SA	A1	00039	0274	+15	0047	LAST FILE ADDRESS LEFT HERE
12460	SA	T1	00050	0275	+15	0602	FILE ADDRESS FOR TAPE WRITING
12470	SU	D1	00001	0276	+05	0355	ALTER FILE ADDRESS
12480	SA	S1	12430	0277	+15	0272	X
12490	SU	S1	19013	0300	+05	0350	SUBTRACT FILE LOWER LIMIT
12500	PT	S2	13010	0301	-03	0304	AVOID ERROR INDICATION
12530	SS	A1	00068	0302	+36	0104	ERROR FILE TOO LARGE
12540	ST	S1	01010	0303	+00	0050	X
13010	RS	T1	00048	0304	+06	0600	COMPLETE CHECK SUM
13020	SU	T1	00049	0305	+05	0601	X

13030	SU	T1	00050	0306	+05	0602	X
13040	SU	T1	00051	0307	+05	0603	X
13050	AR	A1	00018	0310	+27	0022	X
13060	AD	T2	00056	0311	-11	0610	X
13070	SR	T2	00052	0312	-14	0604	X
14010	CS	A2	/ /	0313	-37	0000	COPY WORD
14020	RA	S1	14010	0314	+12	0313	ALTER COPY ADDRESS
14030	SU	D1	00002	0315	+05	0356	X
14040	SA	S1	14010	0316	+15	0313	X
14050	SU	S1	19010	0317	+05	0345	CHECK FOR END OF LOOP
14060	PT	S1	14010	0320	+03	0313	REMAIN IN LOOP IF PLUS
14070	UT	A1	/ /	0321	+01	0000	FIRST THREE CARD GATE
16010	RA	T1	00049	0322	+12	0601	PLACE INITIAL INFORMATION
16020	SR	A1	/ /	0323	+14	0000	X
16030	RA	S1	16020	0324	+12	0323	ALTER SR ADDRESS
16040	SU	D1	00001	0325	+05	0355	X
16050	SA	S1	16020	0326	+15	0323	X
16060	SU	S1	19011	0327	+05	0346	TEST FOR LAST INITIAL SPEC
16070	PT	S1	02010	0330	+03	0071	RETURN TO NEXT CARD IF PLUS
16080	RA	S1	16100	0331	+12	0333	STOP USE OF REGION 16
16090	SA	S1	14070	0332	+15	0321	X
16100	UT	S2	02010	0333	-01	0071	RETURN TO READ NEXT CARD
19001	ST	S1	16010	0334	+00	0322	ENTRANCE TO INITIAL 3 REGION
19002	ST	T1	00000	0335	+00	0520	9 LEFT ROW IMAGE ADDRESS
19003	ST	S1	08190	0336	+00	0161	DOUBLE SUM ADDITION INSTRUCTION
19004	CS	T2	00038	0337	-37	0566	0 RIGHT ROW IMAGE ADDRESS
19005	ST	S1	08210	0340	+00	0163	AVOID DOUBLE SUM TRANSFER ADDR
19006	RS	T1	00044	0341	+06	0574	END OF CARD TEST ADDRESS
19007	CS	T2	00048	0342	-37	0600	CARD IMAGE END TEST ADDRESS
19008	ST	A1	00038	0343	+00	0046	FIRST SR ADDRESS IN INITIAL 3
19009	ST	D1	00000	0344	+00	0354	BASE OF DATA REGION
19010	CS	T2	00054	0345	-37	0606	TAPE WRITING LOOP END TEST
19011	SR	A1	00036	0346	+14	0044	LAST INITIAL 3 TEST WORD
19012	RW	A1	00257	0347	+34	0401	TAPE 2 ADDRESS
19013	SR	A1	00396	0350	+14	0614	FILE LOWER LIMIT TEST
19997	ST	A1	00040	0351	+00	0050	PROGRAM ORIGIN
19998	ST	A1	00100	0352	+00	0144	NUMBER HALF WORDS DATA
19999	ST	A1	00058	0353	+00	0072	NUMBER HALF WORDS TEMPORARY

DATA

<u>Location</u>	<u>Contents</u>	<u>Binary Magnitude</u>
D1 0	4096	17
D1 1	1	17
D1 2	2	17
D1 3	Conversion Test Sum	17
D2 4	Operation Test Constant	35
D1 6	0	17
D1 7	10	17
D1 8	8	17
D1 9	4095	17
D1 10	131071	17
D1 11	Operation Table	
.		
.		
.		
D1 99		

TEMPORARY

<u>Location</u>	<u>Use</u>
T2 0	Card Image
.	
.	
T2 46	Conversion Working Storage Address Type Operation, Type, and Address
T2 48	
T1 50	Conversion Summing Locations
.	
.	
T1 55	Partial Check Sum
T2 56	

20001	SS	A1	00070	0050	+36	0106	CHANGE TAPES IF 2 DOWN
20002	UT	S1	20010	0051	+01	0056	X
20003	RA	S1	49010	0052	+12	0501	X
20004	SA	S1	20010	0053	+15	0056	X
20005	SA	S1	22010	0054	+15	0140	X
20006	SA	S1	22286	0055	+15	0200	X
20010	EF	A1	00256	0056	+33	0400	WRITE END OF FILE ON TAPE
20020	WR	A1	00512	0057	+32	1000	PREPARE TO IDENTIFY PRINTER BOARD
20030	SS	A1	00521	0060	+36	1011	X
20040	RA	A1	00036	0061	+12	0044	PROGRAM LOCATION MINUS FILE LOCATIO
20050	SU	A1	00039	0062	+05	0047	X
20060	SR	T1	00054	0063	+14	0604	X
20080	AD	D1	00003	0064	+11	0511	ADD 4096
20090	AD	D1	00001	0065	+11	0507	ADD ONE TO CARRY IF ODD
20100	AR	A1	00019	0066	+27	0023	FORCE LOW ORDER BIT TO ZERO
20110	AL	A1	00019	0067	+26	0023	X
20120	SR	T1	00055	0070	+14	0605	ORIGIN DATA STORAGE
20130	AD	A1	00037	0071	+11	0045	ADD NUMBER HALF WORDS DATA
20140	AD	D1	00001	0072	+11	0507	ADD ONE TO CARRY IF ODD
20150	AR	A1	00019	0073	+27	0023	FORCE LOW ORDER BIT TO ZERO
20160	AL	A1	00019	0074	+26	0023	X
20170	SR	T1	00056	0075	+14	0606	ORIGIN TEMPORARY STORAGE
20180	AD	A1	00038	0076	+11	0046	ADD NUMBER HALF WORDS TEMPORARY
20190	SU	D1	00007	0077	+05	0515	SUBTRACT 4097
20200	PT	S1	47010	0100	+03	0467	ELECTROSTATIC EXCEEDED IF PLUS
20310	RA	A1	00039	0101	+12	0047	OBTAIN THE LOWER SEARCH EXTREME
20320	SU	D1	00001	0102	+05	0507	X
20330	SR	T1	00057	0103	+14	0607	X
20340	RA	D1	00000	0104	+12	0506	SET FIRST DA ADDRESS TO ZERO
20350	SA	S1	31230	0105	+15	0311	X
20360	RA	T1	00055	0106	+12	0605	OBTAIN NUMBER HALF WORDS IN PROG
20370	SU	A1	00036	0107	+05	0044	X
20380	SR	T1	00059	0110	+14	0611	X
20390	SU	D1	00006	0111	+05	0514	TEST SIZE OF PROGRAM
20400	PT	S1	20450	0112	+03	0117	LARGE PROGRAM IF PLUS
20410	RA	S1	31180	0113	+12	0305	SET DRUM GATES FOR SMALL PROGRAM
20420	SA	S1	31150	0114	+15	0302	X
20430	RA	S1	49009	0115	+12	0500	X
20440	UT	S1	20480	0116	+01	0122	X
20450	RA	S1	49007	0117	+12	0476	SET DRUM GATES FOR LARGE PROGRAM
20460	SA	S1	31150	0120	+15	0302	X
20470	RA	S1	49008	0121	+12	0477	X
20480	SA	S1	37030	0122	+15	0370	X
20490	RA	S1	31270	0123	+12	0315	SET DRUM READ TEST ADDRESS
20500	SU	T1	00059	0124	+05	0611	X
20510	SA	S1	49006	0125	+15	0475	X
20530	SS	A1	00073	0126	+36	0111	SKIPS IF NO PRINTING DESIRED
20535	UT	S2	20550	0127	-01	0131	TRANSFER IF PRINTING DESIRED
20540	UT	S2	20610	0130	-01	0137	NO PRINTING DESIRED
20550	SS	A1	00065	0131	+36	0101	TURN ON SENSE LITES
20584	MY	A1	00000	0132	+20	0000	ALLOW TIME FOR SELECTOR PICKUP
20585	MY	A1	00000	0133	+20	0000	X

20586	MY	A1	00000	0134	+20	0000	X
20590	SS	A1	00522	0135	+36	1012	IS CORRECT PRINTER BOARD IN
20600	ST	S1	20030	0136	+00	0060	STOP IF NOT
20610	SS	A1	00064	0137	+36	0100	TURN OFF SENSE LITES IF SO
22010	RR	A1	00256	0140	+31	0400	PREPARE TO READ IN REVERSE
22020	CS	T2	00052	0141	-37	0602	COPY TAPE CHECK SUM
22030	UT	S1	22050	0142	+01	0144	AVOID END OF FILE OUT
22040	UT	S2	37010	0143	-01	0366	END OF FILE OUT
22050	CS	T2	00050	0144	-37	0600	COPY WORD
22060	RA	S1	22050	0145	+12	0144	SET FIRST ADDRESS OF COPY LOOP
22070	AD	D1	00002	0146	+11	0510	ALTER COPY ADDRESS
22080	SA	S1	22120	0147	+15	0153	X
22090	SA	S1	22130	0150	+15	0154	STORE ADDRESS FOR CK SUM
22100	SU	D1	00001	0151	+05	0507	X
22110	SA	S1	22140	0152	+15	0155	X
22120	CS	A2	/ /	0153	-37	0000	COPY LEFT WORD
22130	RA	A1	/ /	0154	+12	0000	LEFT WORD AGAINST CK SUM
22140	AD	A1	/ /	0155	+11	0000	X
22150	AR	A1	00018	0156	+27	0022	X
22160	AD	T2	00052	0157	-11	0602	X
22170	SR	T2	00052	0160	-14	0602	X
22180	RA	S1	22120	0161	+12	0153	SET RIGHT COPY ADDRESS
22190	AD	D1	00002	0162	+11	0510	X
22200	SA	S1	22210	0163	+15	0164	X
22210	CS	A2	/ /	0164	-37	0000	COPY RIGHT WORD
22220	UT	S1	22070	0165	+01	0146	CONTINUE IN LOOP
22230	SR	A1	00396	0166	+14	0614	INSTRUCTIONAL CONSTANT
22240	RA	T1	00050	0167	+12	0600	COMPLETE CK SUM CHECK
22250	AD	T1	00051	0170	+11	0601	X
22260	AR	A1	00018	0171	+27	0022	X
22270	AD	T2	00052	0172	-11	0602	X
22275	ZT	S1	24010	0173	+04	0206	ZERO TRANSFER IF CORRECT
22280	SS	A1	00067	0174	+36	0103	TURN ON LITE 3 FOR TAPE ERROR
22281	ST	S1	22282	0175	+00	0176	ERROR STOP TAPE ERROR
22282	SS	A1	00072	0176	+36	0110	REREAD OR GO ON
22284	UT	S1	22290	0177	+01	0202	LEAVE MARKER AND GO ON
22286	RD	A1	00256	0200	+30	0400	STEP BACK
22288	UT	S1	20610	0201	+01	0137	REREAD
22290	RA	D1	00001	0202	+12	0507	ERROR INDICATION TAPE ERROR
22300	AR	A1	00015	0203	+27	0017	X
22303	AD	T2	00012	0204	-11	0532	X
22306	SR	T2	00012	0205	-14	0532	X
24010	RA	S1	24030	0206	+12	0210	ORIGIN OF TRANSFER REGION
24020	AP	T1	00048	0207	+13	0576	OPERATION AND TYPE
24030	SA	S1	24040	0210	+15	0211	STORE TRANSFER FORK ADDRESS
24040	UT	A1	/ /	0211	+01	0000	TRANSFER TO TYPE TRANSFER
24050	UT	S2	29010	0212	-01	0264	TRANSFER FOR ABSOLUTE TYPE
24060	UT	S2	26010	0213	-01	0216	TRANSFER FOR SYMBOLIC TYPE
24070	UT	S2	27010	0214	-01	0260	TRANSFER FOR TEMPORARY TYPE
24080	UT	S2	28010	0215	-01	0262	TRANSFER FOR DATA TYPE
26010	RA	T1	00057	0216	+12	0607	OBTAIN LOWER EXTREME
26020	SR	T1	00052	0217	+14	0602	STORE AS WORKING LOWER
26030	RA	D1	00003	0220	+12	0511	OBTAIN UPPER EXTREME



26040	SR	T1	00053	0221	+14	0603	STORE AS WORKING UPPER
26050	RA	T1	00052	0222	+12	0602	BEGINNING OF SEARCH LOOP
26060	AD	T1	00053	0223	+11	0603	SUM OF UPPER AND LOWER EXTREMES
26070	AR	A1	00001	0224	+27	0001	MEAN OF UPPER AND LOWER EXTREMES
26080	SR	T1	00058	0225	+14	0610	STORE MEAN
26090	SA	S1	26100	0226	+15	0227	STORE MEAN AS ADDRESS
26100	RA	A1	/ /	0227	+12	0000	CONTENTS MEAN LOCATION
26110	SU	T1	00049	0230	+05	0577	SUBTRACT SYMBOLIC ADDRESS
26120	ZT	S1	26260	0231	+04	0247	OUT OF LOOP IF SEARCH COMPLETED
26130	PT	S1	26200	0232	+03	0241	AVOID MINUS ACTION
26140	RA	T1	00052	0233	+12	0602	RESET ADD OLD LOWER
26150	SU	T1	00058	0234	+05	0610	SUBTRACT NEW LOWER
26160	PT	S1	26300	0235	+03	0252	ERROR NO SUCH SYMBOLIC ADDRESS
26170	RA	T1	00058	0236	+12	0610	MEAN BECOMES NEW LOWER
26180	SR	T1	00052	0237	+14	0602	STORE NEW LOWER
26190	UT	S1	26060	0240	+01	0223	RETURN FOR NEXT TRIAL
26200	RA	T1	00058	0241	+12	0610	RESET ADD THE MEAN
26210	SU	T1	00053	0242	+05	0603	SUBTRACT OLD UPPER
26220	PT	S1	26300	0243	+03	0252	ERROR NO SUCH SYMBOLIC ADDRESS
26230	RA	T1	00058	0244	+12	0610	RESET ADD THE MEAN
26240	SR	T1	00053	0245	+14	0603	STORE NEW UPPER
26250	UT	S1	26050	0246	+01	0222	RETURN FOR NEXT TRIAL
26260	RA	S1	26100	0247	+12	0227	RESET ADD FINAL SEARCH LOCATION
26270	AD	T1	00054	0250	+11	0604	OBTAIN ACTUAL ADDRESS
26290	UT	S2	29030	0251	-01	0266	END OF INSTRUCTION ASSEMBLY
26300	RA	D1	00001	0252	+12	0507	NO SUCH SYMBOLIC ADDRESS
26310	AR	A1	00016	0253	+27	0020	X
26313	AD	T2	00008	0254	-11	0526	X
26316	SR	T2	00008	0255	-14	0526	X
26320	SS	A1	00068	0256	+36	0104	X
26330	ST	S1	31010	0257	+00	0267	X
27010	RA	T1	00056	0260	+12	0606	ORIGIN OF TEMPORARY
27020	UT	S1	29020	0261	+01	0265	X
28010	RA	T1	00055	0262	+12	0605	ORIGIN OF DATA
28020	UT	S1	29020	0263	+01	0265	X
29010	RA	D1	00000	0264	+12	0506	ZERO
29020	AD	T1	00049	0265	+11	0577	ADD SYMBOLIC ADDRESS
29030	SA	T1	00048	0266	+15	0576	STORE ACTUAL ADDRESS
31010	RA	T1	00050	0267	+12	0600	FILE LOCATION
31020	AD	T1	00054	0270	+11	0604	OBTAIN ACTUAL LOCATION
31030	SR	T1	00050	0271	+14	0600	REPLACE FILE LOC WITH ACTUAL LOC
31040	RA	T1	00049	0272	+12	0576	ACTUAL INSTRUCTION
31050	SR	A1	00394	0273	+14	0612	STORE IN TEMPORARY LOCATION
31060	RA	S1	31050	0274	+12	0273	ALTER TEMPORARY LOCATION BY 1
31070	AD	D1	00001	0275	+11	0507	X
31080	SA	S1	31050	0276	+15	0273	X
31081	SA	S1	31083	0277	+15	0301	NEXT TEMPORARY LOCATION
31082	RA	D1	00000	0300	+12	0506	ZERO
31083	SR	A1	/ /	0301	+14	0000	CLEAR NEXT TEMPORARY LOCATION
31150	UT	A1	/ /	0302	+01	0000	PRESET DRUM GATE
31160	RS	S1	31050	0303	+06	0273	TEMPORARY LOCATION
31170	AD	S1	22230	0304	+11	0166	TEMPORARY TEST LOCATION
31180	PT	S1	31300	0305	+03	0320	TRANSFER IF PAIR NOT READY

31190	WR	A1	00131	0306	+32	0203	PREPARE TO WRITE ON DRUM
31200	RA	S1	31270	0307	+12	0315	RESET TEMPORARY PAIR ADDRESS
31220	SA	S1	31050	0310	+15	0273	X
31230	DA	A2	/ /	0311	-35	0000	SET DRUM ADDRESS
31240	RA	S1	31230	0312	+12	0311	MODIFY DRUM ADDRESS
31250	SU	D1	00002	0313	+05	0510	X
31260	SA	S1	31230	0314	+15	0311	X
31270	CS	A2	00394	0315	-37	0612	COPY WORD
31280	RA	D1	00000	0316	+12	0506	RESET PAIR LOCATION TO ZERO
31290	SR	A2	00394	0317	-14	0612	X
31300	SS	A1	00073	0320	+36	0111	SKIPS IF NO PRINTING DESIRED
31305	UT	S2	31320	0321	-01	0323	PRINTING DESIRED
31310	UT	S1	20610	0322	+01	0137	NO PRINTING DESIRED
31320	WR	A1	00512	0323	+32	1000	PREPARE TO WRITE PRINTER
31330	RA	T1	00040	0324	+12	0566	11 LEFT ROW IMAGE
31340	AL	A1	00011	0325	+26	0013	X
31350	ZT	S1	33010	0326	+04	0330	TRANSFER IF NO X
31360	SS	A1	00520	0327	+36	1010	PICKUP PRINT SELEC FOR PAREN
33010	RA	D1	00002	0330	+12	0510	COLUMN INDICATOR
33020	SR	T2	00052	0331	-14	0602	X
33025	RA	T1	00048	0332	+12	0576	ASSEMBLED INSTRUCTION
33030	PT	S1	33040	0333	+03	0335	AVOID PICKING SELECTOR IF PLUS
33035	SS	A1	00518	0334	+36	1006	PICK FOR MINUS
33040	LR	A1	00035	0335	+25	0043	SHIFT TO MQ
33050	RA	T1	00050	0336	+12	0600	ACTUAL LOCATION
33060	AR	A1	00017	0337	+27	0021	X
33070	LR	A1	00013	0340	+25	0015	SHIFT INTO MQ
33080	LL	A1	00003	0341	+24	0003	BEGIN CONVERSION LOOP
33090	AL	A1	00020	0342	+26	0024	FOUR TIMES DIGIT EQUIV TO ADDR
33100	AD	S1	49001	0343	+11	0471	ADD CARD IMAGE BASE
33110	SA	S1	33160	0344	+15	0351	CORRECT ROW IMAGE ADDRESS
33120	SA	S1	33170	0345	+15	0352	X
33130	RA	T2	00052	0346	-12	0602	ALTER COLUMN INDICATOR
33140	AR	A1	00001	0347	+27	0001	X
33150	SR	T2	00052	0350	-14	0602	X
33160	AD	A2	/ /	0351	-11	0000	ADD CORRECT ROW IMAGE
33170	SR	A2	/ /	0352	-14	0000	STORE IN CORRECT ROW IMAGE
33180	AL	A1	00027	0353	+26	0033	TEST FOR END OF CONVERSION
33190	ZT	S1	33080	0354	+04	0341	REMAIN IN LOOP IF ZERO
35010	RA	S1	49002	0355	+12	0472	SET UP FIRST COPY ADDRESS
35020	SA	S1	35030	0356	+15	0357	X
35030	CS	A2	/ /	0357	-37	0000	COPY WORD
35040	RA	S1	35030	0360	+12	0357	ALTER COPY ADDRESS
35050	SU	D1	00002	0361	+05	0510	X
35060	SA	S1	35030	0362	+15	0357	X
35070	SU	S1	49003	0363	+05	0473	TEST FOR END OF LOOP
35080	PT	S1	35030	0364	+03	0357	REMAIN IN LOOP
35090	UT	S1	20610	0365	+01	0137	TRANSFER TO NEXT TAPE READ
37010	WR	A1	00512	0366	+32	1000	EJECT LAST PAGE IN PRINTER
37020	SS	A1	00519	0367	+36	1007	EJECT LAST PAGE IN PRINTER
37030	UT	A1	/ /	0370	+01	0000	PRESET DRUM GATE
37040	RA	S1	49005	0371	+12	0474	WRITE LAST WORD ON DRUM
37050	SR	S1	31300	0372	+14	0320	X

37060	UT	S1	31190	0373	+01	0306	X	
37070	RD	A1	00131	0374	+30	0203		READ RECORD FROM DRUM
37080	DA	A2	00000	0375	-35	0000	X	
37090	CS	A2	00394	0376	-37	0612	X	
37100	RA	S1	37090	0377	+12	0376	X	
37110	SU	D1	00002	0400	+05	0510	X	
37120	SA	S1	37090	0401	+15	0376	X	
37130	SU	S1	49006	0402	+05	0475	X	
37140	PT	S1	37090	0403	+03	0376	X	
40010	RA	D1	00005	0404	+12	0513		HALF AND FULL WORD COUNT FOR
40020	SR	T1	00053	0405	+14	0603		FIRST CARD
40030	AL	A1	00011	0406	+26	0013	X	
40040	AD	A1	00036	0407	+11	0044		1ST ADDRESS FOR 1ST CARD INTO
40050	SR	T2	00000	0410	-14	0516	X	
40060	RA	T1	00059	0411	+12	0611		BEGINNING OF PUNCH LOOP
40070	ZT	S2	42010	0412	-04	0456		OUT IF PUNCHING FINISHED
40080	WR	A1	01024	0413	+32	2000		PREPARE TO PUNCH
40090	SU	D1	00005	0414	+05	0513		SUBTRACT 44
40100	ZT	S1	40120	0415	+04	0417		TO LAST CARD ACTION
40110	PT	S1	40210	0416	+03	0431		AVOID LAST CARD ACTION
40120	RA	A1	00036	0417	+12	0044		FIRST TO EXELX
40130	AR	A1	00018	0420	+27	0022	X	
40140	AD	T1	00000	0421	+11	0516		FIRST TO READ INTO
40150	LR	A1	00030	0422	+25	0036	X	
40160	RA	T1	00059	0423	+12	0611		HALF AND FULL WORD COUNT FOR LAST
40170	SR	T1	00053	0424	+14	0603	X	
40180	AR	A1	00019	0425	+27	0023	X	
40190	LR	A1	00005	0426	+25	0005	X	
40195	SQ	T2	00000	0427	-16	0516		STORE CONTROL WORD FOR LAST CARD
40200	RA	D1	00000	0430	+12	0506		CLEAR ACCUMULATOR
40210	SR	T1	00059	0431	+14	0611		NUMBER HALF WORDS LEFT
40220	CS	T2	00000	0432	-37	0516		COPY 9 LEFT ROW
40230	RA	T2	00000	0433	-12	0516		START CK SUM
40240	SR	T2	00002	0434	-14	0520	X	
40250	RS	S1	40290	0435	+06	0441		SETUP TEST LOOP WORD
40260	AD	T1	00053	0436	+11	0603	X	
40270	SR	T1	00052	0437	+14	0602	X	
40280	RA	T2	00002	0440	-12	0520		PARTIAL CK SUM
40290	CS	A2	00394	0441	-37	0612		COPY WORD
40300	AD	A2	00394	0442	-11	0612		ADD TO CK SUM
40310	SR	T2	00002	0443	-14	0520	X	
40320	RA	S1	40290	0444	+12	0441		ALTER LOOP ADDRESSES
40330	SU	D1	00002	0445	+05	0510	X	
40340	SA	S1	40290	0446	+15	0441	X	
40350	SA	S1	40300	0447	+15	0442	X	
40360	AD	T1	00052	0450	+11	0602		TEST FOR END OF LOOP
40370	PT	S1	40280	0451	+03	0440		REMAIN IN LOOP IF PLUS
40380	CS	T2	00002	0452	-37	0520		COPY CK SUM
40383	RA	T2	00000	0453	-12	0516		FIRST INTO FOR NEXT CARD
40386	AD	T1	00053	0454	+11	0603	X	
40390	UT	S1	40050	0455	+01	0410		RETURN TO PUNCH NEXT CARD
42010	RA	T1	00055	0456	+12	0605		SETUP DATA ORIGIN
42012	SR	A1	04094	0457	+14	7776	X	

42014	RA	A1	00037	0460	+12	0045	SETUP NUMBER HALF WORDS DATA
42016	SR	A1	04095	0461	+14	7777	X
42018	SS	A1	00074	0462	+36	0112	SKIP IF ANOTHER PROGRAM READY
42020	ST	A1	00006	0463	+00	0006	PROGRAM FINISH
42040	RD	A1	02048	0464	+30	4000	EQUIVALANT OF LOAD BUTTON
42050	CS	A2	00000	0465	-37	0000	X
42060	UT	A1	00000	0466	+01	0000	X
47010	SS	A1	00066	0467	+36	0102	ES CAPACITY EXCFEDED ERROR
47020	ST	S1	20310	0470	+00	0101	X
49001	ST	T2	00036	0471	-00	0562	0 LEFT ROW IMAGE ADDRESS
49002	ST	T2	00000	0472	-00	0516	9 LEFT ROW IMAGE ADDRESS
49003	CS	T2	00048	0473	-37	0576	CARD IMAGE END TEST ADDRESS
49005	UT	S1	37070	0474	+01	0374	DRUM WRITE RETURN ADDRESS
49006	CS	A2	/ /	0475	-37	0000	DRUM READ COPY LOOP TEST
49007	ST	S1	31160	0476	+00	0303	DRUM GATE ADDRESSES
49008	ST	S1	37040	0477	+00	0371	X
49009	ST	S1	40010	0500	+00	0404	X
49010	ST	A1	00257	0501	+00	0401	TAPE 2 ADDRESS
49997	ST	A1	00040	0502	+00	0050	PROGRAM ORIGIN
49998	ST	A1	00008	0503	+00	0010	NUMBER HALF WORDS DATA
49999	ST	A1	00060	0504	+00	0074	NUMBER HALF WORDS TEMPORARY

DATA

<u>Location</u>	<u>Contents</u>	<u>Binary Magnitude</u>
D1 0	0	17
D1 1	1	17
D1 2	2	17
D1 3	4096	17
D1 4	256	17
D1 5	44	17
D1 6	Drum Use Test Amt.	17
D1 7	4097	17

TEMPORARY

<u>Location</u>	<u>Use</u>
T2 0 } . . . T2 46 }	Card Image
T2 48	Operation, Type, and Address
T2 50	File and Symbolic Locations
T2 52	Tape Check Sum
T1 52	Print Conversion Column Indicator
T1 53	Working Search Lower Extreme
T1 54	Punching Loop Test Word
T1 55	Working Search Upper Extreme
T1 56	Half Word Count for Card
T1 57	Program Origin minus File Origin
T1 58	Origin Data Area
T1 59	Origin Temporary Area
T1 59	File Lower extreme
T1 59	Working Search Mean
T1 59	Number Half Words to Punch

## THE THEODOLITE DATA REDUCTION PROBLEM

By Bruce G. Oldfield  
U. S. Naval Ordnance Test Station  
China Lake, California

The development and testing programs at the Naval Ordnance Test Station (NOTS) produce a large amount of data. Success in these programs is often very dependent upon the rapid and accurate reduction of this raw data to useful numerical measures. There are many different kinds of recording instruments used on the ground and aircraft ranges at NOTS, and consequently there are many types of data produced, each requiring a different method of reduction. The reduction of the Askania theodolite data is one of the most important of these problems, and is the one that will be discussed in detail in this presentation.

The theodolite camera was first used at NOTS in 1945. Since this early beginning, there have been vast improvements in the camera, the methods of data reduction, and the computational facilities. This is well illustrated by contrasting the original combination of a Mitchell theodolite, simple film reader, and projection method for solution on a hand computer, with the present combination of a greatly improved Askania theodolite, mechanized film reading with the Iconolog punch film viewers, three camera weighted least squares solution and the 701 for the actual computation.

### 601, 602, and 604 Multipliers

The Computing Branch did its first theodolite reduction on a 601 multiplier in 1948. The Assessment Branch was soon interested in obtaining more accurate trajectories by making additional corrections to the angular data

and by using a more refined method of computation. A 602 multiplier was obtained and a modified least squares solution was set up which required forty different control panels. The best average time that was ever achieved was six minutes per point; however, there were often machine errors and breakdowns which substantially increased the actual computing time. This same method was used on the 604 multiplier when it was acquired, and on that machine the procedure required only twenty control panels, was faster, and was much less susceptible to errors and breakdowns.

#### Card Programmed Calculator (CPC)

The two methods previously mentioned could be computed with the step-by-step techniques of the 602 and 604. The present least squares method\* requires the approximate coordinates of each point to use as weighting factors in the final solution, which means that points along the trajectory must be computed in a sequential manner. This method, as set up for the model I CPC, required that four separate runs be made. In order to solve this problem in a more efficient manner, NOTS was very interested in a better 604. The subject was discussed with IBM representatives and resulted in the delivery of the original model II-604. For the first time it then became possible to sequence the complete theodolite computation. The deck required 295 instructions, three data cards, and approximately 165 seconds per point. With the arrival of the Model II Tabulator, the theodolite problem was set up on a very efficient multi-channel board which could enter as many as three eight-digit numbers and one five-digit number and exit with two eight-digit numbers each card cycle. This method required

\*References (1), (2), and (3) give a comprehensive discussion of the theodolite problem, including the weighted least squares solution now being used at NOTS.

80 seconds per point. By that time a faster and more versatile computer was the only way to make a significant improvement in the computational procedure.

### 701 Calculator

The computation of theodolite data on the 701 falls into three general categories: a searching routine, a computing routine, and an editing-computing routine. These three categories will now be discussed in greater detail.

### Searching Routine

Each flight is normally covered by a number of theodolite cameras stationed at various points along the range line. The problem is to use a particular set of two or three cameras for a specified part of the trajectory, change combinations several times for other parts of the trajectory, and quite often actually recompute sections of the trajectory using a different combination of cameras. To accomplish this generality of selection in a simple way, the following information and processes are used.

Camera Station Constants. Each station has an identification number and 13 constants associated with it. Each set occupies twenty half-words which are read into the 701 and stored at the beginning of the computation. As many as ten different stations can be used on any one trajectory.

Camera Station Coordinates. Each station is located by three position coordinates which may remain constant for several years. The identification numbers and position coordinates of twenty-one different stations are read into the 701 at the beginning.



Camera Data. The Askania records at a rate of four frames per second. The developed film is read on an Iconolog film reader, which is connected to an IBM summary punch that records on each card a station number, a frame number, and six quantities relating to the azimuth and elevation angles. The data from all the stations are read into the 701 at the beginning; provision is made for as many as 297 different points.

Control Numbers. The last type of information consists of three translation constants, three slant range constants, and as many as 10 different sets of control numbers. These control numbers designate the stations to be used and the frame numbers to be considered by this combination. For example, one might have stations 2, 5, and 9, frames 10 to 29, followed by stations 2, 9, and 11, frames 21 to 57, followed by a two station solution such as 5, 11, and 0, frames 50 to 75.

Since all of the information is read in at the beginning, the constants, coordinates, and data must be searched on the basis of the control numbers. The station constants and the station coordinates are searched just once for each set of control numbers. The first frame number is found for each station and then by advancing each frame address, all points are considered until the last frame constant is reached. This process rejects any point whose frame number is missing from one or more stations. The control information is considered one set at a time until all of the desired combinations have been computed.

#### Computing Routine

Computation of the desired space coordinates and residual angles for trajectory points is achieved by the following steps.

Corrected Angles. The first step is to compute the corrected azimuth and elevation angles for each camera. This computation takes the data along with the station constants and coordinates and adjusts both angles for such things as tracking corrections, leveling error, refraction correction, curvature of the earth, zero point correction, and dial eccentricity correction.

Space Position. From these corrected azimuth and elevation angles the direction cosines of each station's line of sight are computed. Using these direction cosines, the least squares coefficients for a set of three simultaneous equations are found. These equations are solved for the space coordinates of the missile.

Residual Angles. The next step is to compute the residual angles. A residual angle is defined as the difference between the measured azimuth or elevation angle of any station and the azimuth or elevation angle as computed from the space coordinates just determined above. These residuals should be quite small, and they provide an excellent check on the accuracy of the original data as well as on the accuracy of each 701 computation.

The following 12 quantities are computed for each point and must be stored: Frame number; x, y, z coordinates; slant range; six residual angles; and the time. It would be desirable to compute a number of points before storing the answers on drum or tape, but this would mean a compromise with the amount of original data which could be handled at any one time. Consequently, each set of answers is stored on the drum as soon as it has been computed.

Editing-Computing Routine

This routine gives, in most cases, a complete set of velocity and

acceleration computations. The program is on binary cards, which follow the last data card, and is called for as soon as the computations are completed. The answers on the drum are read back into electrostatic storage and examined in the following way.

(1) All missing frame numbers are counted and identified. Using this information the data is relocated, filling in zeros for each block of missing data.

(2) The residual angles are examined and tested against a specified tolerance. Any point that has a residual exceeding this tolerance is identified and consequently rejected.

(3) The x, y, z values are now found for all missing and all rejected points. A second degree formula is used which extrapolates forward if there are three good values, and also extrapolates backward if there are three good values. The average of these two extrapolated values is used as the new value. If there are three good values in only one direction, that extrapolation is used. (A previously extrapolated value is not considered a good value.) If there are not three good values in either direction, the routine notes this fact by leaving zero for the x, y and z coordinates.

(4) There is now available as complete a set of space positions as the program can get, so the next step is to compute the velocity and the acceleration.

(5) The results are now printed, each line representing one point. The form of the results are frame number, x, y, z, velocity, acceleration, slant range, six residual angles, and time. If the point has been extrapolated, it is identified by a zero frame number.

(6) An automatic data plotter is used in the preparation of the final report, so it is desirable to record the results on punched cards. The form of the punched results are frame number, x, y, z, velocity, acceleration, slant range, time for the position data, and time for the velocity.

The approximate 701 time required for a typical trajectory may be of interest. Let us consider a three-station, 90-point trajectory as an example. The time required to read in, compute, print, and punch the 90 points would be as follows:

Card reading time (program	24 seconds
Card reading time (data)	112 seconds
Computing time	95 seconds
Printing time	36 seconds
Punching time	<u>54</u> seconds
TOTAL TIME	321 seconds

Thus, this example would require 3.57 seconds per point, which is higher than the usual case since the time per point is reduced if a number of data points are used several times. This occurs when different combinations are required involving the same points.

The following are some interesting consequences of the 701 method.

- (1) The computation is more accurate even though the same equations are used as with the CPC. The method uses an approximate value of x, y and z in order to correct the angles for refraction and curvature of the earth and in order to weight the least squares coefficients. These are minor corrections and consequently the previously computed point on the trajectory is adequate to estimate the corrections unless it happens to be a bad point. In such a case sizeable errors can be introduced into the values for the

succeeding point. This difficulty is eliminated on the 701 by always computing  $x$ ,  $y$  and  $z$  a second time. (2) It is now quite common to request more combinations and more points on each test. In fact, on any given trajectory the tendency has been to compute almost twice as many points as would have been previously requested. (3) The editing-computing routine has caused a significant decrease in the number of man-hours spent on each test. In most cases the results can be plotted and sent out directly, with an estimated saving of twelve to twenty man-hours per test, which often means a saving of a week or more in issuing the final report because better scheduling is possible.

Work is continuing in an effort to improve the accuracy of the theodolite data. These efforts include improvements in the Askania camera, consideration of other methods of reduction, and an investigation of the bias of each Askania camera.

#### REFERENCES

- (1) U. S. Naval Ordnance Test Station, Inyokern. Methods of Measurement and Computation to Determine Trajectory Data from Askania Cinetheodolite Records, by John Titus, Mary Driggers, and Laurence Minvielle. China Lake, Calif., NOTS, 10 September 1951. (NAVORD Report 1907, NOTS 471).
- (2) U. S. Naval Ordnance Test Station, Inyokern. Techniques for the Statistical Analysis of Cinetheodolite Data, by R. C. Davis. China Lake, Calif., 22 March 1951. (NAVORD Report 1299, NOTS 369).
- (3) International Business Machines Corporation, Endicott, New York. Proceedings, Computational Seminar of August 1951. IBM Card Programmed Electronic Calculator Operations Using a Type 402-417 BB and 604-2, by Martha Kenyon, Bruce Oldfield, and Harley E. Tillitt.

## AN EXPERIMENT IN INFORMATION SEARCHING WITH THE 701 CALCULATOR

By Harley E. Tillitt  
U. S. Naval Ordnance Test Station  
China Lake, California

At the U. S. Naval Ordnance Test Station, an attempt has been made to use the 701 Calculator as a tool in the task of searching library files for documents referring to special subjects. The present system includes only reports which have been written in certain agencies throughout the country and does not include periodicals or books. Furthermore, the subjects are for the most part related to the development and testing of items of naval ordnance.

In any organization that includes research and development in its functions, it is economical in both time and money to be able to determine what has been done in a field before new programs are started. Scientists and engineers, therefore, are anxious to learn what is in the literature prior to starting some new task. Frequently, however, the labor of searching library files is so great or so unprofitable that it is either not done, or is done very incompletely.

One of the reasons for the difficulty in searching is that the cataloging of reports may be such that important aspects of their contents are obscured. For example, the following report, Equilibrium Composition and Thermodynamic Properties of Combustion Gases, could logically be cataloged under one or more of several subject headings, which might or might not be appropriate, depending somewhat upon the technical skill of the cataloger.

This particular report was filed in the Inyokern Technical Library under two subjects: Gases and Physics. Both of these are standard Library of Congress subject headings, and are more or less descriptive of the report.

However, under each subject heading there were found to be several hundred other reports filed, in itself a situation that could discourage searching. More serious however, was the fact that scientists interested in such a category of ordnance development might be equally likely to search under the subjects of Combustion or Physical Chemistry. Most serious however, was the fact that there was no indication in the cataloging process that one of the main contributions of the report was to describe a numerical method by means of which the thermodynamic properties were computed. As a result, for one reason or another, the report was, in certain respects, lost, as far as many interested individuals were concerned.

To avoid some of the difficulty of cataloging documents by subject heading, a system can be used that depends upon a document being described by several single terms called descriptors.<sup>1</sup> In the library application of this system, there is a card for each descriptor. As documents come to the library they are given an acquisition serial number and this number is entered upon as many different descriptor cards as seem necessary to describe the document.

In the example above, if the serial number of the report had been 1234, this number might have been entered on the following cards: Thermodynamics; Combustion; Gases; Computation; Fuel; Impulse; Pressure; Temperature; Entropy; Enthalpy; Adiabatic. Some descriptors do not seem related

<sup>1</sup>One discussion of this type of system is given in a series of 8 technical reports by Mortimer Taube of Documentation Incorporated, Washington 6, D.C. These reports were prepared under Contract No. AF 18(600)-376 for the Armed Forces Technical Information Agency in the period July 1952 to March 1953.

to the title, but could have been assigned after a brief inspection of the contents by the cataloger. To use such a system when information of a certain type is desired, an individual would list descriptors that would, in his opinion, describe his needs. These descriptor cards would then be pulled from the files and be visually compared for numbers that matched on the several cards. Reports corresponding to these matching serial numbers would then be withdrawn.

The original purpose of the 701 program to be described was to mechanize the above procedure with a view to the possible establishment of a daily schedule for library searching.

In designing the 701 system, attention was given to the current size of the file and the expected growth during the next five years. The two quantities considered were the expected total number of serial numbers and the total number of descriptors. It was estimated that during the next five years there would be no more than 30,000 serial numbers nor more than 5000 descriptors. Furthermore, it was estimated that in searching for documents on a particular subject no more than 8 descriptors would be listed and that any one of these would not have more than 1000 serial numbers associated with it.

This coordinate index system has only recently been put into use at the Naval Ordnance Test Station, and at the time the 701 programming was started there had been established a list of approximately 2500 descriptors. New descriptors are being added at the rate of about 100 per month, with an anticipated upper limit of 5000. On these 2500 cards there had been recorded a total of about 20,000 numbers describing nearly 4000 documents, indicating that each serial number was recorded on an average



of 5 different descriptor cards.

At the present time, additions are being made to the system at the rate of about 500 documents per month. This currently represents approximately 4000 additional entries of serial numbers per month, since the catalogers are becoming more experienced with the system and are now using about 8 descriptors per document.

The 701 operations are quite simple and go through nearly the same steps as are required in a normal hand search. These steps are as follows:

(1) Install the master tape reel on which the file has been written. (The present arrangement of information on the tape is that each descriptor and associated report serial numbers form a unit record. The unit records are on the tape in order of increasing descriptor number.)

(2) Load the searching program plus from 1 to 20 cards on each of which are punched the 2 to 8 descriptors called K's, that describe the subject of interest. (After loading, transition is made to the search program itself.)

(3) Report serial numbers which appeared under all selected descriptors are printed.

The following brief descriptions show the purpose of several programs used in the system.

It can be seen that A and B will be used only once, when the system is started; C through G whenever a search is made; and H and I when additions are made as required by the continued acquisition of documents.

Program A. Read into electrostatic storage as many decimal cards as required for a descriptor group.

- Program B. Compute a check sum for a descriptor group and write it plus the group on tape.
- Program C. Read a group, including its descriptor, from tape and match the descriptor against the 2 to 8 K's.
- Program D. If a group descriptor matches any K write the group on drum.
- Program E. After either 8 groups have been written on drum, or all of the K's exhausted, read the first two groups from drum and match report serial numbers. Store the matches where the first group had been.
- Program F. Read subsequent groups from drum one at a time and continue to match those that remain with each new group, storing these, where the first group had been.
- Program G. When all groups have been read from drum and matched, print the final matches that remain.
- Program H. Read a group from cards. Determine whether this is an addition to a group already on the tape or a group having a descriptor not previously used.
- Program I. If the group in H is an addition to an old group, produce a new check sum for the old group plus the addition. If the group in H is new, produce a check sum for it. In either case collate the new information with that on the tape and write it on a second tape.

One of the objectives of the experiment is to attempt to reduce the amount of time spent in entering new serial numbers onto cards. This is a

hand job requiring manipulation of the file and the recording of numbers, which is a slow process as well as a source of interference with individuals wishing to use the file at the same time. With the use of Programs H and I, the system can be kept up to date without the need for hand operations except for the listing of additions on the sheet of paper as contrasted with making card entries.

A second objective is to attempt to establish a daily schedule for document searching. Presumably, this would eliminate conflicts that arise when more than one person happens to want to use the file at the same time. Also, it is possible that if the mechanics of searching are such that scientists and engineers can delegate the task to their secretaries (and the 701), the general use of reports will be increased, with presumably beneficial results. At the present time from 10 to 20 searches are made per day.

Although there are cases when an individual may wish to search at once, it is believed that most such "urgent" needs can be planned to meet a schedule, especially if such a schedule would include two periods, such as 11:30 AM and 4:00 PM. There has been no real experience on this part of the experiment as yet.

At present only one search can be made at a time. This is because the system is built to accommodate 8 descriptors per search, each of which might contain up to 1000 serial numbers. However, as indicated above, up to 20 searches can be made to follow in order with only one loading.

An improvement planned but not yet in effect is that of searching for 8 K's but also printing those serial numbers that match for 7, 6, 5, 4, 3, or 2 K's.

It is difficult to estimate the 701 time required for what may become a typical scheduled searching period. This depends upon several factors,

including the total number of searches to be made in one period, the number of K's, the number of serial numbers per descriptor, and the location of the descriptor groups on the tape.

The time required to load the cards, which include the program and the K's, plus the pushing of card reader and load buttons, is about 10-15 seconds. A search for 8 K's through 300 groups, with from 5 to 40 serial numbers each, all located at the front of the tape, requires about 10-15 seconds. The minimum time of search, therefore, is the range of 20-30 seconds.

As the file increases in size, by new descriptor groups being entered and new serial numbers being added to old groups, the time per search will approach that required to read the tape.

If the present estimate of 4000 new entries per month proves to be correct, there will be about 240,000 plus the present 20,000 in five years. Since these are recorded as half words, there would be room to load the file on one 1200 foot tape. Therefore, a maximum search would require about the time needed to read 1200 feet of tape, or approximately 4 minutes. The Inyokern Technical Library staff suggests that if the labor of putting entries on cards is reduced, the number of descriptors assigned to a document may be greatly increased, perhaps by a factor of two. If this should happen, a single tape might not be sufficient.

In summary, this paper describes a method by means of which the 701 Calculator can perform certain library searching tasks. Depending upon several variables, a single search may require as little as 20 seconds or as much as 4 minutes. The system is at present in the nature of an experiment, and whether or not it will prove to be economical or practical remains to be seen.

## FOR CUSTOMERS

IN PRACTICE, as well as influence, IBM educational programs reach outside the Company. This is a result of the widespread use of IBM products, with consequent need for personnel trained in their applications, operation, and administration, and to the fact that IBM is primarily a service institution.

As a direct service to users of its products, IBM conducts several types of educational courses for customer personnel. Year-round one- and two-week courses and seminars on IBM Accounting, statistical and computing methods and related subjects are given at Endicott by the IBM Department of Education for groups of customers — executives, supervisors, administrators, accountants. More than 18,000 customer representatives from 55 countries have attended these sessions since their inception. Over 200,000 customer employees and trainees have entered the field of IBM Accounting machine operation through enrollment in operator training courses conducted by IBM branch offices in principal cities.

In still another field, IBM goes outside of its own membership to sponsor the specialized training of selected groups of college students in the Watson Scientific Computing Laboratory at Columbia University. IBM maintains a co-operative relationship with many other colleges and universities engaged in the teaching of IBM Accounting and computing methods in professional and graduate schools.

## LIFELONG LEARNING

IN ADDITION to its many instruction programs, IBM also offers educational advantages in the form of exhibitions, public events, lecture programs, concerts, publications, and other activities. These support the view that education is a vital force which is not to be confined to formal programs and class meetings. The challenge of every-day living calls for comprehensive and lifelong learning. IBM's policy is to assist individuals and groups to meet this challenge.

Steady increase of enrollments in the various programs, by customers as well as employees, furnishes good evidence that the participants recognize and appreciate the personal benefits of IBM educational opportunities.

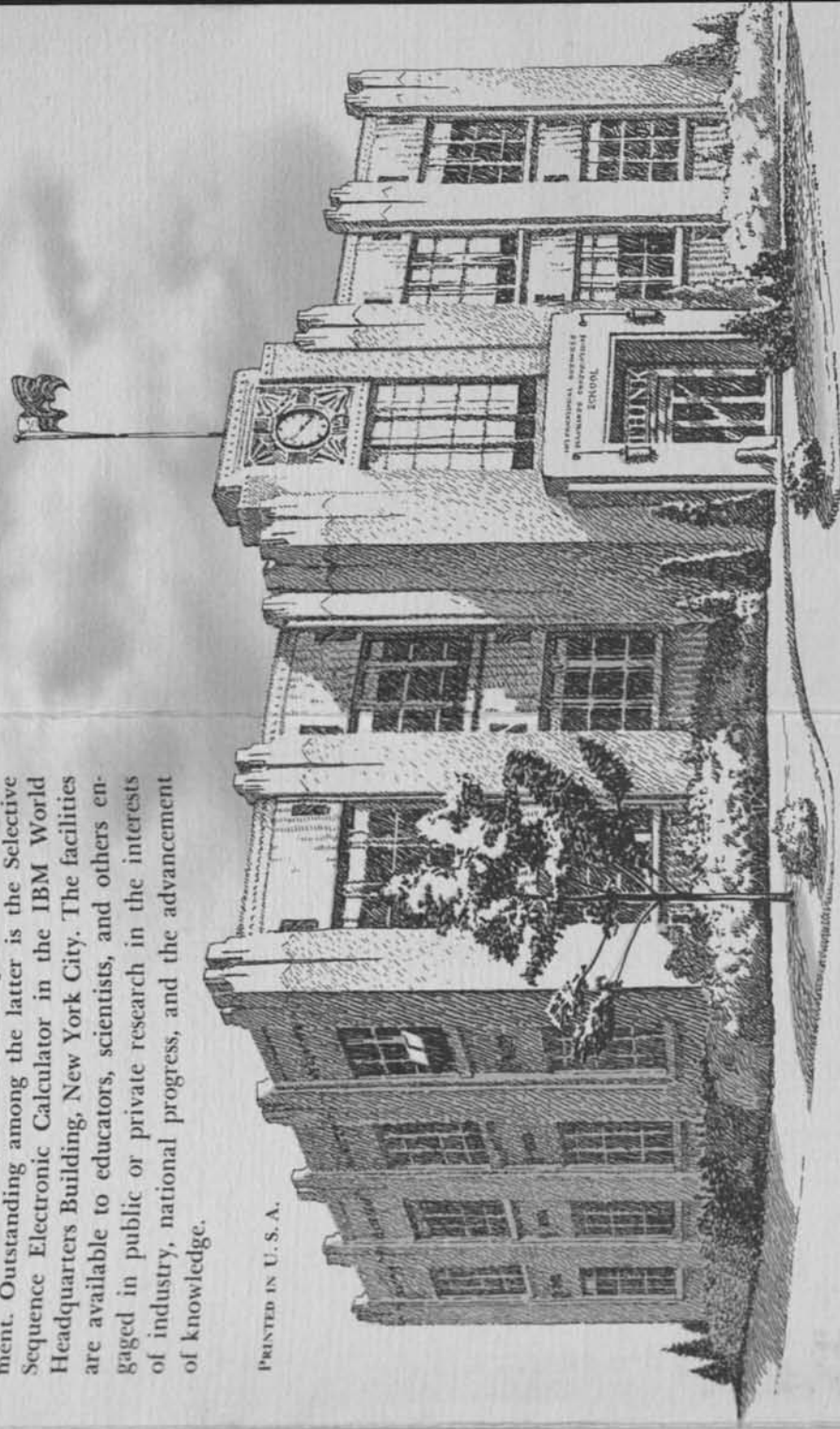
## IBM IN EDUCATION

ANY DESCRIPTION of the place of education in IBM would be incomplete without mention of the increasingly important role of the Company in the general field of education. Educational and personnel testing procedures have been advanced by the IBM Electric Test Scoring Machines, which make possible rapid and accurate scoring and analysis of objective-type examinations. In addition, IBM Accounting as a whole has been applied effectively to the needs of education in general and to various school and college operations in particular — educational surveys and analyses, registration procedures, financial and administrative control.

Directly and indirectly, IBM further serves education and science through statistical data, calculations, and mathematical problem-solving made possible or readily available by standard and special IBM equipment. Outstanding among the latter is the Selective Sequence Electronic Calculator in the IBM World Headquarters Building, New York City. The facilities are available to educators, scientists, and others engaged in public or private research in the interests of industry, national progress, and the advancement of knowledge.

PRINTED IN U. S. A.

# Education in IBM



590 MADISON AVENUE, NEW YORK 22, N. Y.

FORM 20-3696-0

The Endicott IBM School

ATION IN IBM is primarily an organized range of ideas, facts and experiences. Everyone in the Company takes part in some form of systematic learning, and the participation continues from year to year. Educational programs, covering both recreational courses for vocational preparation and voluntary courses meeting general interests, make a threefold contribution to the development of IBM people, their knowledge, and service.

At the beginning of his work in IBM, Mr. Thomas J. Watson, president of the Company for 35 years and a member of the Board since 1949, laid the foundation and guided the development of the educational programs which have been identified with every phase of the business. The early beginnings, steady growth, and wide range of education in IBM are outlined by the following outline of some of the programs and the years in which they were introduced:

Instruction in applications of products, and in IBM policies, 1915.

Engineering instruction in the installation and maintenance of IBM equipment, 1918.

Operators' training, 1929.

Development Engineering training, 1932.

Maker apprentice training, 1932.

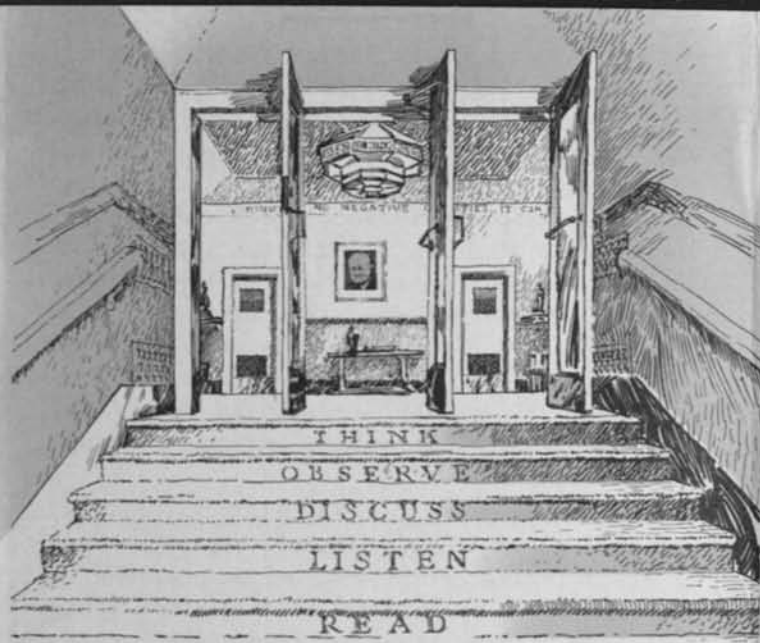
Misses Service training of women field representatives of IBM, 1935.

Administrators' and executive instruction in IBM Accounting, statistics, and scientific research, 1936.

Advanced applied mathematics and computation, 1946.

Graduate engineering studies, 1949.

Meanwhile, other types of instruction which were streams for what became a comprehensive general vocational education program, as well as management and job training activities, had beginnings in the 20's and 30's. And with the completion of a school building in 1933 at Endicott, New York, its largest manufacturing plant and engineering development, IBM had a useful symbol of the significance of education in the operations of the Company.



FIVE STEPS TO KNOWLEDGE  
Entrance to the Endicott IBM School

IBM EDUCATIONAL PROGRAMS fall generally into these broad classifications: for IBM employees, for IBM's customers, and for special professional and scientific groups. By their participation all groups acknowledge the fundamental truth stated by Mr. J. Edgar Watson, "There is no saturation point in education."

#### FOR EMPLOYEES

*Professional Training.* The specialized technical knowledge required in the development, building, selling and servicing of IBM products, together with the high degree of teamwork characteristic of the organization, makes continuing education an indispensable part of Company operations. Engineering, sales, customer engineering, systems service and other technical work all require specialized IBM training, and the various programs give IBM personnel a good start in their careers. Training proceeds from preliminary preparation in branch offices, plants or laboratories to full-time instruction in the specialized schools. Nor does the

training end upon completion of the basic courses in each field. Keeping pace with advances, continuing their learning as a natural part of their work, employees in these and other divisions frequently attend classes for post-graduate study of new methods and machines.

*Job Training.* Apart from the above programs, other members of IBM participate, at one time or another, in educational activities within the Company's regular operations. Every level of plant, office and field activity is reached by job training programs. Periodic short-term courses in job instruction, job methods and human relations are conducted for managers and executives. Longer courses, requiring as much as one to three years for completion, are also given in specialized fields or management activities. Extension education materials have been developed for employees, especially those in the field, who may be unable to attend regular classes. Additional material for home study is made available from time to time.

*Business, Technical and Cultural Courses.* Surrounding the various courses and groups of courses related to particular occupations and responsibilities within the Company is the General and Vocational Education program conducted at plant locations and in the New York City area. This is a full-scale adult education program offering spare-time courses ranging from ungraded technical subjects to liberal arts studies on the college level. Enrollment is entirely voluntary, without cost to employees. Seventy-five per cent of plant employees have participated in this program, with many of them enrolling regularly each term. Certificates are awarded for fifteen or more units. There is no hard-and-fast distinction between vocational and general development but many employees regularly maintain a good balance by taking various kinds of courses in subjects serving either both aims or none save the need for a worthwhile, stimulating pastime. These courses, whose class meetings usually are once a week for two hours, are given in a two-semester year which conforms generally to public school practice. Instructors are specially qualified Company personnel and faculty members from nearby colleges and schools.



EDUCATION IN IBM is primarily an organized interchange of ideas, facts and experiences. Everyone in the Company takes part in some form of systematized learning, and the participation continues from year to year. Educational programs, covering both required courses for vocational preparation and voluntary courses meeting general interests, make a three-fold contribution to the development of IBM people, business, and service.

From the beginning of his work in IBM, Mr. Thomas J. Watson, president of the Company for 35 years and Chairman of the Board since 1949, laid the foundations for and guided the development of the educational programs which have been identified with every phase of the business. The early beginnings, steady development, and wide range of education in IBM are indicated by the following outline of some of the particular programs and the years in which they were instituted:

SALES instruction in applications of products, and in IBM policies, 1915.

ENGINEERING instruction in the installation and maintenance of IBM equipment, 1918.

CUSTOMERS' OPERATORS training, 1929.

DEVELOPMENT ENGINEERING training, 1932.

TOOLMAKER apprentice training, 1932.

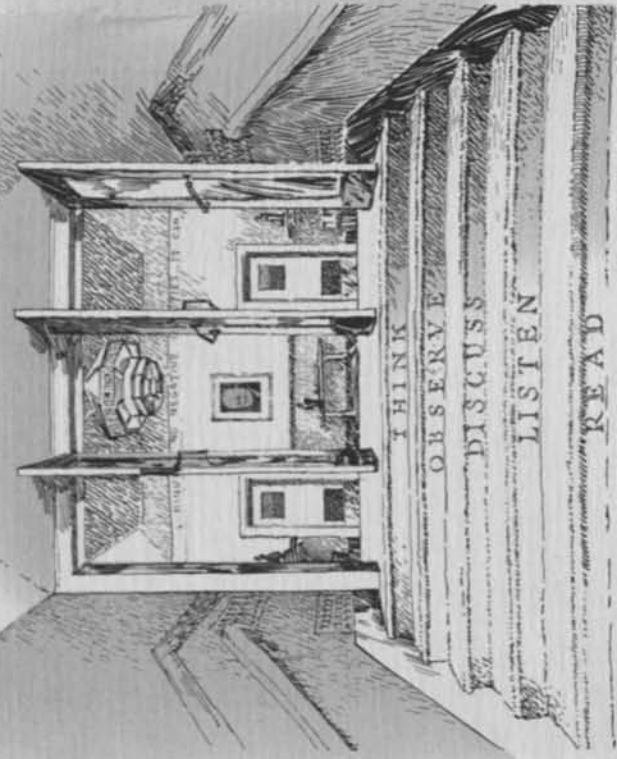
SYSTEMS SERVICE training of women field representatives of IBM, 1935.

CUSTOMERS' ADMINISTRATORS and executive instruction in IBM Accounting, statistics, and scientific research, 1936.

ADVANCED applied mathematics and computation, 1946.

POST-GRADUATE engineering studies, 1949.

Meanwhile, other types of instruction which were source streams for what became a comprehensive general and vocational education program, as well as management and job training activities, had beginnings through the 20's and 30's. And with the completion of its own school building in 1933 at Endicott, New York, site of its largest manufacturing plant and engineering establishment, IBM had a useful symbol of the significance of education in the operations of the Company.



#### FIVE STEPS TO KNOWLEDGE

Entrance to the Endicott IBM School

IBM EDUCATIONAL PROGRAMS fall generally into these broad classifications: for IBM employees, for IBM's customers, and for special professional and scientific groups. By their participation all groups acknowledge the fundamental truth stated by Mr. Watson, "There is no saturation point in education."

#### FOR EMPLOYEES

*Professional Training.* The specialized technical knowledge required in the development, building, selling and servicing of IBM products, together with the high degree of teamwork characteristic of the organization, makes continuing education an indispensable part of Company operations. Engineering, sales, customer engineering, systems service and other technical work all require specialized IBM training, and the various programs give IBM personnel a good start in their careers. Training proceeds from preliminary preparation in branch offices, plants or laboratories to full-time instruction in the specialized schools. Nor does the

training end upon completion of the basic courses in each field. Keeping pace with advances, continuing their learning as a natural part of their work, employees in these and other divisions frequently attend classes for post-graduate study of new methods and machines.

*Job Training.* Apart from the above programs, other members of IBM participate, at one time or another, in educational activities within the Company's regular operations. Every level of plant, office and field activity is reached by job training programs. Periodic short-term courses in job instruction, job methods and human relations are conducted for managers and executives. Longer courses, requiring as much as one to three years for completion, are also given in specialized fields or management activities. Extension education materials have been developed for employees, especially those in the field, who may be unable to attend regular classes. Additional material for home study is made available from time to time.

*Business, Technical and Cultural Courses.* Surrounding the various courses and groups of courses related to particular occupations and responsibilities within the Company is the General and Vocational Education program conducted at plant locations and in the New York City area. This is a full-scale adult education program offering spare-time courses ranging from ungraded technical subjects to liberal arts studies on the college level. Enrollment is entirely voluntary, without cost to employees. Seventy-five per cent of plant employees have participated in this program, with many of them enrolling regularly each term. Certificates are awarded for fifteen or more units. There is no hard-and-fast distinction between vocational and general development but many employees regularly maintain a good balance by taking various kinds of courses in subjects serving either both aims or none save the need for a worthwhile, stimulating pastime. These courses, whose class meetings usually are once a week for two hours, are given in a two-semester year which conforms generally to public school practice. Instructors are specially qualified Company personnel and faculty members from nearby colleges and schools.





*THE HOMESTEAD  
and PARK AREA*

IBM AT ENDICOTT, NEW YORK

Approach to the IBM Homestead and Park



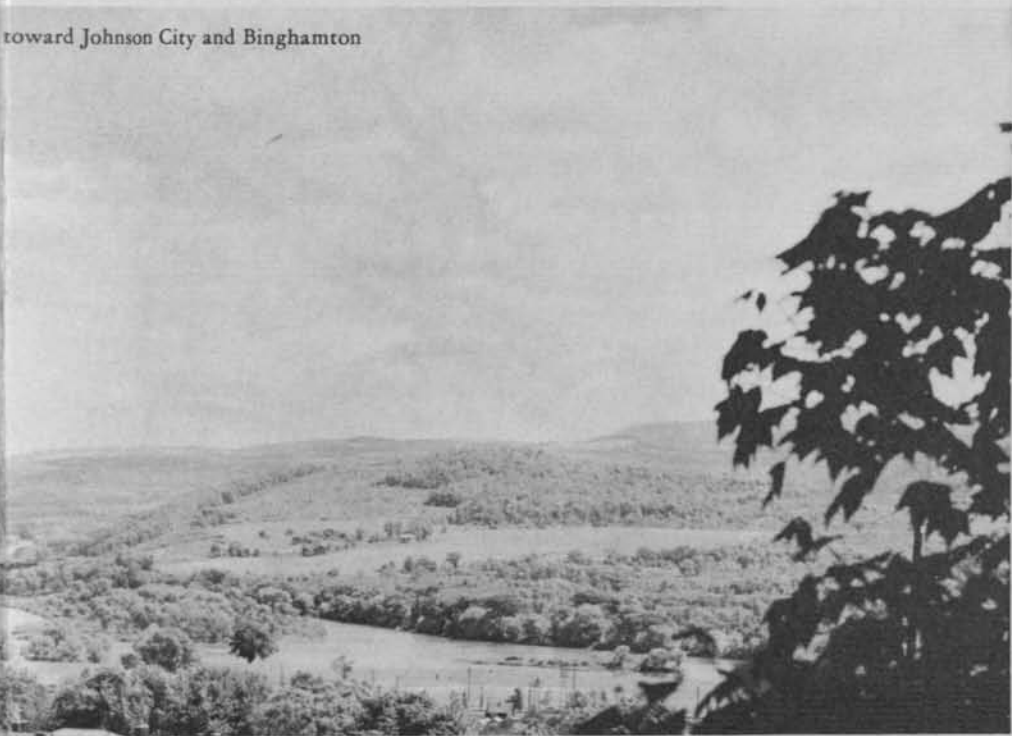
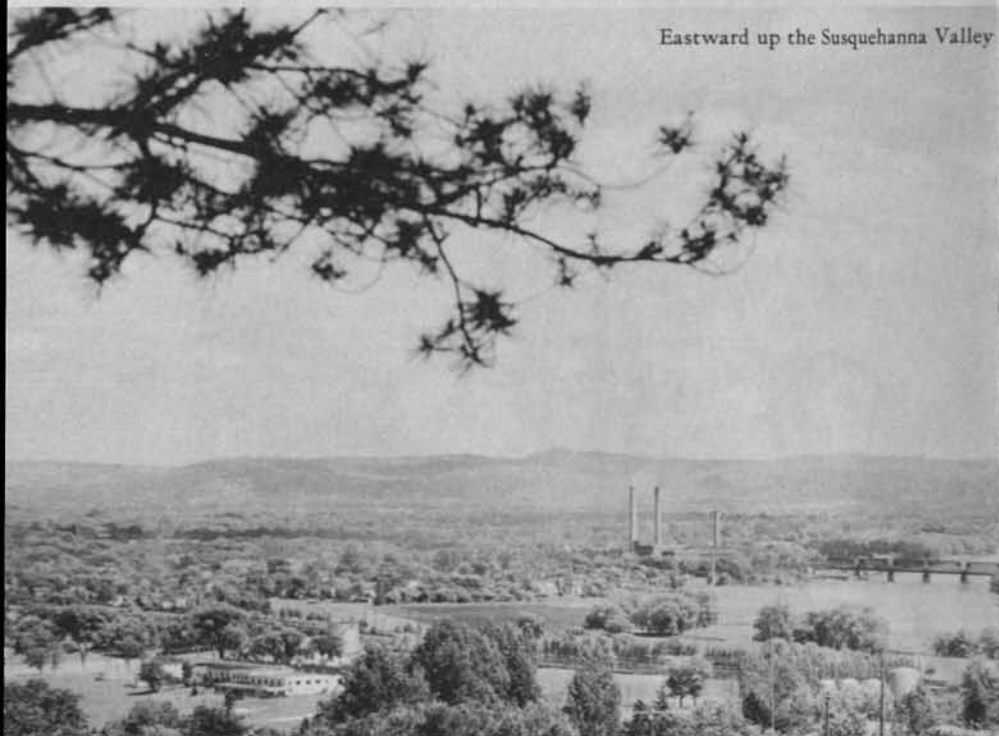
Overlooking the upper Susquehanna Valley about two miles east of Plant J, the Homestead is a guest house for members of IBM customer classes meeting in Endicott, New York

THE IBM HOMESTEAD

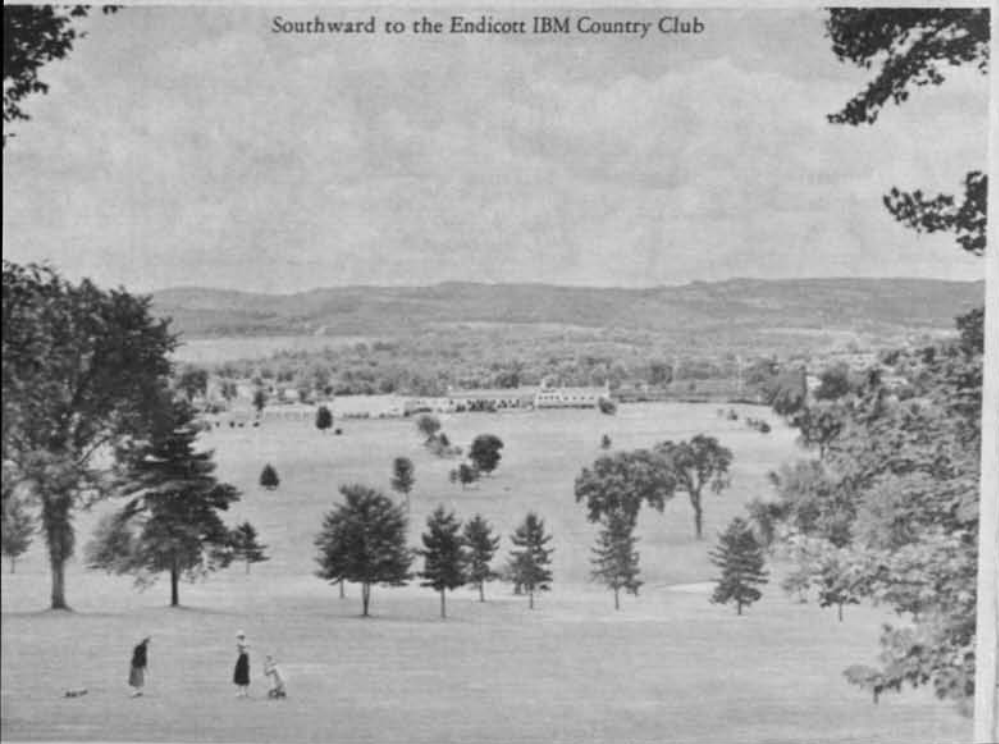
VIEWS FROM RIM TRAIL

MAP OF THE PARK AREA

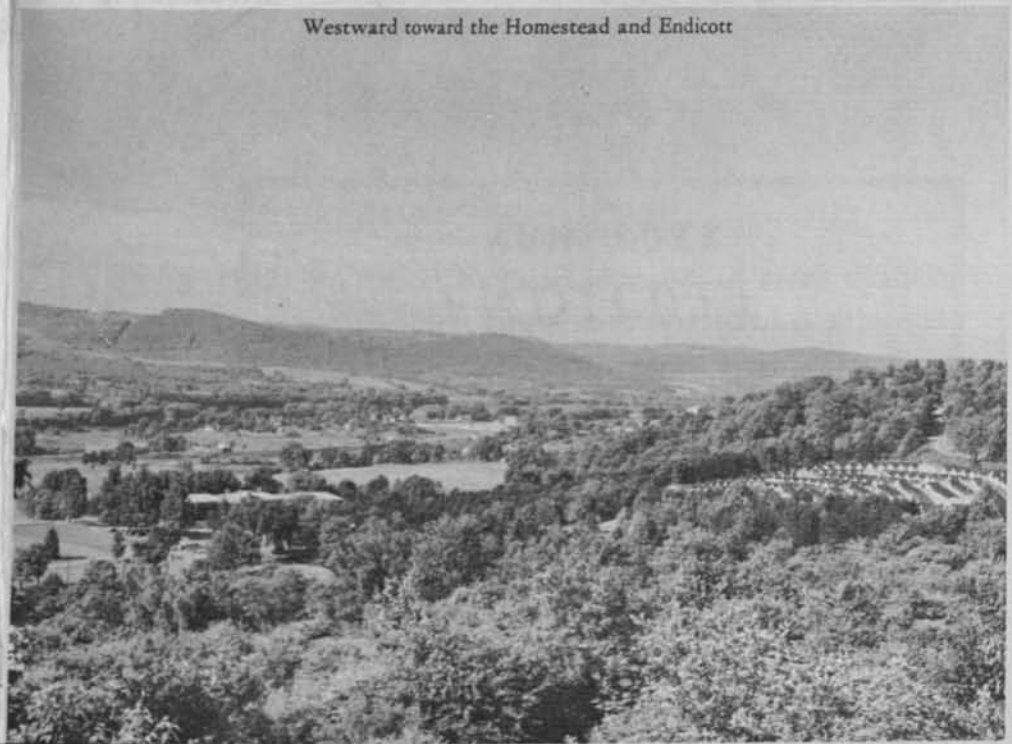
Eastward up the Susquehanna Valley toward Johnson City and Binghamton



Southward to the Endicott IBM Country Club



Westward toward the Homestead and Endicott



VIEWS FROM RIM TRAIL

MAP OF THE PARK AREA

# IBM PARK

ENDICOTT  
NEW YORK



SCALE - One inch = 150 Yards  
LEGEND for Picnic Sites

THE IBM PARK at Endicott consists of more than six hundred acres of rolling greens and woodland overlooking the Susquehanna Valley in south-central New York State. It abounds with hiking and nature trails, picnic sites, and other recreational facilities. Several miles of surfaced roads wind through the park, affording vantage points for lovely upland scenery in all seasons. The area contains buildings and grounds of the IBM Homestead and the IBM Country Club, with their respective golf courses of nine and 18 holes.



A waterfall in one of the picnic glens north of the Homestead



REPORT MR-E-18

DATE 20 April 1954

*Convair*

TITLE

"Calculation of Aerodynamic Characteristics for  
Lifting Surfaces by the Vortex Lattice Theory"

CONSOLIDATED VULTEE AIRCRAFT CORPORATION  
FORT WORTH DIVISION • FORT WORTH, TEXAS

REPORT MR-E-18

DATE 20 April 1954

TITLE

"Calculation of Aerodynamic Characteristics for  
Lifting Surfaces by the Vortex Lattice Theory"

PREPARED BY: W. E. Meyer GROUP: Aerophysics  
CHECKED BY: H. S. W. Planski APPROVED BY: J. C. Good

NO. OF PAGES 47

NO. OF DIAGRAMS 10

TABLE OF CONTENTS

	Page
LIST OF FIGURES.....	3
LIST OF TABLES.....	4
LIST OF SYMBOLS.....	5
ABSTRACT.....	8
INTRODUCTION.....	9
THEORY.....	10
EQUATIONS.....	13
PROGRAM.....	23
RESULTS.....	33
DISCUSSION.....	44
REFERENCES.....	47



LIST OF FIGURES

Number		Page
1	Diagram of Wing Planform Parameters.....	20
2	Diagram of Vortex Lattice and Control Points.....	21
3	General Flow Chart of Computation Procedure.....	29
4	Flow Chart for Computation Program F.....	30
5	Flow Chart for Computation Program G.....	31
6	Flow Chart for Computation Program M.....	32
7	Diagrams of Example Wings.....	37
8	Local Aerodynamic Center versus Semi-span Length for Wing A.....	38
9	Local Aerodynamic Center versus Semi-span Length for Wing B.....	39
10	Local Aerodynamic Center versus Semi-span Length for Wing C.....	40

## LIST OF TABLES

Number		Page
1	P-circulation, Downwash, and Modified Simpson Factors.....	22
2	Spanwise Loading Coefficients for Wing A.....	41
3	Spanwise Loading Coefficients for Wing B.....	41
4	Spanwise Loading Coefficients for Wing C.....	42
5	Aerodynamic Center of the Complete Wing for Wings A, B, and C.....	43
6	Lift Curve Slope for Wings A, B, and C.....	43

## LIST OF SYMBOLS

- |     |                    |  |
|-----|--------------------|--|
| 1.  | AR                 | wing aspect ratio, $4s^2/S$  |
| 2.  | a                  | wing cutout factor (Figure 1)                                      |
| 3.  | $a_{mn}$           | loading series coefficients  |
| 4.  | $C_{Di}$           | induced drag coefficient, $D_i/qS$                                 |
| 5.  | $C_L$              | lift coefficient, $L/qS$   |
| 6.  | $C_{L\alpha}$      | lift curve slope $\partial C_L / \partial \alpha$                  |
| 7.  | $C_O$              | chordwise dimension used in wing planform definition $C_R / (1-a)$ |
| 8.  | $C_R$              | wing root chord  |
| 9.  | c                  | local wing chord   |
| 10. | $\bar{c}$          | wing mean aerodynamic center, $\frac{1}{S} \int_0^l c^2 d\eta$     |
| 11. | $c_l$              | local lift coefficient at a given span station                     |
| 12. | $D_i$              | induced drag   |
| 13. | F                  | geometric function used in downwash calculation                    |
| 14. | $F_m(\eta)$        | spanwise term in basic loading series                              |
| 15. | $G_{V,A}; G_{V,B}$ | dimensionless circulation factors                                  |
| 16. | L                  | lift   |
| 17. | l                  | local lift at a given span station                                 |
| 18. | M                  | Mach number  |
| 19. | $P_a, P_b$         | auxiliary circulation functions used in modified solution          |
| 20. | $p_a, p_b$         | additional circulation series coefficients                         |

- used in modified solution
21.  $p$  pressure
  22.  $q$  dynamic pressure,  $\frac{1}{2} \rho V^2$
  23.  $S$  wing area
  24.  $s$  wing semispan
  25.  $V$  free stream velocity
  26.  $W$  downwash velocity
  27.  $X^* Y^*$  dimensionless coordinates used in downwash calculation for a horseshoe vortex
  28.  $x_{ac}$  local aerodynamic center at a given span station
  29.  $\bar{x}_{ac}$  aerodynamic center of a complete wing
  30.  $\alpha$  angle of incidence
  31.  $\Gamma$  total circulation about the wing chord
  32.  $\epsilon$  complement of the leading edge sweepback angle (Figure 1)
  33.  $\eta$  dimensionless wing span station (Figure 2)
  34.  $\eta_{cp}$  wing spanwise center of pressure
  35.  $\lambda$  taper ratio, i.e., ratio of wing tip chord to root chord (Figure 1)
  36.  $\mu$  spanwise station defining the center of a horseshoe vortex (Figure 2)
  37.  $\nu$  chordwise position index for bound vortex locations (Figure 2)
  38.  $\rho$  density of air

39.  $\phi$  span parameter used in induced drag calculations,  $\sin^{-1}\sqrt{1-\eta^2}$

40. R represents residual or loading error due to planform discontinuities near the root

Subscripts:

o refers to vortex locations  
p refers to control point locations

Note:

The quantities  $a_{00}$ ,  $a_{02}$ ,  $a_{04}$ ,  $a_{10}$ ,  $a_{12}$ ,  $a_{14}$ , defined as symbol 3 are concerned with the standard, uncorrected solution. They will be referred to, hereafter, as the Faulkner "a" functions or simply as the "a" functions. The quantities  $a_0'$ ,  $p_{a0}$ ,  $p_{b0}$ ,  $a_1'$ ,  $p_{a1}$ ,  $p_{b1}$ , defined as symbols 3 and 20, are concerned with the modifications to the standard solution and will be referred to, hereafter, as the Faulkner "p" functions or the "p" functions.

## ABSTRACT

A procedure for the calculation of aerodynamic characteristics for lifting surfaces by V. M. Falkner's Vortex Lattice Theory has been used at Convair on a study of over 500 wing planforms. This study has supplied the aerodynamicist with predictions of the lift curve slope, spanwise loading, spanwise center of pressure, induced drag coefficients, aerodynamic center and local wing pressure coefficients for the various wing planforms. The uniform simplicity of the vortex lattice allows straightforward calculations which can be handled successfully on electronic calculators. Due to the large number of calculations required for one wing planform, application of this theory has become practical only with the aid of a high speed, large storage capacity electronic calculator.

## INTRODUCTION

In the evaluation of a given airplane design proposal the accurate estimation of the aerodynamic characteristics of the configuration is of primary importance. For the purpose of developing various prediction methods, Convair has set up at its Fort Worth Division a group devoted to the study of generalized lift and drag problems. The prediction methods are usually derived from experimental data correlation based on available theoretical results. This paper summarizes the adaptation on one of these theoretical methods, the Falkner subsonic lifting surface theory, to the IBM 701 calculator.

## THEORY

In the field of aerodynamic theory the existence of an exact solution is the exception rather than the rule. As a result a given method is only as accurate as the approximations used to arrive at its solution and, because of the various possible approximations, many methods exist for the solution of the various aerodynamic problems. For example, at least 15 methods, suggested by various authors, exist for the solution of the subsonic lifting wing problem.

Of the various lifting wing methods the three currently receiving the most interest are the Weissinger modified lifting line theory, the Multhopp continuous lifting surface theory and the Falkner vortex lattice theory. The latter two, being lifting surface theories, define chordwise loading on the surface whereas the lifting line theory assumes a given chordwise distribution. This consideration ruled out the use of the Weissinger theory. The Multhopp theory, though slightly more accurate than the Falkner theory, is more difficult to adapt to a calculator because of the occurrence of incomplete elliptic integrals in a part of the downwash summation procedure.

The Falkner theory was chosen for use in this study for the following reasons:

1. It is only slightly less accurate than the Multhopp theory.
2. Being a lifting surface theory, it provides solutions for the chordwise loading.



3. Because of its mathematical simplicity it is relatively easier to adapt for calculation and also easier to extend to related lifting wing solutions such as the downwash in space and biplane problems.

The Falkner theory arranges (or concentrates) the vortex sheet of the wing into a number of elements of line vorticity which are arranged in a lattice over the surface. The circulation of a given vortex can be calculated and knowing the circulation, the downwash can be obtained. The downwash effect of all of the vortices is then obtained at a series of stations or control points. The results obtained at the control points can be written into a set of equations, the solutions of which are used to obtain the total circulation of the wing at various span stations. If the circulation is known, it is possible to obtain various aerodynamic characteristics such as, the lift curve slope, the spanwise loading coefficients, the spanwise center of pressure, the coefficient of induced drag, the local aerodynamic center, the complete aerodynamic center and the local wing pressure coefficient among others.

Moreover, by modifying the procedure, it is possible to determine changes in lift caused by camber and twist, the effectiveness of flaps, the downwash in space to obtain airflow characteristics in the vicinity of the wing, and other quantities determined by the downwash or circulation.

The Falkner method of calculating these characteristics was summarized into a set of equations by Kulakowski (reference 5), which were programmed for the IBM-701 calculator.

The method was restricted, previously, to the study of a few carefully chosen wing planforms because of the great amount of computational labor involved (estimated by Falkner at twelve to fourteen days per wing planform). The time required for the present program, on the 701, is about ten minutes.

The greatly reduced time of calculation makes possible the use of this procedure for generalized wing studies. The program has been applied at Convair to over 500 planforms and a preliminary investigation of the results have indicated that the theory is valuable enough to justify an additional study of around 800 more planforms.

## EQUATIONS

The equations used in the calculations are summarized below. These equations have been formulated from Falkner's Vortex Lattice Theory by Mr. L. J. Kulakowski, Senior Aerodynamics Engineer, Convair-Fort Worth Division (ref.5).

1.  $Y_p^* = 20 \bar{\eta}_p$
2.  $X_p^* = -\frac{20}{\tan \epsilon} \left[ |\bar{\eta}_p| + \frac{2\nu_p - 1}{12} (1-a) \left( \frac{1}{1-\lambda} - |\bar{\eta}_p| \right) \right]$
3.  $Y_o^* = 20 \mu$
4.  $X_o^* = -\frac{20}{\tan \epsilon} \left[ |\mu| + \frac{2\nu_o - 1}{12} (1-a) \left( \frac{1}{1-\lambda} - |\mu| \right) \right]$
5.  $Y^* = Y_p^* - Y_o^*$
6.  $X^* = X_p^* - X_o^*$
7.  $F = \frac{X^* - \sqrt{X^{*2} + (Y^* + 1)^2}}{X^* (Y^* + 1)} - \frac{X^* - \sqrt{X^{*2} + (Y^* - 1)^2}}{X^* (Y^* - 1)}$   
WHEN  $X^* \neq 0$
8.  $F = \frac{1}{Y^* + 1} - \frac{1}{Y^* - 1}$  WHEN  $X^* = 0$
9.  $\sum F \sqrt{1 - \mu^2} \left[ G_{v,A} (a_{0,0} + \mu^2 a_{0,2} + \mu^4 a_{0,4}) + G_{v,B} (a_{1,0} + \mu^2 a_{1,2} + \mu^4 a_{1,4}) \right] = 1/40$
10.  $\sum F \sqrt{1 - \mu^2} \left[ G_{v,A} (a_{0,0} + \mu^2 a_{0,2} + \mu^4 a_{0,4}) + G_{v,B} (a_{1,0} + \mu^2 a_{1,2} + \mu^4 a_{1,4}) \right] - 1/40 = R$

$$11. \quad \sum F \left[ G_{v,A} (a'_0 \sqrt{1-\mu^2} + P_a p_{a,0} + P_b p_{b,0}) \right. \\ \left. + G_{v,B} (a'_1 \sqrt{1-\mu^2} + P_a p_{a,1} + P_b p_{b,1}) \right] = -R$$

$$12. \quad P_a = 0.0478104 \sqrt{1-\eta^2} + \frac{5}{\pi} \left[ (0.10-\eta)^2 \text{Log}_e A_a \right. \\ \left. + (0.10+\eta)^2 \text{Log}_e B_a + 2\eta^2 \text{Log}_e C_a \right]$$

$$\text{WHERE: } A_a = \frac{0.9949874 (1-\eta) + 0.9 \sqrt{1-\eta^2}}{|0.9949874 (1-\eta) - 0.9 \sqrt{1-\eta^2}|}$$

$$B_a = \frac{0.9949874 \sqrt{1-\eta^2} + 0.9 (1-\eta)}{|0.9949874 \sqrt{1-\eta^2} - 0.9 (1-\eta)|}$$

$$C_a = \frac{\sqrt{1-\eta^2} - (1-\eta)}{\sqrt{1-\eta^2} + (1-\eta)}$$

$$13. \quad P_b = 0.0960329 \sqrt{1-\eta^2} + \frac{5}{2\pi} \left[ (0.20-\eta)^2 \text{Log}_e A_b \right. \\ \left. + (0.20+\eta)^2 \text{Log}_e B_b + 2\eta^2 \text{Log}_e C_b \right]$$

$$\text{WHERE: } A_b = \frac{0.9797959 (1-\eta) + 0.8 \sqrt{1-\eta^2}}{|0.9797959 (1-\eta) - 0.8 \sqrt{1-\eta^2}|}$$

$$B_b = \frac{0.9797959 \sqrt{1-\eta^2} + 0.8 (1-\eta)}{|0.9797959 \sqrt{1-\eta^2} - 0.8 (1-\eta)|}$$

$$C_b = \frac{\sqrt{1-\eta^2} - (1-\eta)}{\sqrt{1-\eta^2} + (1-\eta)}$$

$$14. \quad F_0(\eta) = \sqrt{1-\eta^2} (a_{0,0} + \eta^2 a_{0,2} + \eta^4 a_{0,4}) \\ + \sqrt{1-\eta^2} a'_0 + P_a p_{a,0} + P_b p_{b,0}$$

$$15. \quad F_1(\eta) = \sqrt{1-\eta^2} (a_{1,0} + \eta^2 a_{1,2} + \eta^4 a_{1,4}) \\ + \sqrt{1-\eta^2} a'_1 + P_a p_{a,1} + P_b p_{b,1}$$

$$16. \frac{C_{L\alpha}}{AR} = \frac{\pi^2}{8} \left[ 8(a_{0,0} + \frac{1}{2}a_{1,0}) + 2(a_{0,2} + \frac{1}{2}a_{1,2}) + (a_{0,4} + \frac{1}{2}a_{1,4}) \right. \\ \left. + 8(a'_0 + \frac{1}{2}a'_1) + 0.50888(p_{a,0} + \frac{1}{2}p_{a,1}) \right. \\ \left. + 1.01517(p_{b,0} + \frac{1}{2}p_{b,1}) \right]$$

$$17. \frac{C_{L\alpha}}{C_{L\alpha}} = \frac{4\pi AR}{C_{L\alpha}} \left[ F_0(\eta) + \frac{1}{2}F_1(\eta) \right]$$

$$18. \eta_{CP} = \frac{4\pi AR}{C_{L\alpha}} \left[ \frac{1}{3}(a_{0,0} + \frac{1}{2}a_{1,0} + a'_0 + \frac{1}{2}a'_1) \right. \\ \left. + \frac{2}{15}(a_{0,2} + \frac{1}{2}a_{1,2}) + \frac{8}{105}(a_{0,4} + \frac{1}{2}a_{1,4}) \right. \\ \left. + 0.01595(p_{a,0} + \frac{1}{2}p_{a,1}) + 0.03229(p_{b,0} + \frac{1}{2}p_{b,1}) \right]$$

$$19. \frac{x_{ac}}{c} = \frac{1}{4} \frac{F_0(\eta) + F_1(\eta)}{F_0(\eta) + \frac{1}{2}F_1(\eta)}$$

$$20. \frac{\bar{x}_{ac}}{C_R} = \frac{1-\lambda}{1-\alpha} \eta_{CP} + \frac{4\pi AR}{C_{L\alpha}} \left\{ \left( \frac{\pi}{16} - \frac{1-\lambda}{12} \right) (a_{0,0} + a_{1,0} + a'_0 + a'_1) \right. \\ \left. + \left( \frac{\pi}{64} - \frac{1-\lambda}{30} \right) (a_{0,2} + a_{1,2}) + \left[ \frac{\pi}{128} - \frac{2(1-\lambda)}{105} \right] (a_{0,4} + a_{1,4}) \right. \\ \left. + \frac{1}{4} \left[ 0.04996 - 0.01595(1-\lambda) \right] (p_{a,0} + p_{a,1}) \right. \\ \left. + \frac{1}{4} \left[ 0.09967 - 0.03229(1-\lambda) \right] (p_{b,0} + p_{b,1}) \right\}$$

$$21. \frac{ARC_{D_i}}{C_L^2} = 4 \left( \frac{AR}{C_{L\alpha}} \right)^2 \int_0^1 \frac{\omega}{V} \left( \frac{\Gamma}{4SV} \right) d\eta$$

$$22. \frac{\Gamma}{4SV} = \pi \left( F_0 + \frac{1}{2} F_1 \right)$$

$$23. \frac{\omega}{V} = \left( \frac{\omega}{V} \right)_V + \left( \frac{\omega}{V} \right)_P$$

$$24. \left( \frac{\omega}{V} \right)_P = \pi \left( p_{a,0} + \frac{1}{2} p_{a,1} \right) \left( \frac{\omega}{V} \right)_{P_a} + \pi \left( p_{b,0} + \frac{1}{2} p_{b,1} \right) \left( \frac{\omega}{V} \right)_{P_b}$$

$$25. \left( \frac{\omega}{V} \right)_V = \frac{1}{\sin \phi} \sum_{n=1}^3 (2n-1) A_{2n-1} \sin (2n-1) \phi$$

$$26. \sin \phi = \sqrt{1-\eta^2}$$

$$27. A_1 = \pi \left( a_{0,0} + \frac{1}{2} a_{1,0} + \frac{1}{4} a_{0,2} + \frac{1}{8} a_{1,2} \right. \\ \left. + \frac{1}{8} a_{0,4} + \frac{1}{16} a_{1,4} + a'_0 + \frac{1}{2} a'_1 \right)$$

$$28. A_3 = \pi \left( \frac{1}{4} a_{0,2} + \frac{1}{8} a_{1,2} + \frac{3}{16} a_{0,4} + \frac{3}{32} a_{1,4} \right)$$

$$29. A_5 = \frac{\pi}{16} \left( a_{0,4} + \frac{1}{2} a_{1,4} \right)$$

Note 1:

For the twelve values of  $X^*$  and  $Y^*$  calculated using the tip correction vertices ( 0.9625),  $X^*$ ,  $Y^*$ , and  $F$  (equations 5,6,7, and 8 respectively) must be multiplied by four.

- Note 2: Equations 12, 13, 14, 15, 17, and 19 which are functions of  $\eta$  are calculated for eleven values of  $\eta$  incremented equally from  $\eta$  equal 0 to  $\eta$  equal one.
- Note 3: Values of the Simpson Factors used in the integration in equation 21 found in Table 1.
- Note 4: Values of  $(W/V)_{P_a}$ , and  $(W/V)_{P_b}$  used in equation 24 are found in Table 1.

The general scheme of calculation is as follows. For a given wing planform defined by  $a$ ,  $\lambda$ , and  $\epsilon$ , (Figure 1), and a given control point with generalized coordinates  $(\nu_p, \eta_p)$ , (Figure 2), and a vortex with generalized coordinates  $(\nu_0, \mu)$ , (Figure 2), calculate equations through 9. Repeat this calculation for each of the 126 vortices representing the wing vorticity, summing at equation 9. This series of calculations will result in an equation (equation 9), in six variables, the Falkner "a" functions. Repeat this procedure for each of the control points, 1 through 6, (Figure 2), and the result is a set of six equations in the six "a" functions. This set may be solved for the Falkner "a" functions.

The solution for the "a" functions (derived by writing the downwash equations at control points 1 through 6), does not satisfy the boundary conditions at control points 7 through 10 because of wing planform slope discontinuities. The discrepancy may be determined by writing the downwash equations for points 7 through 10 and using the "a" functions in equation 10.

Using the same set of vortices, and control points 1, 2, 7, 8, 9, 10, calculate equations 1 through 8, 10, 11, 12, and 13, summing, for each control point, at equations 10 and 11. This results in a set of six equations (equation 11) in the six "p" functions, which may be solved for the Falkner "p" functions.

It may be noted that the calculation for control points 1 and 2 has been repeated. This was done in order that six equations could be formed. The residual, or loading error,  $R$ , (equation 10), should be zero for control points 1 and 2.



At this point, six "a" functions and six "p" functions have been calculated. These calculations have taken about 92% of the total time of calculation of this problem. What remains is the calculation of the wing aerodynamic characteristics, using the "a" and "p" functions.

Given the Falkner "a" and "p" coefficients, equations twelve through twenty are calculated. They give the lift curve slope (equation 16), the spanwise loading (equation 17), the spanwise center of pressure, (equation 18), the local aerodynamic center (equation 19), and the aerodynamic center of the complete wing (equation 20). The spanwise center of pressure and the local aerodynamic center are calculated for eleven span positions from the root to the wing tip, represented by values of  $\eta = 0, .1, .2, .3, .4, .5, .6, .7, .8, .9,$  and 1.0.

The induced drag is obtained from equation 21. The integration indicated is performed using a Simpson integration procedure. Since the rate of change of the integral becomes large in the vicinity of the wing tip (large  $\eta$ ) special revised Simpson factors are used. These factors were calculated by Falkner and are tabulated in Table 1. The  $(W/V)_{P_a}$  and  $(W/V)_{P_b}$  used in equation 24 are included in Table 1.

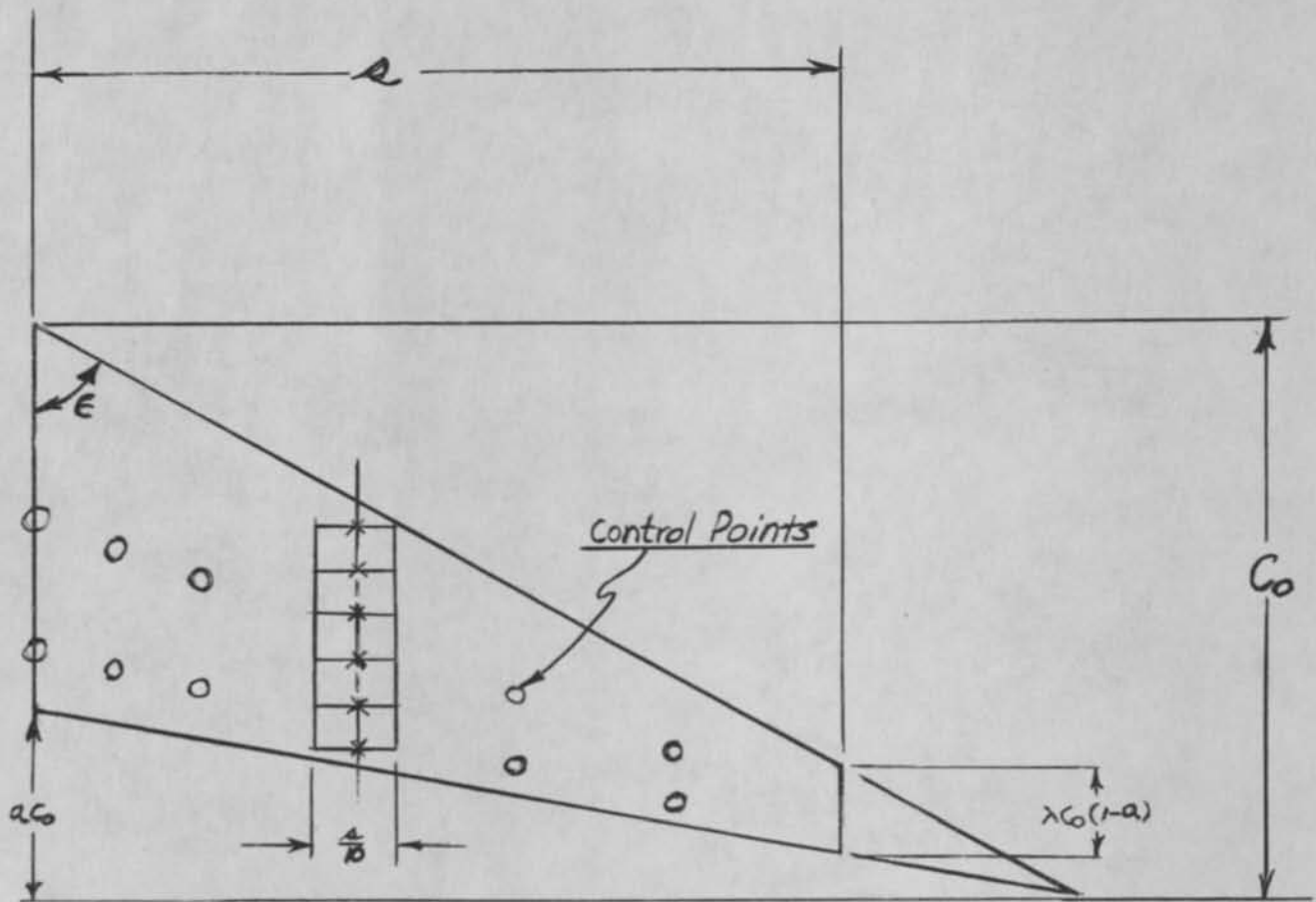


Figure 1 Diagram of Wing Planform Parameters

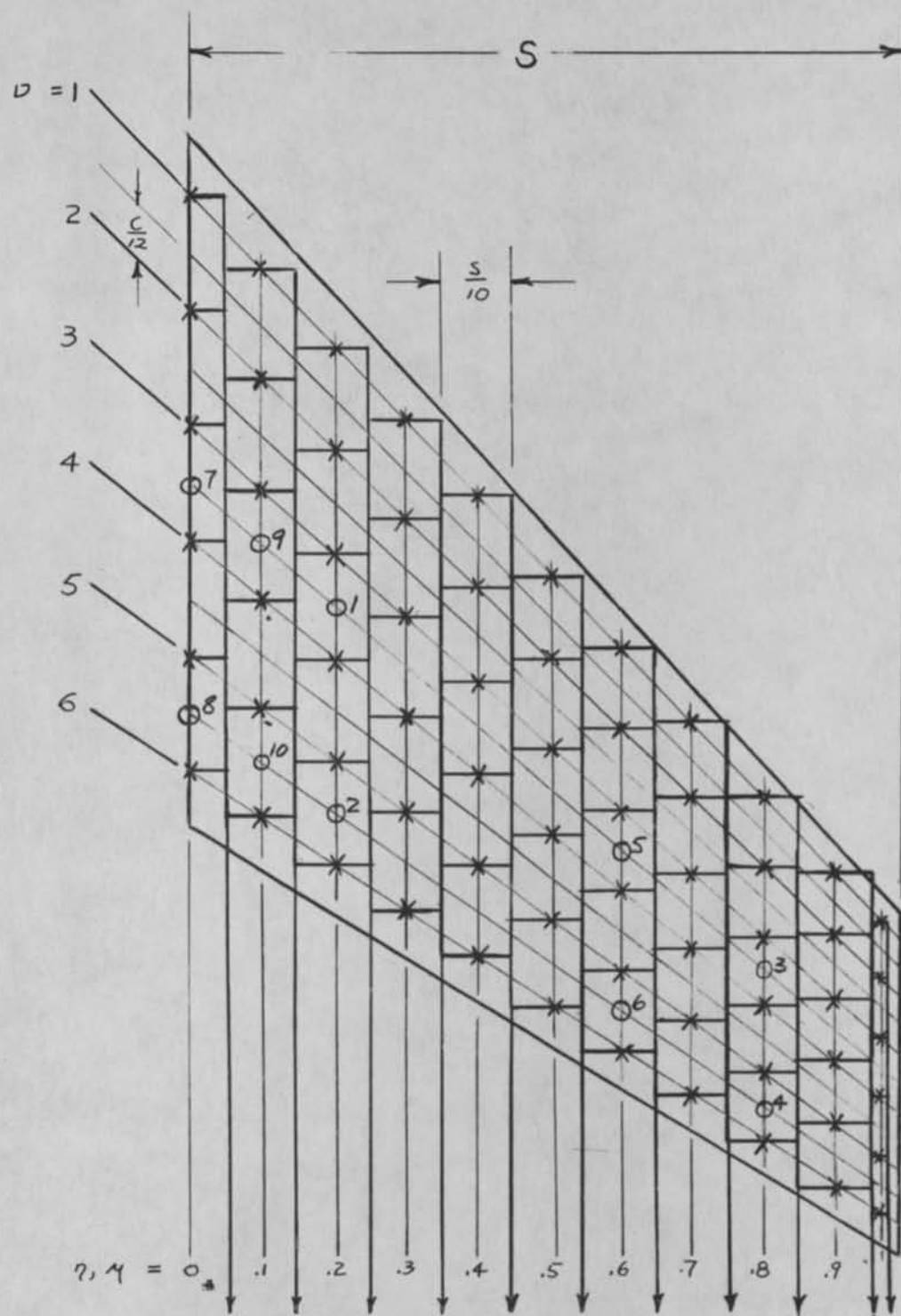


Figure 2 Diagram of Vortex Lattice and Control Points

Table 1.

P-circulation, Downwash, and Modified Simpson Factors

$\eta$	$(W/V)_{P_a}$	$(W/V)_{P_b}$	Simpson Factors
0.00	1.0	1.00	1
0.05	0.5	0.75	4
0.10	0.0	0.50	2
0.15	0.0	0.25	4
0.20	0.0	0.00	2
0.25	0.0	0.00	4
0.30	0.0	0.00	2
0.35	0.0	0.00	4
0.40	0.0	0.00	2
0.45	0.0	0.00	4
0.50	0.0	0.00	2
0.55	0.0	0.00	4
0.60	0.0	0.00	2
0.65	0.0	0.00	4
0.70	0.0	0.00	2
0.75	0.0	0.00	4
0.80	0.0	0.00	2
0.85	0.0	0.00	4
0.90	0.0	0.00	1.800
0.95	0.0	0.00	4.525
1.00	0.0	0.00	0.675

## PROGRAM

Programming of this problem was begun shortly before Convair received the 701. The program was written in the IBM Speedco System. There were three main reasons for this choice of coding system. First, preliminary investigations of the calculations involved showed that for the input data which was to be used, there was large variation in some of the calculated values with different problems. Some quantities, if scaled for maximums, would result in only two significant digits for the minimums. Since it would be obviously impractical to scale differently for different problems, it was felt that at least some of the calculations should be written in a floating point system. At this time there were no floating point sub-programs available at Convair. The second reason was the decision that the problem should be done as quickly as possible and in as simple a form as possible. This qualified Speedco in several ways. Time would not be required to write floating-point sub-routines, the time required to scale the fixed point operations would be saved, and the programming would be carried out in the familiar three-address arithmetic. The third reason was incidental to the first two. In anticipation of other problems to use Speedco, it was necessary to become familiar with the system as soon as possible.

As a consequence of the use of Speedco, the available ES storage was limited to 713 positions. This necessitated the division of the program into three major blocks, the F program, the M program and the G program. The F program brings in the problem data via the card reader, computes the a and the p coefficients, brings the M and G programs off the drums, and is the program which controls the flow of the entire problem. The M program takes the coefficients of the a's and the p's and computes the a and p coefficients using a standard elimination method to reduce the matrices. The G program takes the a and p coefficients and computes and prints out the required results.

The flow of the problem is as follows: The M program is loaded onto drum; the G program is loaded onto drum; the F program is loaded into ES followed by the data for the first problem. The F program computes the 42 quantities required for the solution of the six equations in the Falkner a coefficients and then dumps all of ES except that occupied by Speedco onto a tape (for availability in case of machine error). The F program then calls the M program from drum and transfers control to the M program. The M program computes the a coefficients and transfers control back to the F program which then calculates the required matrix elements for the p functions. Once again the ES is placed on tape (with the exception of that occupied by Speedco). The F program transfers control to the M program which computes the p coefficients and transfers.

control back to the F program. The F program reads the G program off the drum and into the ES occupied by the M program and transfers control to the G program. The G program computes and prints out the results and transfers control to the F program which repeats the cycle by activating the card reader for the next set of problem data.

This flow is developed in more detail in figures 4,5, and 6 and in figure 3.

The data for this problem is in two groups, the problem data and the program data. Because the theory imposes the same vortex network on all surfaces, it is possible to reduce the problem data to three quantities,  $a$ ,  $\lambda$ , and  $\epsilon$ , which define uniquely the shape of the wing according to the figure 5. The coordinates for the vortices and control points are loaded previous to the first problem and serve for all problems. These coordinates might have been generated by the program, but the programs for the generating functions would have taken approximately the same amount of ES, so they were loaded as program constants, simplifying the program.

The breakdown of storage is as follows; problem data, 3; program data, 110; erasable, 100 (this includes the print area of 85); program F, 230; program G, 257; program M, 75.

The Speedco checking system was used for all of the arithmetic operations, except the summation operations. After running problems with this program to the extent of more than

15 machine hours, it was noted that at no time had the program stopped as a result of an error within the checking loop but that in all cases of machine error, the program was damaged badly enough to cause the machine error to become obvious because of tight loops, stops, etc. Therefore, the Speedco check was removed from the program which cut the time per problem from approximately 15 minutes to 10 minutes. However, at the same time, it was decided that the machine was making a sufficiently large number of errors that it would be advisable to incorporate a set-back routine to cut the possible maximum time loss for a given problem. It had been noticed that in several cases Speedco had failed at times when the program was very close to the print-out stage. In these cases it was felt that very little time could be saved by searching the Speedco program for possible manual set-back or correction procedures and so, unless the stop was a well defined program stop, the problem was started over, resulting in a loss of almost 15 (or 10) minutes. This can be rather costly in terms of machine time, particularly if the machine is not behaving exactly as it should.

For this reason, the entire usable ES is placed on tape at two points in the course of the calculation; at the end of the calculation of the "a" equation coefficients (or about time 4 minutes) and similarly after calculation of the "p" equation coefficients (about time 9 minutes). Thus, although



time may still be lost due to machine error, the possible maximum has been cut from about ten minutes to around five. The second dumping has the following "bonus" benefit. When one set of results are printed out, they are almost immediately erased in ES by other numbers. If the operator has not set up the printer correctly (the wrong board, carriage tape, alteration switches, etc.), the final portions of the program may be repeated (from the last dump - including the print-out) with the loss of only around a minute of machine time.

The ES on tape is called back (in case of trouble) by a four card deck which is loaded after resetting memory and which is followed in the card hopper by the data cards for those problems not completed.

Prior to print-out, there is no check on the calculations. At print-out there is a visual check of the results by the operator to insure that certain key results lie within pre-determined limits. If they do not, the problem is rerun from the last dump. If the results remain the same, the problem is rerun from the beginning. As a second check certain results are plotted as soon as possible. This will almost always catch errors in the output data or results of this problem. Luckily, over 95% of the machine errors in the running of the program have been spotted as unfamiliar situations on the control panel or as errors in the results which were obvious to the operator.

The matrix reduction used here was not a sub-routine but was programmed specifically for this program. Because of this it was possible to compress it into a small portion of ES.

The program makes use of an instruction not in the standard Speedco list (although it is now standard at Convair). The "Read Card Reader" instruction transfers control to the Speedco loading program which reads any cards which may be ready in the card reader. This loading is terminated by the standard Speedco termination card which transfers control to Speedco location 300. This instruction allows the program to load only as much data as is required for the current problem and therefore the operator is given a check on the progress of the program which is independent of the printed results.

When the program was completely checked out, it was loaded and ES from 1200 to 4056 was punched in binary, resulting in a binary deck of around 65 cards of instructions and program data. This contrasts very favorably with the more than 700 decimal Speedco cards.

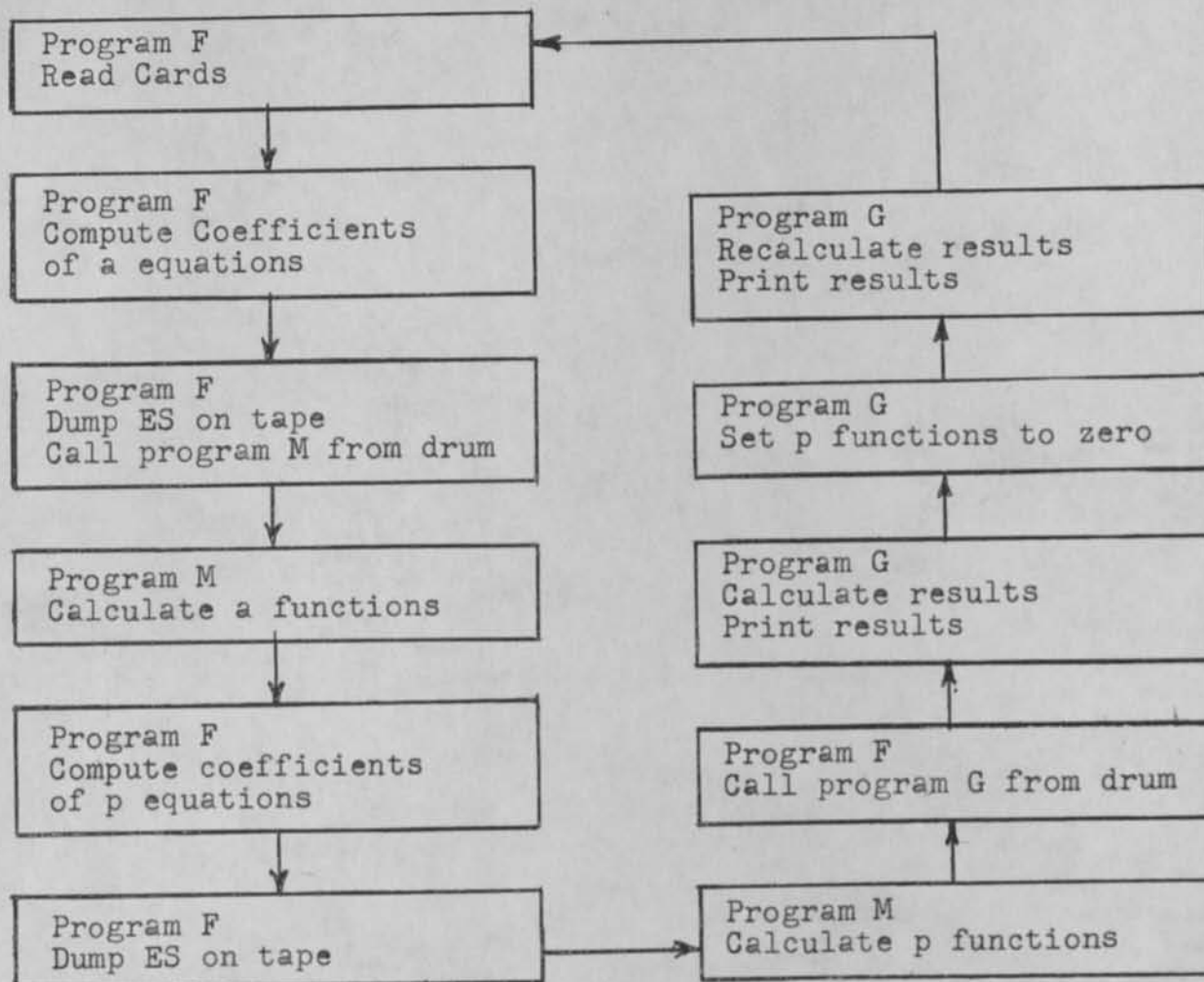


Figure 3 Generalized Flow Chart of Computation Procedure

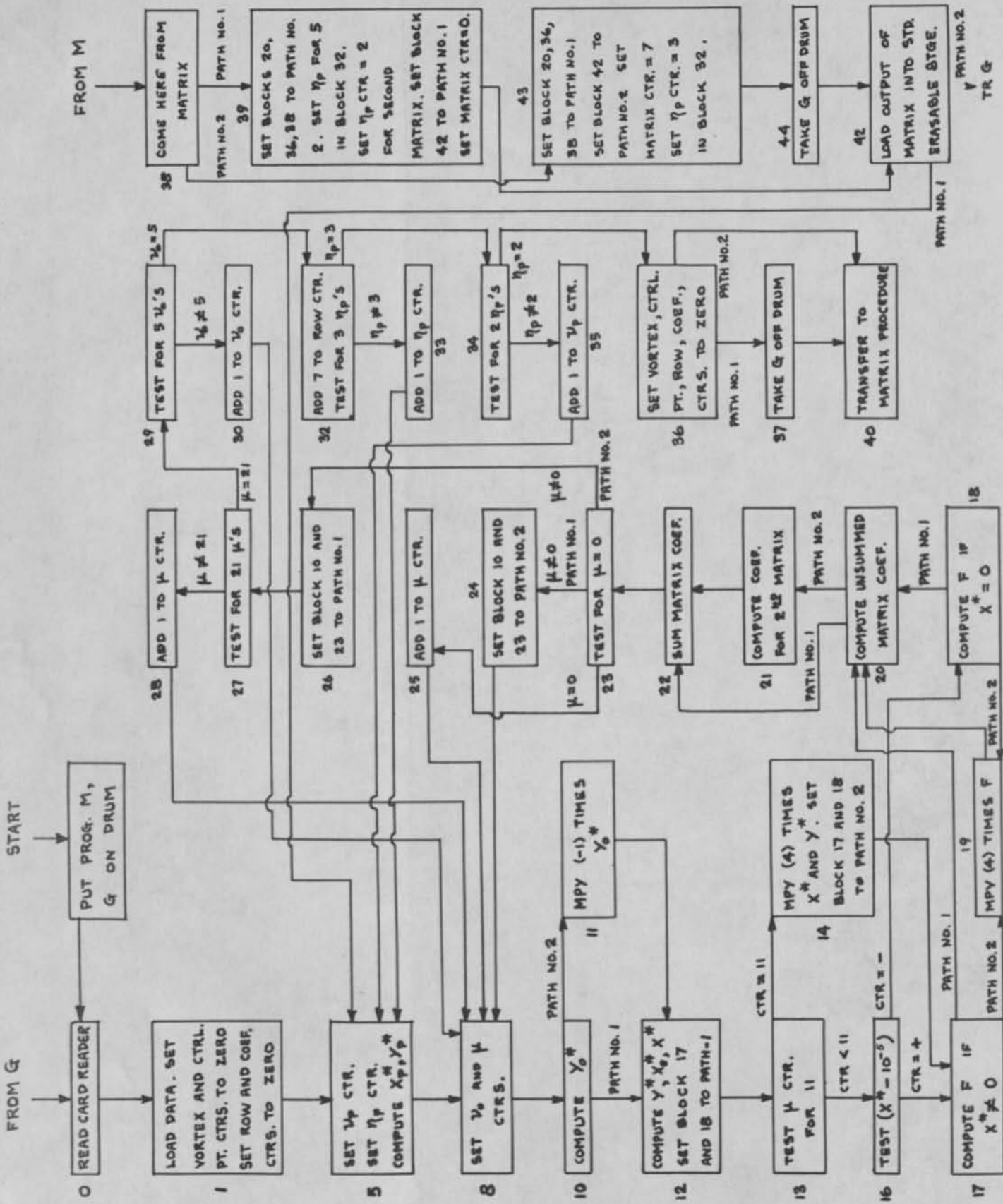


FIGURE - 4 FLOW CHART FOR COMPUTATION PROGRAM F

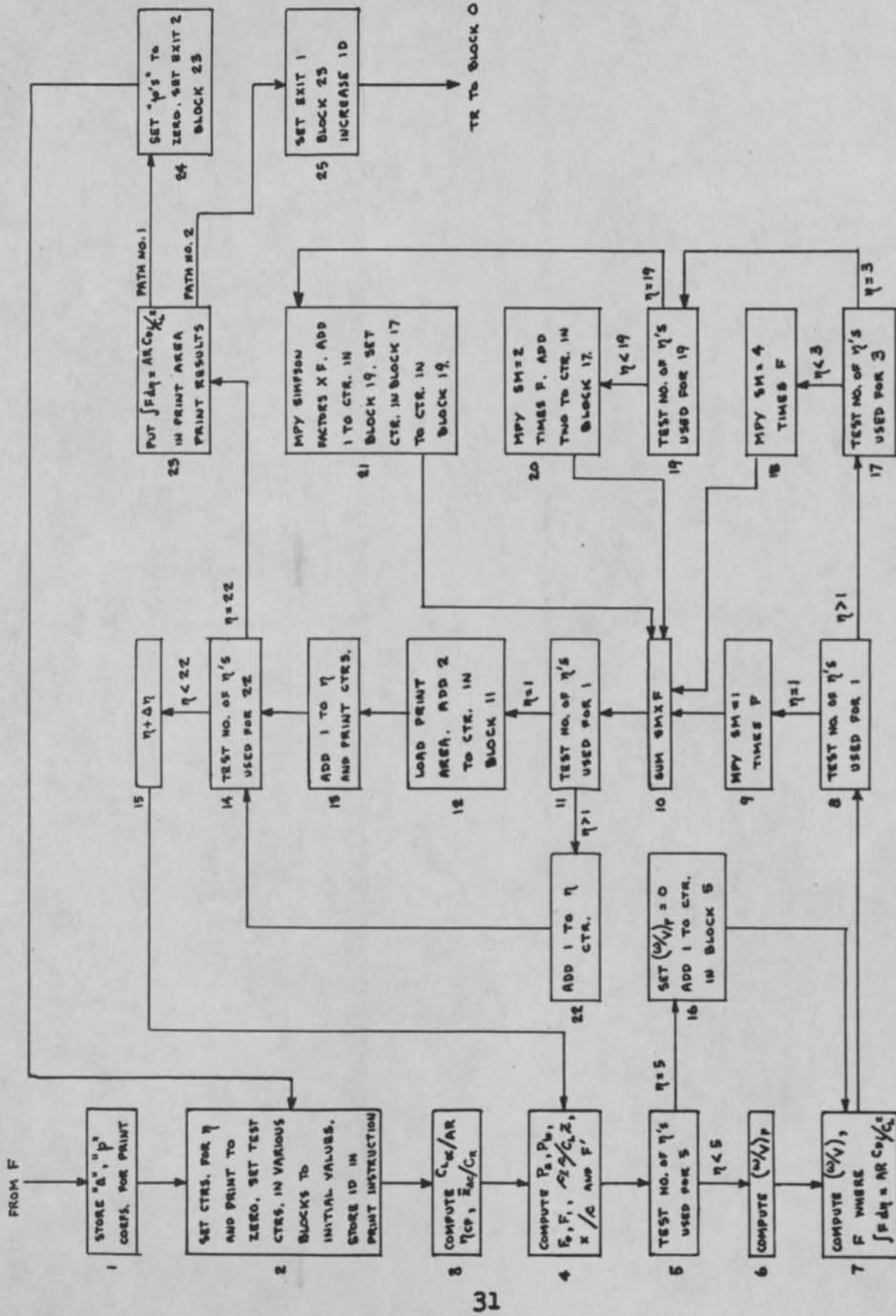
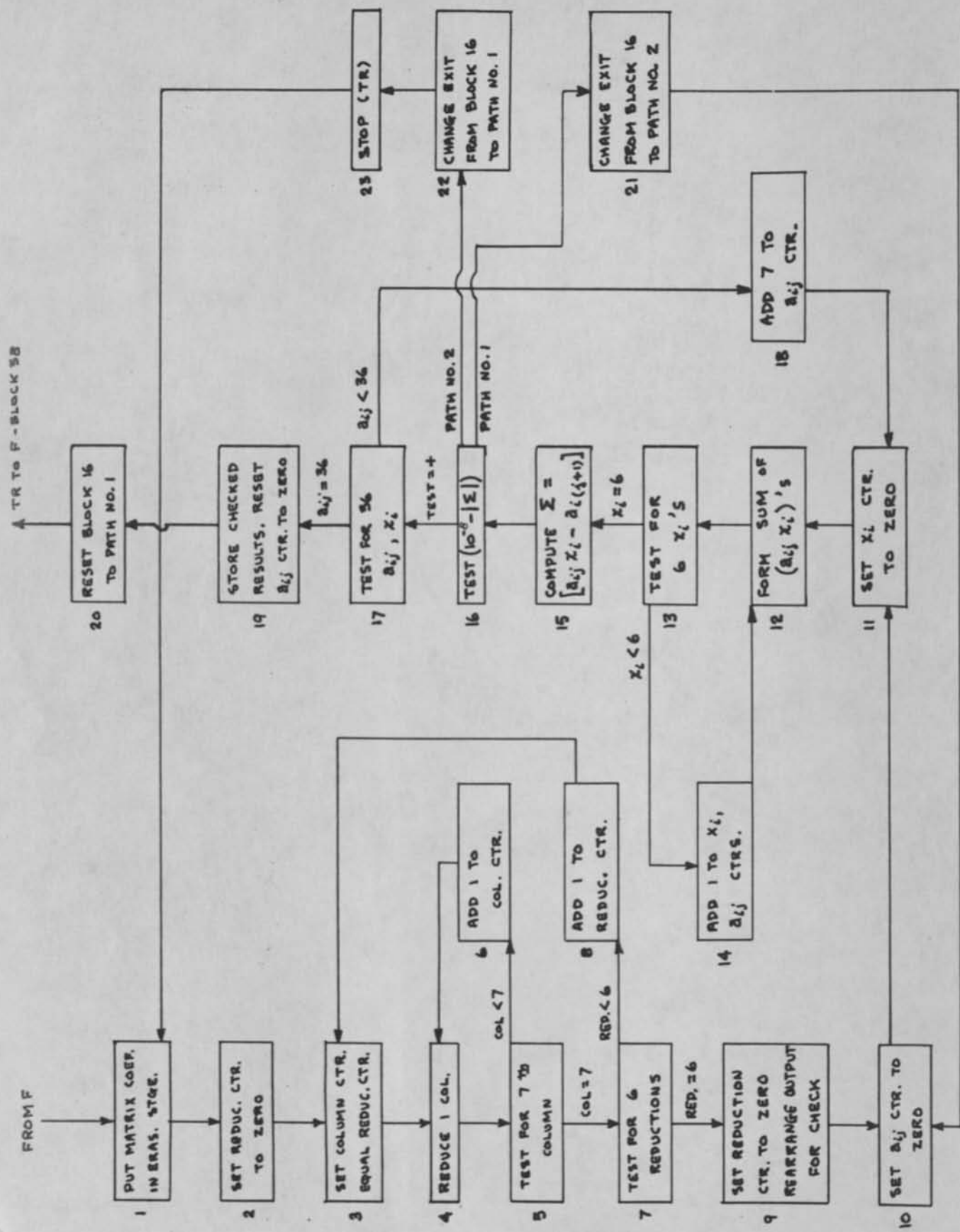


FIGURE - 5 FLOW CHART FOR COMPUTATION PROGRAM G



32

FIGURE -6 FLOW CHART FOR COMPUTATION PROGRAM M

## RESULTS

A discussion of the results of this project may be divided into two sections, the application of the 701 to the problem, and the degree of success of the theory.

The 701 program is assumed correct; the word assumed is used because there conceivably are data combinations, not yet used, which would result in unforeseen situations, not considered in the programming. The program was never checked, digit for digit, with any standard solution, and for this reason, also, there is a possibility of undetected program errors which, for the solutions used for comparison, would result in small or insignificant errors. There were three set of solutions available for comparison and checking.

One; a set of solutions was published by Falkner which could not be compared exactly because of the fact that they were calculated using eight control points instead of the ten used at Convair. In addition to this, they were calculated, to a large extent, on desk calculators, probably carrying less than the ten digits used by Speedco.

Second; a solution was calculated, at Convair, using the ten control points, on desk calculators. This solution also carried less than ten significant digits and hence, could not be checked against the 701 solution.

Third; a problem was run on the Convair CPEC's carrying eight digits in a floating decimal scheme. This problem was

run four times resulting in three different sets of results. The two sets which agreed were identical to eight places in about 75% of the results, with the other 25% showing significant discrepancies in the sixth, seventh, and eighth places.

The 701 solution, when all of the "Bugs" were removed, checked all three of these solutions (that is, the Falkner, the Convair hand solution, and the CPC solution) to at least three significant digits in the final results. In addition to this it checked the CPC results, final and intermediate, to four, and in most cases, five or more significant digits. All of the arithmetic operations had been checked in detail, and the logic operations had been checked as carefully as possible. These considerations led to the belief that the program was correct.

Insofar as the numerical methods are concerned, it appears that the ten significant digits carried are sufficient to produce good results. In the case of some wings of low  $\epsilon$ , the calculation of  $X^*$  results in the subtraction of two numbers of the order of thousands, resulting in an  $X^*$  of the order of only two or three significant digits. But this occurs only for four or six of the vortices in a set of 126, and so, cannot be regarded too seriously.

How successfully the theory obtains the aerodynamic characteristics is a question which cannot be answered as readily. It is difficult to make quantitative comparisons



among the various theories. Probably the best, ultimate test is the comparison of the theory with experimental results.

The decision made at Convair, with respect to the theory, is; a. the Falkner theory probably does a better job, for the wings considered, than any of the existing theories with the exception of the Multhopp theory; b. the modified 10 point solution, (involving the root correction control points), for most cases, results, in better correspondence than the standard 6 point solution with the more exact Multhopp theory and with experimental results and some theoretical results; c. the Falkner theory, in common with the other theories, is not too good when applied to wings with pointed tips.

In Figures 8,9, and 10 are comparisons of the local aerodynamic center with the vortex lattice 6 point, the vortex lattice 10 point, and the Multhopp theories. It might be noted that as the amount chopped off of the tip increases, the three curves more closely approach each other, even though the aspect ratio is decreasing.

In tables 2,3, and 4, the spanwise loading is compared using the three previously named and Weissinger theory. In this case all of the values are close and in the case of the more outboard stations, the plots of these curves (with the possible exception of the vortex lattice 6 point solution) become indistinguishable. Tables 5 and 6 give similar comparisons of the complete wing aerodynamic center and the lift curve slope, respectively.

The values obtained by the Weissinger and Multhopp theories used in this comparison were taken from reference 4. The Multhopp values were assumed the most correct.

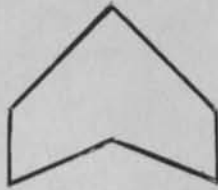


Figure 7a Wing A  
 $a=3/7$ ,  $\lambda=5/9$ ,  $\epsilon=45^\circ$ ,  $AR=1.714$

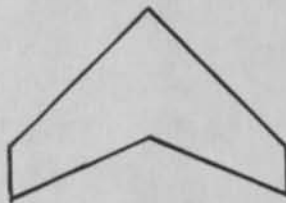


Figure 7b Wing B  
 $a=3/7$ ,  $\lambda=7/18$ ,  $\epsilon=45^\circ$ ,  $AR=2.640$

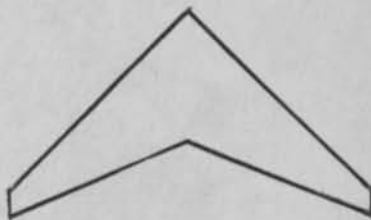
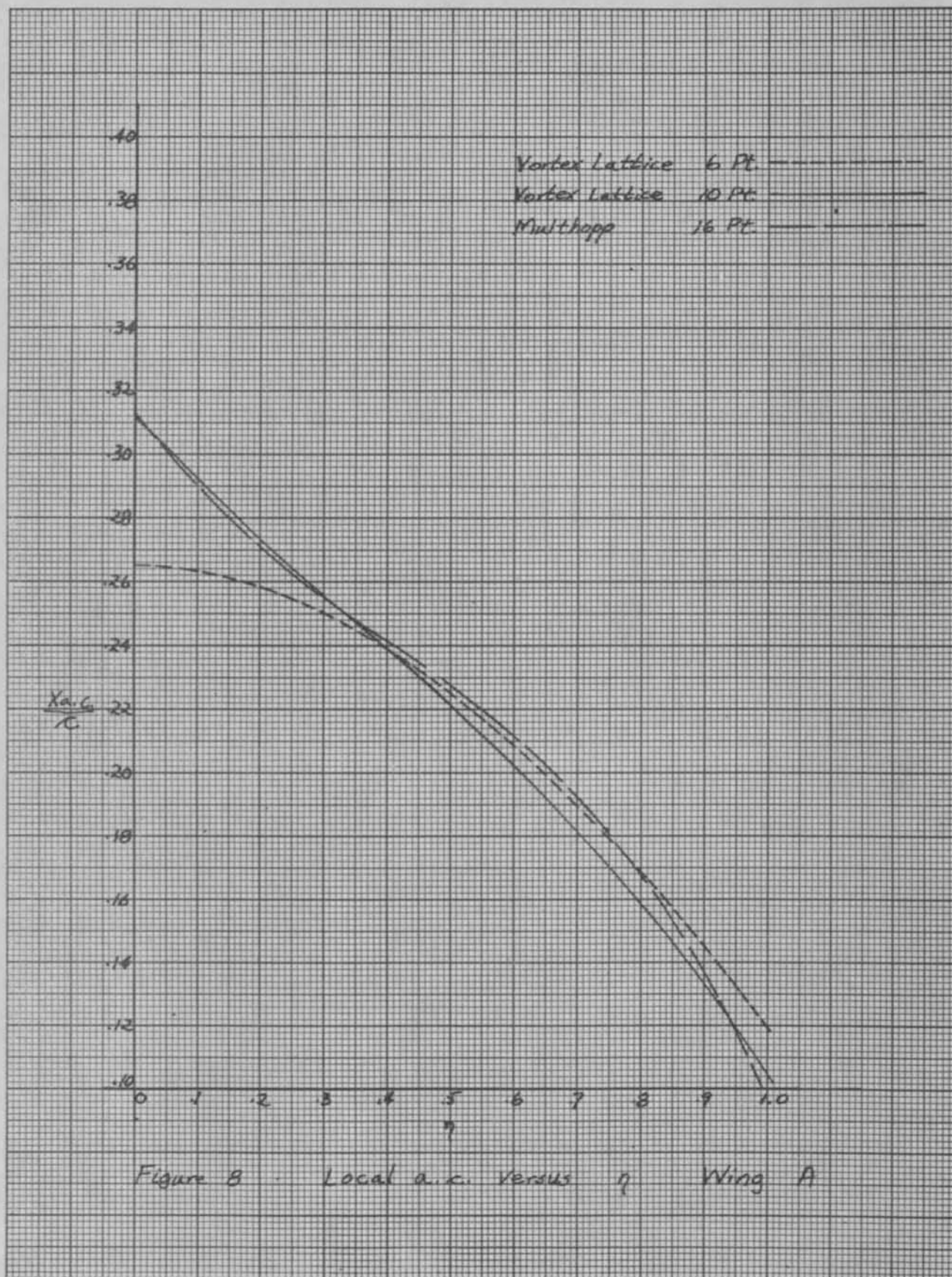
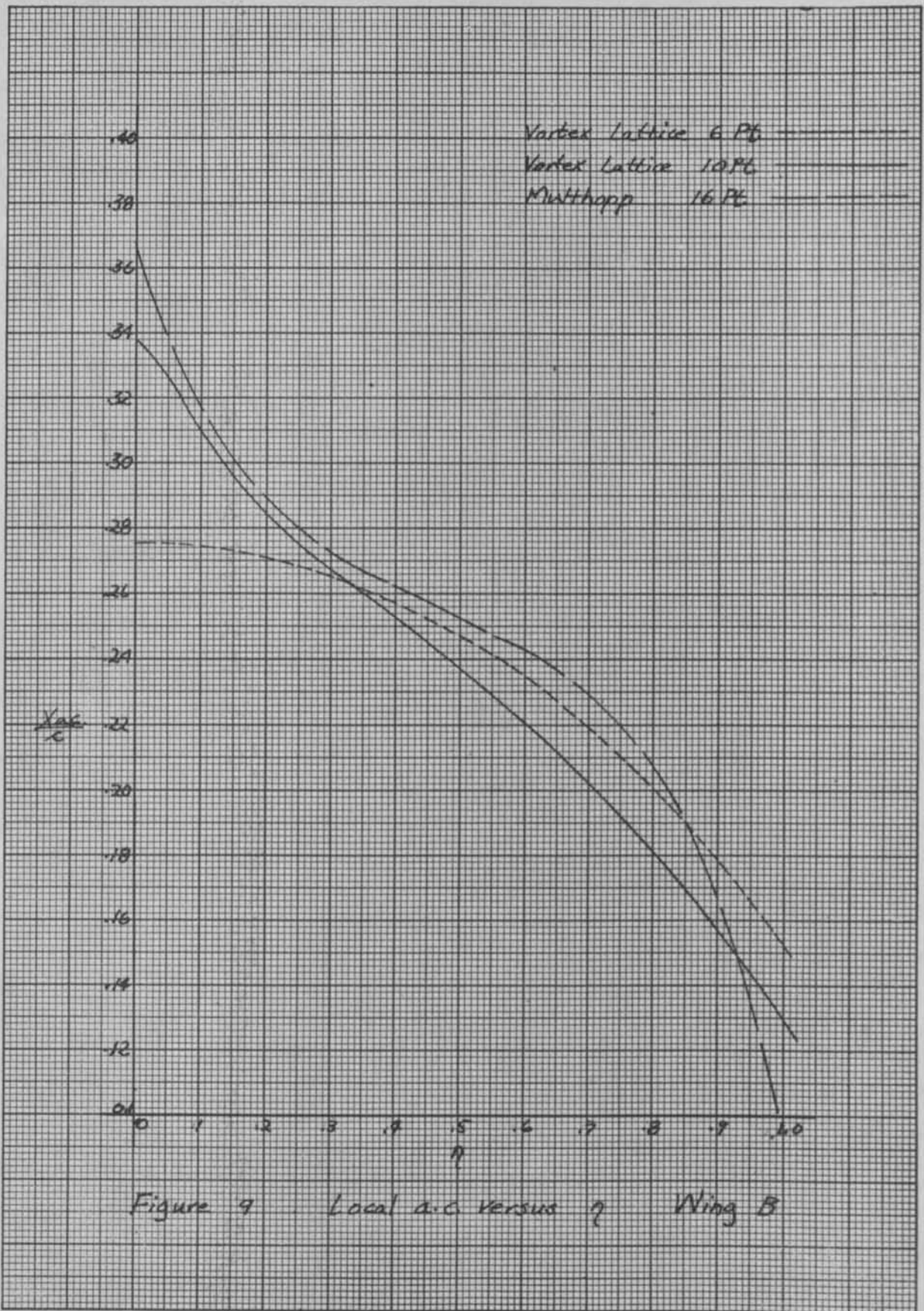


Figure 7c Wing C  
 $a=3/7$ ,  $\lambda=2/9$ ,  $\epsilon=45^\circ$ ,  $AR=3.818$

Figure 7 Diagram of Example Wings





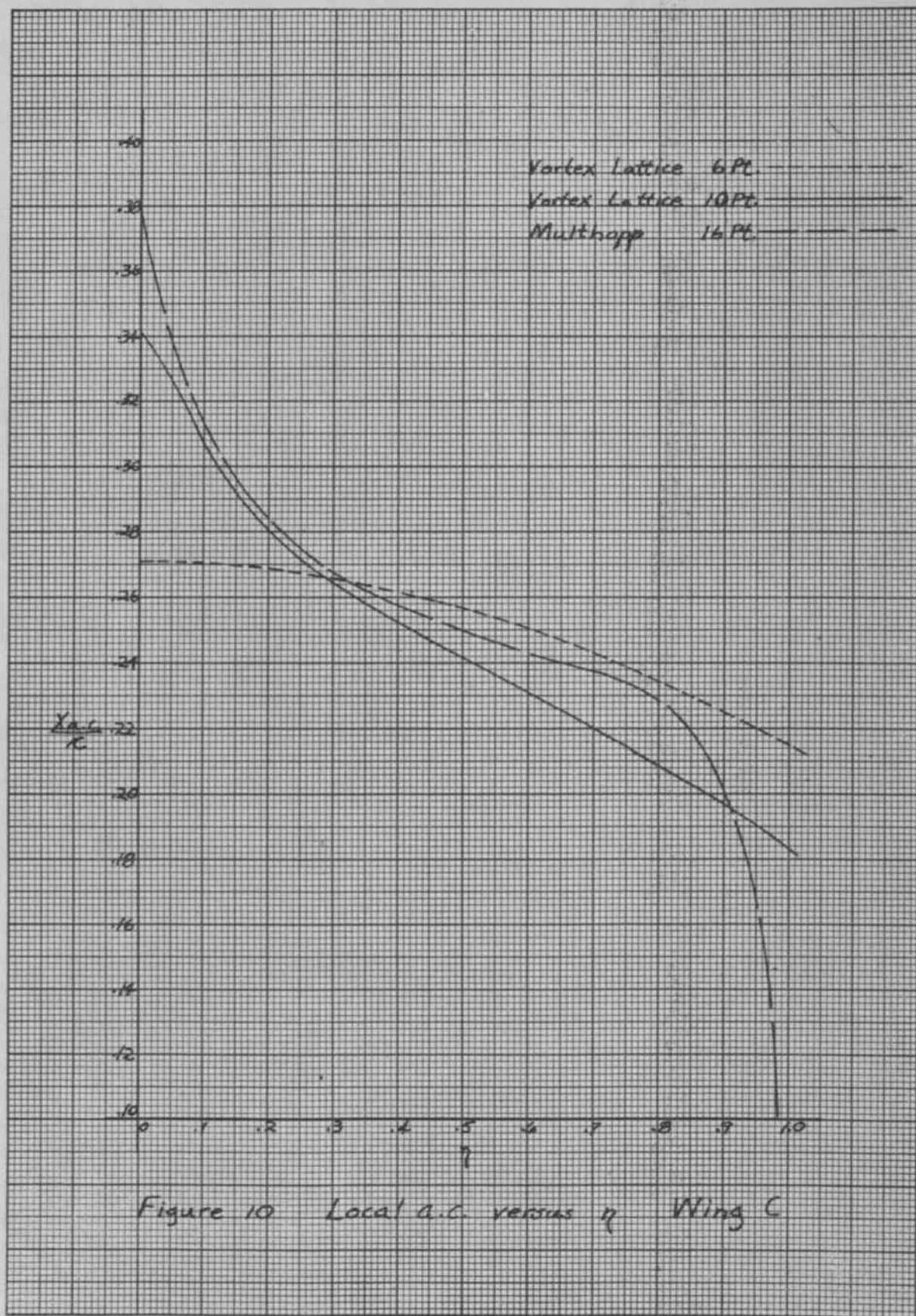


Figure 10 Local a.c. versus η Wing C

TABLE 2  $\frac{C_{L,C}}{C_{L,E}}$  Wing A

$\eta$	Vortex Lattice 6 point	Vortex Lattice 10 point	Multhopp 16 point	Weissinger 4 point
0.0	1.239	1.224	1.235	1.227
0.1	1.235	1.225	1.231	1.222
0.2	1.221	1.218	1.222	1.213
0.3	1.197	1.197	1.201	1.196
0.4	1.160	1.162	1.173	1.161
0.5	1.108	1.111	1.120	1.110
0.6	1.034	1.038	1.044	1.040
0.7	0.933	0.937	0.944	0.944
0.8	0.791	0.795	0.808	0.808
0.9	0.579	0.582	0.598	0.598
1.0	0.000	0.000	0.000	0.000

TABLE 3  $\frac{C_{L,C}}{C_{L,E}}$  Wing B

$\eta$	Vortex Lattice 6 point	Vortex Lattice 10 point	Multhopp 16 point	Weissinger 4 point
0.0	1.223	1.195	1.208	1.200
0.1	1.218	1.202	1.212	1.203
0.2	1.206	1.200	1.204	1.198
0.3	1.183	1.183	1.188	1.188
0.4	1.150	1.153	1.153	1.161
0.5	1.102	1.107	1.109	1.120
0.6	1.036	1.043	1.041	1.050
0.7	0.944	0.951	0.949	0.948
0.8	0.812	0.819	0.816	0.816
0.9	0.606	0.611	0.609	0.609
1.0	0.000	0.000	0.000	0.000

TABLE 4  $\frac{C_{Lc}}{C_{L\bar{c}}}$  Wing C

$\eta$	Vortex Lattice 6 point	Vortex Lattice 10 point	Multhopp 16 point	Weissinger 4 point
0.0	1.257	1.217	1.220	1.211
0.1	1.250	1.227	1.230	1.210
0.2	1.231	1.223	1.225	1.202
0.3	1.199	1.199	1.199	1.189
0.4	1.154	1.158	1.158	1.164
0.5	1.094	1.102	1.102	1.120
0.6	1.018	1.028	1.028	1.033
0.7	0.920	0.930	0.930	0.930
0.8	0.788	0.798	0.798	0.798
0.9	0.589	0.597	0.597	0.597
1.0	0.000	0.000	0.000	0.000



TABLE 5  $\bar{x}_{ac}/c$ 

	Wing A	Wing B	Wing C
Vortex Lattice, 6 point	0.5217	0.6198	0.7579
Vortex Lattice, 10 point	0.5282	0.6563	0.7657
Weissinger, 4 point	0.5385	0.6502	0.7563
Multhopp, 16 point	0.5318	0.6640	0.7714

TABLE 6  $C_{L_{oc}}$ 

	Wing A	Wing B	Wing C
Vortex Lattice, 6 point	2.190	2.810	3.268
Vortex Lattice, 10 point	2.163	2.771	3.217
Weissinger, 4 point	2.088	2.681	3.129
Multhopp, 16 point	2.136	2.735	3.204

## DISCUSSION

This problem was previously programmed at Convair for the Model I and Model II CPC, using a floating decimal board with all of the necessary operations. Because of the large number of operations involved, it was broken down into four separate programs, one to obtain and punch out the coefficients of the Falkner "a's", one to obtain and punch out the coefficients of the Falkner "p's", one a 6 X 6 matrix reduction, and one to obtain and print the aerodynamic characteristics. The first program took approximately six and one half hours, the second took about seven and one half hours, the third about two minutes, and the fourth took about ten minutes making a total running time for one problem of about 14 to 15 hours. It was learned that the results were not reliable, but that in running the machines continuously for 6-8 hours on one set of coefficients, a machine error was almost certain to occur. This meant that it would be necessary to either incorporate a check of some kind, essentially doubling the running time of the problem, or to assume the results were good and hope that in the plotting of the results the errors would be obvious. Because of the time required and because of the uncertainty about the results, it was decided that the problem could not be handled efficiently by the CPC.

However, a problem which was much too large for the CPC turns out to be almost ideal for the 701. The decimal program

and program data cards, once the program has been checked out, are punched out in binary, making a total of 60 to 70 cards to be loaded as the program. (The number is flexible since the program is occasionally modified and the decimal change cards are inserted behind the binary deck). The problem data consists of three numbers on one card, and the output is 34 lines of print. Thus in 10 minutes of running time, one card is read, 34 lines are printed, and less than fifteen seconds are spent in reading drums and writing tape.

The time required may be further contrasted with the required time of 14 days estimated by Falkner to compute one solution. However, the time of ten minutes is somewhat excessive in the sense that the large number of logical operations used by this program are all, of necessity, performed in the Speedco System. This tends to slow down the running. First, the fact that the logic commands must be first interpreted by Speedco and then executed means a delay. Second, it was noted that, for this problem at least, the standard machine commands, or instructions, probably would have led to a more flexible, more natural logic. Unfortunately, time has not been available to reprogram this problem. If it were to be started over again, it would probably be programmed using standard machine instructions and, in addition, using floating point programs for these arithmetic steps which require them. The computing time per problem would certainly drop; how much, has not been estimated.

The program has behaved very nicely in the sense that to date, no unforeseen situations have appeared. Because of its almost complete use of ES and drums, it has become a fair indicator of how well the machine is operating on a particular day.

Because of the uniformity of the results, it has been decided that the theory and the method of calculation might well be applied to the study of further effects. At the present time, the 701 program is being modified to calculate several aerodynamic properties not covered in the original set-up. It appears that the Falkner theory will be a long-range project at Convair.

## REFERENCES

1. V. M. Falkner            The Calculation of Aerodynamic Loading on Surfaces of Any Shape (August, 1943) A.R.C.R. and M., 1910 (British)
2. V. M. Falkner            The Solution of Lifting Plane Problems by Vortex Lattice Theory (29 September 1947) A.R.C. 10, 895 (British)
3. V. M. Falkner            Calculated Loadings Due to Incidence of a Number of Straight and Swept-Back Wings (5 June 1948) A.R.C. 11, 542 (British)
4. H. C. Garner             A Critical Comparison of Four Subsonic Vortex Sheet Theories (12 July 1951) A.R.C. 14, 138 (British)
5. L. J. Kulakowski         Subsonic Aerodynamic Characteristics of Wing Planforms as Given by Application of the Falkner Lifting Surface Theory, Convair FZA-076 (to be published)

NUMERICAL SOLUTION OF THREE SIMULTANEOUS SECOND-ORDER DIFFERENTIAL EQUATIONS  
ARISING IN THE LOW ENERGY MESON THEORY OF THE DEUTERON

---

Harwood G. Kolsky  
Los Alamos Scientific Laboratory  
Los Alamos, New Mexico

---

1. Introduction:

Considerable progress has been made during the past few years toward getting a consistent picture of the structure of the nucleus by use of the meson theory of nuclear forces. In this theory the nucleus is postulated as being composed of neutrons and protons (called nucleons collectively) with very strong, very short-range forces being transmitted between them by particles weighing about 280 times as much as an electron, called pi-mesons. An excellent account of the historical development and status of the theory has been given in a series of lectures by H. A. Bethe (refs. 1,2).

Although the qualitative success of the meson theory is very good, attempts by theoretical physicists to predict accurate quantitative results from the theory have so far led to contradictions. Since the deuteron is the simplest of the compound nuclei, containing only two nucleons, it has naturally been the object of intense research in recent years. Most workers have used the perturbation approach to the theory, that is, the nucleon-nucleon interaction is expanded in a power series of a coupling constant,  $G$ . This approach was very successful when applied to atoms and electromagnetic fields, mainly because their coupling constant is quite small ( $1/137$ ). In meson theory, however, one finds the coupling constant to be the order of  $1/2$ , so the equations are strongly perturbed indeed. In view of this, the fact that serious difficulties arise in using the perturbation method is perhaps not surprising.

An approach which yielded an approximate second-order equation for two particles interacting in a scalar meson field was developed independently by Tamm (ref. 3) and Dancoff (ref. 4). M. Lévy (ref. 5) extended this formalism to include higher order processes involving multiple meson

exchange and pair creation. To incorporate radiative corrections, he used the relativistic two-body equation of Bethe and Salpeter (refs. 6).

From these theoretical considerations, Lévy was able to deduce potential functions which for the low energy case could be substituted in the ordinary Schrödinger equation to give a set of equations for the radial wavefunctions of the deuteron. From these wavefunctions one can compute values for the physically observed properties of the deuteron and thus have a check on the theory. By the suitable selection of the parameters in his potentials, Lévy was able to get rather good agreement with the experimental quantities.

In spite of this apparent success, his work was no sooner published than Lévy's methods fell under attack by other workers in the field. In particular, Klein (ref. 7) felt that the derivations, although probably correct in general, lacked cogency in certain details and completeness. Following a procedure which was as internally consistent as possible, Klein derived a set of potentials which differed from Lévy's in the second and higher terms of the perturbation expansion. The present calculation (ref. 8) was an attempt to check these potentials against the experimental results.

It must be stated in all fairness, that although Klein's potentials were accepted as being more correct, they gained a pyrrhic victory, because it proved to be impossible to fit the experimental quantities using them without the introduction of additional adjustable parameters.

At the present time, there is a widespread feeling among workers in the field that perhaps the whole perturbation approach to the meson theory of nuclear forces is doomed and something much more elegant is needed. As yet no such theory has put in its appearance, although one can be certain that there will be plenty of computing needed to check it when it does come.

## II. The Differential Equations:

The differential equations for the radial wavefunctions ( $u, v, w$ ) of the deuteron may be written as follows for the low energy case:

(See ref. 5)

(A) For the singlet-S state:

$$\frac{d^2 v}{dx^2} = S(x, a) v(x) \quad \text{for } x \geq x_c$$

with boundary conditions  $v(x) = 0$  for  $x \leq x_c$

and asymptotic form  $v(x) \sim 1 + \frac{x}{\mu a_s}$  for large  $x$

(B) For the triplet-S and triplet-D states:

$$\frac{d^2 u}{dx^2} = U(x, a) u(x) + W(x, a) w(x)$$

for  $x \geq x_c$

$$\frac{d^2 w}{dx^2} = W(x, a) u(x) + Y(x, a) w(x)$$

with boundary conditions  $u(x) = w(x) = 0$  for  $x \leq x_c$

and asymptotic forms  $u(x) \sim e^{-\eta x}$

for large  $x$

$$w(x) \sim \rho e^{-\eta x} \left[ 1 + \frac{3}{\eta x} + \frac{3}{(\eta x)^2} \right]$$

Where the functions  $S, U, \text{etc.}$  may be written to terms of second degree in  $a$ :

$$S(x, a) = a [L(x) + a M(x)]$$

$$U(x, a) = \eta^2 + a [L(x) + a M(x)]$$

$$W(x, a) = a [N(x) + a R(x)]$$

$$Y(x, a) = \eta^2 + \frac{6}{x^2} + a [P(x) + a Q(x)]$$



where

$M$  = mass of nucleon

$\mu$  = mass of pi-meson

$\epsilon$  = binding energy of deuteron ground state (-2.23 Mev)

$$\eta^2 = -\frac{M}{\mu} \epsilon$$

$a_s$  = zero energy singlet effective scattering length

$a = \frac{G^2}{4\pi}$  = the meson coupling constant

$x_c$  = the "cutoff" radius of the interaction

The functions  $L(x)$ ,  $M(x)$ , etc. depend on the particular potential used. The equations have been written in the above form so that one can change the potential by changing the calculations for  $L(x)$ , etc. without having to recode the rest of the problem provided, of course, one keeps just squared terms in  $a$ . In terms of usual notation, the functions are:

$$aL(x) = \left(\frac{M}{\mu}\right) V_{c2}(x)$$

$$a^2 M(x) = \left(\frac{M}{\mu}\right) V_{c4}(x)$$

$$aN(x) = 2^{\frac{3}{2}} \left(\frac{M}{\mu}\right) V_{T2}(x)$$

$$a^2 R(x) = 2^{\frac{3}{2}} \left(\frac{M}{\mu}\right) V_{T4}(x)$$

$$P(x) = L(x) - 2^{-\frac{1}{2}} N(x)$$

$$Q(x) = M(x) - 2^{-\frac{1}{2}} R(x)$$

where  $V_{c2}$  means the  $G^2/4\pi$  term of the central force,  $V_{T4}$  means the  $(G^2/4\pi)^2$  term of the tensor force, etc.

For Lévy's potentials, the functions were:

$$L(x) = -\frac{1}{4} \left(\frac{\mu}{M}\right) \frac{e^{-x}}{x}$$

$$M(x) = -\frac{3}{2\pi} \left(\frac{\mu}{M}\right) \frac{1}{x^2} \left\{ K_1(2x) + \left(\frac{\mu}{M}\right) \frac{1}{\pi} K_1^2(x) \right\}$$

$$N(x) = 2^{3/2} \left\{ 1 + \frac{3}{x} \left(1 + \frac{1}{x}\right) \right\} L(x)$$

$$R(x) = 0$$

For Klein's potentials, they were:

$$L(x) = -\frac{1}{4} \left(\frac{\mu}{M}\right) \frac{e^{-x}}{x}$$

$$M(x) = +\frac{1}{8\pi} \left(\frac{\mu}{M}\right)^3 \frac{1}{x} \left\{ \frac{K_1(2x)}{x} \left(\frac{81}{x^2} + 44\right) + K_0(2x) \left(\frac{81}{x^2} + 12\right) \right\}$$

$$N(x) = +2^{3/2} \left\{ 1 + \frac{3}{x} \left(1 + \frac{1}{x}\right) \right\} L(x)$$

$$R(x) = -2^{3/2} \frac{1}{8\pi} \left(\frac{\mu}{M}\right)^3 \frac{1}{x} \left\{ \frac{K_1(2x)}{x} \left(\frac{15}{x^2} + 4\right) + K_0(2x) \left(\frac{12}{x^2} + 0\right) \right\}$$

where  $K_0(x)$ , and  $K_1(x)$  are Bessel functions of purely imaginary argument (ref. 9)

The formulas for calculating the experimental quantities from the above wavefunctions are as follows:

(A) The singlet effective range

$$l_{r_0} = \frac{2}{\mu} \int_0^{\infty} \left\{ \left(1 + \frac{x}{\mu a_c}\right)^2 - v^2 \right\} dx$$

(B) The triplet effective range

$$3r_0 = \frac{2}{\mu} \int_0^{\infty} \left\{ e^{-2\eta x} - \frac{u^2 + w^2}{1 + \rho^2} \right\} dx$$

(C) The electric quadrupole moment of the deuteron

$$Q = \frac{2^{\frac{1}{2}}}{10\mu^2} \int_0^{\infty} x^2 (uw - 2^{-\frac{3}{2}} w^2) dx / \int_0^{\infty} (u^2 + w^2) dx$$

(D) The effective proportion of D State

$$P_D = \int_0^{\infty} w^2 dx / \int_0^{\infty} (u^2 + w^2) dx$$

### III. Numerical Procedure:

The step-wise integration of the differential equations was done using centered-difference equations. For example, for  $v(x)$ :

$$v_{n+1} = v_n + (v_n - v_{n-1}) + \left( \frac{d^2 v}{dx^2} \right)_n (\Delta x)^2$$

The error in this formula is  $\frac{(\Delta x)^4}{12} v_n''''$ , as can be shown by substituting in Taylor series expansions for  $v_{n+1}$  and  $v_{n-1}$ .

Since the equation for  $v(x)$  is not coupled to those for  $u(x)$  and  $w(x)$  except through the cut-off boundary condition,  $x_c$ , it can be solved separately. Solving it for various values of  $a$ , one gets a function of  $x_c$  vs  $a$  in tabular form. The equations for  $u(x)$  and  $w(x)$ , when solved together, give a second function of  $x_c$  vs  $a$ . The final solution for all three equations is the  $x_c$  and  $a$  corresponding to the intersection of these two functions.

The 701 calculations were done using the Dual Coding System (ref. 10), a floating decimal interpretive routine perfected at Los Alamos. Dual was considered ideal for this problem, since it contains a built-in exponential calculation, and takes care of all questions of scaling. Since the total running time of the problem is very short compared to the time spent in setting it up, the slight increase in machine time required in using Dual is considered insignificant.

The calculation was broken into the following parts:

- (A) Evaluate  $L(x)$ ,  $M(x)$ ,  $N(x)$ ,  $R(x)$ .
- (B) Integrate  $v(x)$ .
- (C) Integrate  $u(x)$  and  $w(x)$ .
- (D) Evaluate the experimental check quantities for  $v(x)$ .
- (E) Evaluate the experimental check quantities for  $u(x)$  and  $w(x)$ .

In (A) a separate code was made for each potential tried. The Bessel functions were somewhat of a problem because, although "reasonable" series expansions for them exist for large values of the argument, they can be computed only with difficulty for values near 1 (ref. 9). This problem was solved in the present case by having tables of the  $K$ 's to be used punched up and stored in the 701. The functions  $M(x)$  and  $R(x)$  for values of  $x \leq 4.0$  were computed using these tables. For  $x$  greater than 4.0, the series expansions were used. Tables of  $L$ ,  $M$ ,  $N$ ,  $R$  vs  $x$  were then punched out in binary form ready to be loaded with the other routines.

In solving the equations for  $u(x)$  and  $w(x)$ , it was necessary to find a value of  $\rho$  in the asymptotic expression for  $w$  such that  $u$  and  $w$  went to zero at the same value of  $x_c$ . This proved to be rather tricky to code, since if the value of  $\rho$  is very far from the correct value, one or the other of the wavefunctions will fail to cross the axis and will start climbing toward positive infinity. Decisions had to be made to advance the value of  $\rho$  in the right direction if one of the functions exceeded a certain large value. After  $u$  and  $w$  both cross the axis, ordinary linear interpolation was used to select the next guess on  $\rho$  to bring them to the same  $x_c$ . Another interesting case arises when there exists no solution for  $x_c$  for the value of  $a$  being tried. In this case there is a region of  $\rho$ 's where both  $u$  and  $w$  start to increase without limit and neither crosses the axis. A special decision was required to take care of this case and prevent the machine from going berserk looking for a solution.

When the final values of  $a$  and  $\rho$  have been found, the code (E) is called in, which evaluates the following integrals using the trapezoidal rule while the equations for  $u$  and  $w$  are being solved numerically:

$$(A) \int_{x_c}^{x_s} u^2 dx$$

$$(B) \int_{x_c}^{x_s} w^2 dx$$

$$(C) \int_{x_c}^{x_s} x^2 \left( uw - 2^{-\frac{1}{2}} w^2 \right) dx$$

From these and their analytic extensions from  $x_s$  to infinity, the experimental quantities listed in II were easily found and printed out.

IV. Results:

To check out the system, Levy's potentials were tried first. Several attempts were made changing the size of  $\Delta x$ , etc. to test the numerical methods. The results checked out within the rather large experimental error quoted in his paper, although the solution was not as clear-cut as one would have hoped.

In trying Klein's potentials, listed in II, it soon became apparent that no solutions existed for any values of  $a$  small enough to be interesting. It was finally necessary to use different values of  $a$  for the first power terms and square terms to get meaningful results. Although there is nothing in the theory which forbids one using different values of the coupling constant for the different terms, it means that the system is really more complicated than has been assumed, and that any attempt to extend the perturbation theory to higher degree terms is probably hopeless.

One example of the type of results obtained is:

taking for  $V_2$ :  $a_2 = 15$ ; for  $V_4$ :  $a_4 = 5$ ;  $x_c = 0.28$

	Calculated Results	Experimental Values (ref 11)
$\mu^1_{r_0}$	$1.5424 \times 10^{-13}$	$1.55 \pm .30 \times 10^{-13}$
$\mu^3_{r_0}$	$1.6821 \times 10^{-13}$	$1.197 \pm .024 \times 10^{-13}$
Q	$2.7647 \times 10^{-27}$	$2.738 \pm .016 \times 10^{-27}$
$P_2$	0.06355	.02 to .06

Since one can get reasonably good agreement with experiment for a considerable range of parameters  $a_2, a_4, x_c$  or for that matter, a considerable variety of potentials, Any thought of working backwards from the experimental data to the theory is futile, unless one is interested in a purely phenomenological approach.

References:

- (1) H. A. Bethe, Physics Today, 7 no. 2, p. 5 (1954)
- (2) H. A. Bethe, Lectures on High Energy Phenomena, Cornell University Physics Dept. (1953)
- (3) I. Tamm, J. Phys. U.S.S.R. 9, 449 (1945)
- (4) S. M. Dancoff, Phys. Rev. 78, 382 (1950)
- (5) M. Lévy, Phys. Rev. 88, 72 and 725, (1952)
- (6) E. Salpeter and H. A. Bethe, Phys. Rev. 84, 1232, (1951)
- (7) A. Klein, Phys. Rev. 90, 1101 (1953)
- (8) H. G. Kolsky and A. Klein, Phys. Rev. 91, 459 (1953)
- (9) G. N. Watson, "A Treatise on the Theory of Bessel Functions", p. 77, p. 202 (1952 Ed.)
- (10) S. Schlesinger, "Dual Coding System," Los Alamos Scientific Laboratory Report LA-1573 (1953)
- (11) G. Snow, Phys. Rev. 87, 21 (1952)

The Monte Carlo Method Applied to a  
Problem in  $\gamma$ -ray Diffusion

by Bengt Carlson

The Monte Carlo Method is, by now, a well known and widely used statistical technique for obtaining numerical solutions to problems in applied mathematics. It has, for instance, been applied successfully to the evaluation of multiple integrals, the inversion of matrices, and to the solution of differential and integral equations of various types. The method was originally proposed by Dr. S. Ulam of Los Alamos and formalized by Dr. J. von Neumann, early in 1948, as a method for solving neutron diffusion problems. The mathematical description of such problems usually takes the form of an integro-differential equation. Rather than solve this equation directly, a very difficult task even in simple cases, one would, using the Monte Carlo technique, "follow" a large number of neutrons, collect the appropriate statistics, and, from the latter, extract the solution. It was evident from the beginning that the success of the method would be closely connected with the development of high speed computing machines. Nevertheless, much work was done by hand, using graphical aids and desk calculators, and, as calculators of moderate speed, such as the 604 and CPC, became available, work of this kind increased considerably.

In this paper, an example of a Monte Carlo calculation performed on the 701 will be presented to illustrate the principal techniques involved; those relating to the method, as well as those connected with the best utilization of the 701. A problem in the diffusion of  $\gamma$ -rays in an electron gas will be considered for this purpose. The details of this problem will be discussed later.

It might be mentioned now that the 701 performed this particular problem elegantly and with impressive speed, at least 1000 times faster than possible on the punch-card equipment mentioned, without, however, entirely suppressing one's desire for still faster and more versatile calculators; for we are handling and anticipating far more elaborate applications of the Monte Carlo Method.

The Geometry of Diffusion.

A particle may be specified as to location and direction by six variables, the position by the coordinates  $x$ ,  $y$ , and  $z$ , and the direction by the cosines  $\alpha$ ,  $\beta$ , and  $\gamma$ , where  $\alpha^2 + \beta^2 + \gamma^2 = 1$ . In addition, a number of parameters may be attached to the particle, such as its energy or velocity. A particle may be followed from one collision through the next by prescribing three numbers, the prescription depending on the physical laws involved. One quantity  $l$ , the path length to the next collision, is required to find the new location,  $x' = x + \alpha l$ ,  $y' = y + \beta l$ , and  $z' = z + \gamma l$ . Two quantities,  $\nu \equiv \cos \theta$  and  $\delta \equiv \cos \phi$ , defining the direction of scattering, are required to find the new direction. Here  $\theta$  and  $\phi$  represent the deflection and azimuthal angles, respectively. Using the tools of spherical trigonometry, the following formulae were derived

for the new direction cosines:  $\alpha' = \alpha \nu + \delta \sqrt{1 - \alpha^2} \sqrt{1 - \nu^2}$ ,  
 $\beta' = \left[ \beta(\nu - \alpha \alpha') \pm \gamma \sqrt{1 - \alpha^2} \sqrt{1 - \nu^2} \sqrt{1 - \delta^2} \right] + (1 - \alpha^2)$ , and  
 $\gamma' = \left[ \gamma(\nu - \alpha \alpha') \pm \beta \sqrt{1 - \alpha^2} \sqrt{1 - \nu^2} \sqrt{1 - \delta^2} \right] + (1 - \alpha^2)$ ,  
 where the upper signs are chosen if  $0 \leq \phi < \pi$ , the lower if  $\pi \leq \phi < 2\pi$ .

If spherical, rather than Cartesian coordinates are used, it is convenient to take  $r^2$  and  $r\mu$  as position and direction variables where  $\mu$  is the direction cosine of the particle with respect to the radius  $r$ . In this case, the new location is given by  $r'^2 = r^2 + l^2 + 2lr\mu$ . The direction  $r\mu$  is replaced by  $\bar{r}\mu$  in the process where  $\bar{r}\mu = l + r\mu$ .



The direction after scattering is given by  $(r\mu)' = \nu r\mu + \delta \sqrt{1 - \nu^2} \sqrt{r^2 - r\mu^2}$ .

### Random Numbers and Random Variables.

The quantities  $\ell$ ,  $\nu$ , and  $\delta$  are determined with the aid of random numbers biased according to the physical laws involved in the problem. There are several methods for generating random numbers, or, at least, reasonable facsimiles thereof. Two methods are in general use. In the middle square method, a 34-bit (referring to the 701) fraction  $\lambda_1$  is selected and squared and the middle 34 bits ( $\lambda_2$ ) of the 68-bit product regarded as a random number. A chain of such numbers  $\{\lambda_i\}$  is then generated by subjecting  $\lambda_2$  and its successors to the same procedure. In the method described in a letter by Dr. D. P. Wall to Dr. C. C. Hurd, as a variation of the method due to Dr. D. H. Lehmer, a fixed 35-bit fraction  $\lambda_0$  ending with the bits 101 is multiplied by a selected 35-bit fraction  $\lambda_1$  ending with 1, and the last 35 bits ( $\lambda_2$ ) of the product are regarded as a random number. By repeating this procedure, multiplying  $\lambda_2$  and its successors by  $\lambda_0$ , a chain of 33-bit random numbers  $\{\lambda_i\}$  is generated. The last two bits are always the same and must, therefore, be excluded. The number  $\lambda_1$  must, in each case, be selected so that a chain of sufficient length for the problem is generated.

A variable  $x$ ,  $a \leq x \leq b$ , is said to be a random variable, if to  $x$  corresponds a probability density function  $p(x)$ ,  $a \leq x \leq b$ , such that  $\int_a^b p(x)dx = 1$ . For random numbers generated by one of the methods mentioned above,  $0 \leq \lambda < 1$  and  $p(\lambda) \equiv 1$ . If we now want to relate two random variables  $\lambda$  and  $x$ , and if  $p(\lambda) \equiv 1$  and  $p(x)$  subject to the integral condition stated, we have  $d\lambda = p(x)dx$ , hence, by integration,  $\lambda = \int_a^x p(x)dx \equiv P(x)$ , and, by inversion,  $x = P^{-1}(\lambda)$ . To amplify this, note that the probability of  $\lambda$  being in the range  $(\lambda_1, \lambda_2)$  equals  $\lambda_2 - \lambda_1 = P(x_2) - P(x_1)$ , which is precisely the probability that  $x$  be

in the range  $(x_1, x_2)$ . The above concepts may be generalized to the case where the variable is discrete and to the case where  $p$  is a function of several variables. Now, it may not always be possible to perform the integration and inversion analytically, but numerical procedures exist which accomplish the same thing, precisely or approximately. We shall return to this subject later.

The probability of a particle penetrating a distance  $\ell$  is, in many cases, given in the form  $p(\ell) = \sigma e^{-\sigma\ell}$  where  $\sigma$  is a function of the particle parameters, as well as the scattering material. In this case, the integration of  $p(\ell)$  gives  $\lambda = P(\ell) = 1 - e^{-\sigma\ell}$  and the inversion  $\ell = -\frac{\ln(1-\lambda)}{\sigma}$ . The values of the azimuthal scattering angle are, in many applications, equally likely, hence  $p(\varphi) = \frac{1}{2\pi}$ ,  $P(\varphi) = \frac{\varphi}{2\pi}$ , and  $\varphi = 2\pi\lambda$ .

#### Scattering and Absorption of $\gamma$ -rays.

The Monte Carlo Method was applied to the problem of finding the emerging energy spectrum at the surface of a sphere of radius  $a$  due to a central source of  $\gamma$ -rays of initial energy  $e_0$ , with  $e_0$  in the MEV range. The scattering material was, in this particular case, taken to be tungsten ( $Z = 74$ ). The processes of scattering and absorption of  $\gamma$ -rays by the electrons were considered, the former defined by the Klein-Nishina formula, and the latter by a  $1/e^{7/2}$  law. This is not a realistic treatment of the problem, since, in this  $e$  and  $Z$  range, (1) the absorption law given is only an approximate one, and (2) processes other than those mentioned are not unimportant. Nevertheless, this furnishes us with a good test problem.

The quantities  $\ell$  and  $\delta$  are determined, using random numbers, as described previously with  $\sigma = \sigma_a + \sigma_s$ , where:

$$(1) \begin{cases} \sigma_a = C_a/e^{7/2}, \\ \sigma_s = C_s \frac{3}{8e^2} \left[ \frac{2e^2(1+e)}{(1+2e)^2} + 4 + (e - 2 - \frac{2}{e}) \ln(1+2e) \right], \end{cases}$$

with  $C_a = 1.475$  and  $C_s = 3.063$  for tungsten. Note that both  $\sigma_a$  and  $\sigma_s$  are functions of the  $\gamma$ -ray energy  $e$ ,  $e \leq e_0$ . The direction  $\nu$  is calculated from  $\nu = 1 - \frac{1}{e'} + \frac{1}{e}$  where  $e$  and  $e'$  are the energies before and after scattering. The probability function associated with  $e'$ ,  $\frac{e}{1+2e} \leq e' \leq e$ , is given by:

$$(2) p(e',e) = \frac{\frac{e}{e'} + \frac{e'}{e} + (1 - \frac{1}{e'} + \frac{1}{e})^2 - 1}{\frac{2e^2(1+e)}{(1+2e)^2} + 4 + (e - 2 - \frac{2}{e}) \ln(1+2e)}$$

These formulae may be found in a text by W. Heitler<sup>1)</sup> or in the collection of tables prepared under the direction of H. Mayer<sup>2)</sup>.

Calculation of Functions by Approximate Methods.

Although  $p(e',e)$  can be integrated analytically, it cannot be so inverted. The inverted function is, however, tabulated in the report by H. Mayer. We, therefore, chose to approximate it by a rational expression which could be easily calculated. We obtained:

$$(3) e' = P^{-1}(\lambda, e) = \frac{e}{1 + s\lambda + (2e - s)\lambda^3},$$

where  $s = e/(1 + .5625e)$ , a formula which was regarded as sufficiently accurate for the problem at hand. It could, however, easily be improved upon, if necessary.

This brings up an important point. Since the final result will be statistical in nature, and hence appear with a probable error which one

1) W. Heitler: The Quantum Theory of Radiation.

2) H. Mayer: Tables of the Compton Effect Cross Sections and Energies, LAMS-1199.

cannot normally afford to bring below, say, the 1% level, it is not worthwhile calculating functions appearing as part of calculation to excessive accuracy. The table look-up method is, therefore, used extensively. The functions  $-\ell nx$ ,  $\cos \pi x$ , and  $\sqrt{1-x^2}$  were, for instance, stored in the 701, each function occupying 128 half-word locations,  $x$  being  $n/256$  in each case,  $n = 1, 3, 5, \dots, 255$ . Similarly,  $\sigma_a + \sigma_s$  and  $\sigma_a / (\sigma_a + \sigma_s)$  were calculated just once and stored for the arguments  $e = e_0 - \frac{n}{32}$ ,  $n = 1, 2, \dots$  down to  $e = \frac{11}{32}$ , the cut-off energy for which the probability of absorption is about 97%. A square root routine was required, and this was based on the approximate formula:

$$(4) \quad \sqrt{x} = \frac{.16022 + x}{.70719 + .45303x}, \quad \frac{1}{4} \leq x \leq 1,$$

which is good, on the average, to 1/4%. Similar formulae have been derived for  $-\ell nx$  and  $\sin \frac{\pi}{2} x$  with average absolute errors of .001 and .0002, respectively:

$$(5) \quad \begin{cases} -\ell nx = \frac{1-x}{.42135 + .6x}, & \frac{1}{2} \leq x \leq 1, \\ \sin \frac{\pi}{2} x = x \frac{11 - 3x^2}{7 + x^2}, & 0 \leq x \leq 1. \end{cases}$$

These formulae were, however, not used in the problem we are now discussing.

Details and Results of the Calculation.

We have now outlined the techniques and stated or derived the formulae required for the problem. The logical structure of the calculation is best visualized with the aid of a flow diagram. Some of the notations have to be explained.  $N$  represents the number of initial particles,  $n$  the number,  $i$  the accumulated track count, and  $\sum \ell$  the accumulated path length of escaping particles. The random numbers  $\lambda_1, \lambda_2, \lambda_3$ , and  $\lambda_4$  were generated all in one cycle by the second method mentioned,  $\lambda_1$  being the first 7 bits,

$\lambda_2$  the next 10,  $\lambda_3$  the next 8, and  $\lambda_4$  the last 8; a total of 33 bits. Two print-outs were provided for, one printing the data obtained for an individual escape, the other printing a table of results.

The problem was coded in fixed-point with the use of regional programming and the absolute code obtained by using Assembly Program 607. These topics are discussed in a paper by D. Sweeney. The de-bugging was done using Tracing Program 795, discussed by E. Voorhees in another paper. About 300 orders were required for the problem, plus 300 full-words of storage for constants and tables, plus a print program.

The results of a calculation with  $e_0 = 3 \text{ mc}^2$  units (1.533 MEV) and  $a = 5.13 \text{ cm}$  (4.17 mean free path units) are given in the table and represent about 20 minutes of 701 time. Thus, an average of 2100 particles was processed per minute, or about a track length every 10 milliseconds, a very satisfactory speed.

Certain integral results are of primary interest, such as the percentages of escaping particles ( $T_n$ ) and of escaping energy ( $T_e$ ). The correct value of  $T_0$ , the number of particles escaping without collisions, is given in parentheses at the bottom of the table. This number is, of course, equal to  $e^{-\sigma(e_0)a}$  and gives us a gross check on the randomness of the "random numbers" generated.

A quantity which is of particular interest is the average path length which a particle traverses in the sphere before escaping. Note that this average is only 2% longer than the radius. A calculation similar to the one reported on here was performed to verify that the calibration procedure associated with a certain experiment was satisfactory. This procedure assumed, in fact, that the path length correction was small.

In conclusion, I would like to mention that a solution of this problem can also be obtained by solving a certain integro-differential

equation numerically, but this would be a far more time-consuming approach than the one discussed here. Neither approach is very practical without a calculator comparable in speed to the 701. Nevertheless, much work has been done in this field using semi-analytical methods developed for the purpose, in conjunction with less powerful computing equipment.

Table

Leakage Spectrum for a Central Source of  $\gamma$ -rays in a Sphere.

$e_0 = 3.0 \text{ mc}^2 (1.533 \text{ MEV}), a = 5.13 \text{ cm.}$

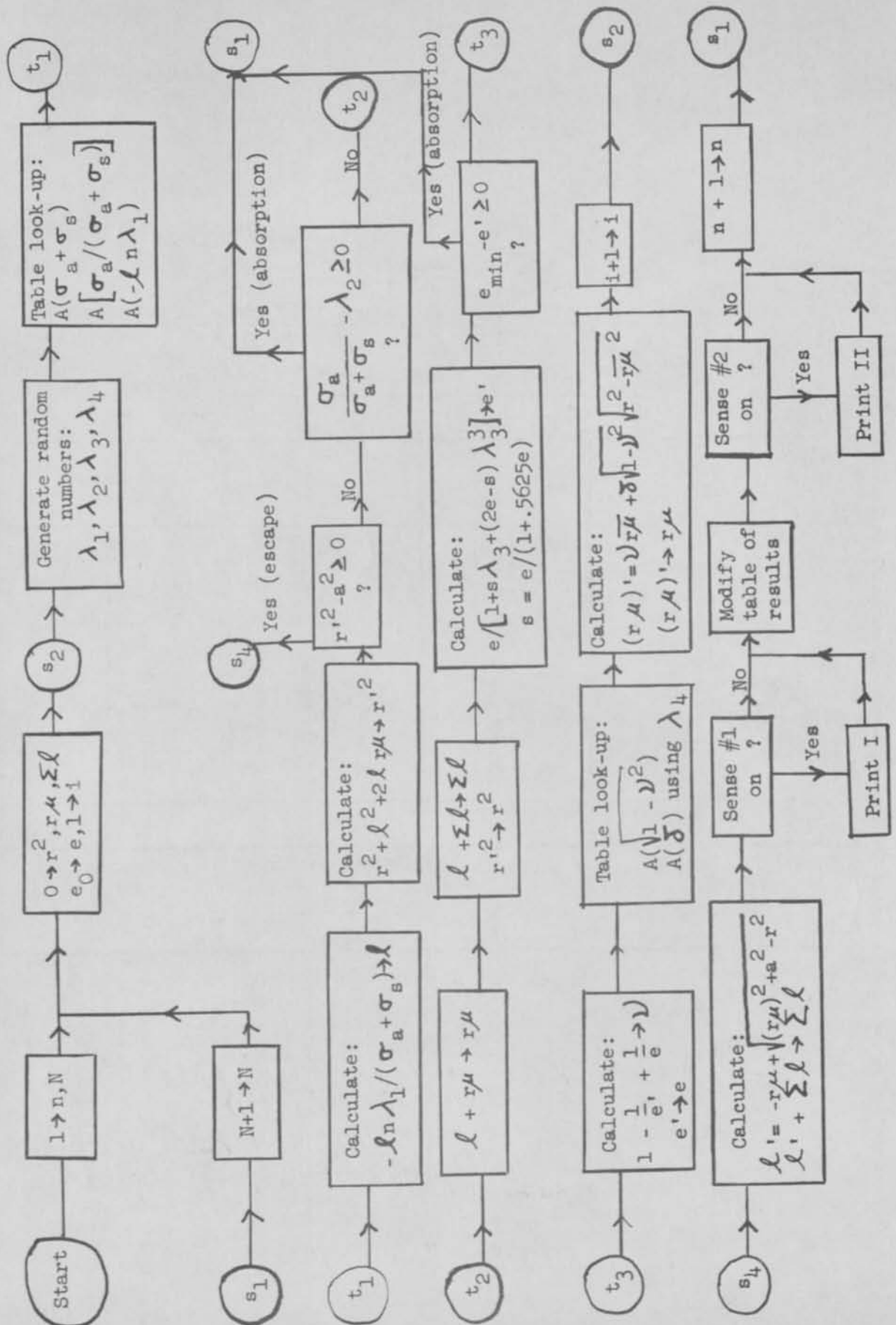
$e(\text{mc}^2)$	No. of escapes	Av. no. of collisions	Av. length of path
3.00	649	0	5.13
2.75-3.00	185	1.06	5.14
2.50-2.75	245	1.25	5.17
2.25-2.50	231	1.41	5.21
2.00-2.25	220	1.67	5.26
1.75-2.00	209	2.02	5.31
1.50-1.75	198	2.04	5.38
1.25-1.50	157	2.35	5.40
1.00-1.25	100	2.62	5.52
.75-1.00	47	3.13	5.52
.34-.75	11	3.55	5.47
.34-3.00	2252	1.26	5.24
$N = 43176, T_0 = 1.50\% (1.54), \sigma(e_0) = .813$ $T_e = 3.53\%, T_n = 5.22\%$			

Refs: The Statistical Lab. Univ. of Florida  
Gainesville, Fla. (undated)

"Bibliography of the Monte Carlo Method  
& associated material."

Book: H. Kahn: "appl. of Monte Carlo" (being written now)

Flow Diagram.





On the solution of linear equations  
by diagonalizing the coefficients matrix.

By E. G. Kogbetliantz

\* \* \* \* \*

§1. In our report #2 (November 1953) the diagonalization of real symmetric matrices was extended to Hermitian matrices. Using instead of plane rotations through real angles the rotations through complex angles  $z = \theta + i\varphi$ , so that the unitary transformation  $u_n(z_n) = u_n(\theta_n + i\varphi_n)$

$$u_n(z_n) = (\operatorname{Ch} 2\varphi_n)^{-\frac{1}{2}} \begin{pmatrix} \cos z_n & -\sin \bar{z}_n \\ \sin z_n & \cos \bar{z}_n \end{pmatrix}$$

depends on two real parameters  $\theta_n$  and  $\varphi_n$ , we succeeded in forming the modal matrix  $\underline{M}$  of a Hermitian matrix  $H$ . It was obtained in the form of an infinite product

$$(2) \quad \underline{M} = \prod_{n=1}^{\infty} u_n(z_n)$$

so that

$$(3) \quad \bar{M}' H M = L,$$

where  $L$  is a diagonal matrix the elements of which are characteristic roots (eigenvalues) of  $H$ . The diagonalization of  $H$  yields a direct

solution of the system of  $n$  complex equations P. 2

(4)

$$Hx = h$$

thus avoiding its transformation into the equivalent system of  $2n$  real equations with a symmetric coefficient's matrix. The relation (3) gives indeed

$$H = ML\bar{M}'$$

and therefore

$$H^{-1} = M L^{-1} \bar{M}'$$

This simple way of inverting  $H$  yields the solution:

(5)

$$x = H^{-1}h = M L^{-1} \bar{M}' h$$

It seems to me that the program B00 (real symmetric matrix) should be replaced by programming the diagonalization of Hermitian matrices since this new program would include B00 as its special case. If the diagonalization of Hermitian matrices is programmed, its application to the solution of the system (4), that is, to (5) should be included. There are reasons which make me believe that for matrices of large order this new solution method will be not only more economical but also more accurate. In any case the comparison of two different methods is fruitful.

§2. The diagonalization of a hermitian

matrix yields its characteristic roots, but the fact that the diagonal matrix  $L$  in (3) has as its elements the characteristic roots of  $H$  is not at all used in the solution of (4) by the method (5). This observation suggests a further generalization of our solution method to most general equations

$$Gx = g$$

where the non-singular matrix  $G$  is a general complex matrix. Diagonalizing  $G$  facilitates its direct inversion, avoiding the transformation of the system (6) into the equivalent system (4) with  $H = \bar{G}'G^*$  and  $h = \bar{G}'g$ . This transformation in itself is easy, but the subsequent solution of (4) is long since the order of  $G$  (and  $H$ ) is doubled, unless a diagonalization of  $H = \bar{G}'G$  is applied. But, instead of applying the diagonalization to  $H = \bar{G}'G$ , it is possible to diagonalize a general complex matrix (non-singular) directly and this report deals with such a direct procedure applied to  $G$ .

\*)  $H = \bar{G}'G$  is hermitian.

It is a well known fact that, given any non-singular complex square matrix  $G$ , two unitary transformations  $U$  and  $T$  do exist such that

$$(7) \quad \bar{U}' G T \equiv K,$$

where  $K$  is a diagonal matrix the elements of which are not characteristic roots of  $G$ , but are related to those of two matrices  $\bar{G}'G$  and  $G\bar{G}'$ . These two hermitian matrices have the same characteristic roots which are real and positive numbers  $\lambda_j$ . The elements  $k_j$  of the diagonal matrix  $K$  obtained in our method have the squares of their moduli equal to characteristic roots of  $\bar{G}'G$  and  $G\bar{G}'$ :  $|k_j|^2 = \lambda_j$ .

Inverting (7), we have

$$G = U K \bar{T}'$$

and therefore

$$G^{-1} = T K^{-1} \bar{U}'$$

and the direct solution of (6) is obtained:

$$(8) \quad x = G^{-1} g = T K^{-1} \bar{U}' g$$

The two unitary transformations  $\bar{U}'$  and  $T$  are obtained in our method as infinite convergent products of unitary transformations of the type (1):

See H. Weyl *Re. N. A. Sc.*  
PIA 18-411

$$(9) \quad \Gamma = \lim_{N \rightarrow \infty} \prod_{g=1}^N u_g(z_g)$$

$$\overline{\Gamma}' = \lim_{N \rightarrow \infty} u'_g(\zeta_g) \prod_{g=1}^N$$

The symbols  $\prod \rightarrow$  and  $\prod \leftarrow$  denoting the products of right (post-) and left (pre-) factors respectively. The complex angles  $z_g$  and  $\zeta_g$

are:  $z_g = \theta_g + i\varphi_g$ ,  $\zeta_g = \xi_g + i\eta_g$ .

In practice, finite products  $\prod_{g=1}^N$ ,  $\prod_{g=1}^N$  are used with sufficiently large  $N$  to insure a good approximation to the infinite products representing  $\Gamma$  and  $\overline{\Gamma}'$ . The choice of four parameters  $\theta$ ,  $\varphi$ ,  $\xi$  and  $\eta$  at each step is described in the §6.

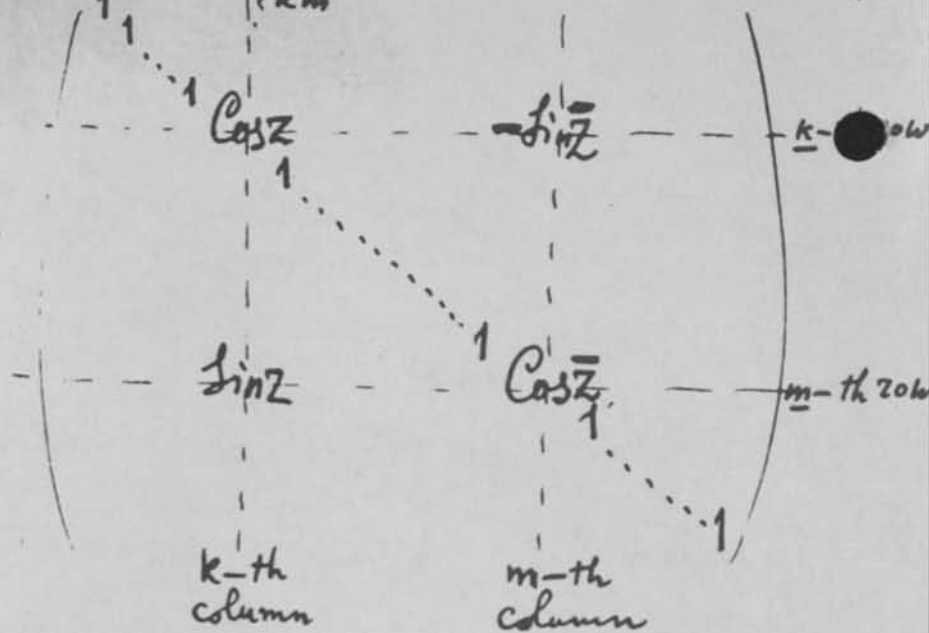
### §3. Description of the transformation $u(z)$ .

Using the Kronecker's symbols  $\delta_{ij}$  which vanish for  $i \neq j$ , while  $\delta_{ii} = 1$ , the identity matrix  $\mathbf{I}$  of  $n$ -th order becomes  $\mathbf{I} = (\delta_{ij})$ ;  $i, j = 1, 2, \dots, n-1, n$ . Its four elements  $\delta_{kk} = 1$ ,  $\delta_{km} = \delta_{mk} = 0$  and  $\delta_{mm} = 1$ , ( $1 \leq k < m \leq n$ ), are at the intersections of the  $k$ -th and  $m$ -th rows and columns. Replacing them in  $\mathbf{I}$  by the four elements of a rotation matrix through a complex angle  $z = \theta + i\varphi$  namely

$$(\text{Ch } 2\varphi)^{-\frac{1}{2}} \begin{pmatrix} \cos z & -\sin \bar{z} \\ \sin z & \cos \bar{z} \end{pmatrix},$$

we build a unitary matrix  $u_{km}(z)$

$$u_{km}(z) = (\text{ch } 2\varphi)^{-\frac{1}{2}}$$



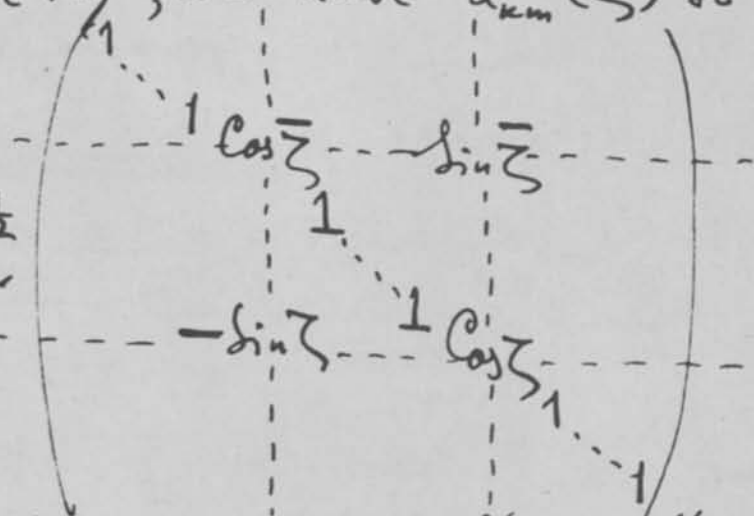
The  $u_{km}(z)$  is unitary because

$$|\cos z|^2 + |\sin z|^2 = \cos z \cdot \cos \bar{z} + \sin z \cdot \sin \bar{z} =$$

$$= \cos(z - \bar{z}) = \cos[\theta + i\varphi - (\theta - i\varphi)] = \cos 2i\varphi = \text{ch } 2\varphi.$$

Replacing  $z$  by  $\zeta = \xi + i\eta$ , we have  $u_{km}(\zeta)$  to the

$$u'_{km}(\bar{\zeta}) = (\text{ch } 2\eta)^{-\frac{1}{2}}$$



Denoting the values of  $k$  and  $m$  at the  $j$ -th step by  $k_j, m_j$ , as well as those of  $z$  and  $\zeta$  by  $z_j$  and  $\zeta_j$  we introduce the abbreviations

$$u_{k_j m_j}(z_j) = u_j(z) ; \quad u'_{k_j m_j}(\bar{\zeta}_j) = \bar{u}'_j(z).$$

Using this notation, the diagonalization of a mat-

$2 \times 2$   $A_0$  is performed by forming the sequence P.7 of matrices

(9)  $A_0, A_1, A_2, \dots, A_{j-1}, A_j, \dots$  defined by the recurrence relation

$$(10) \quad A_j = \bar{u}'_j(z) \cdot A_{j-1} \cdot u_j(z) \quad (A_0 = A)$$

Therefore

$$A_N = \bar{u}'_N(z) \prod_{n=1}^N A \cdot \prod_{n=1}^N u_n(z)$$

and  $\lim_{N \rightarrow \infty} A_N = K$  (11)

where  $K$  — as it will be shown — is a diagonal matrix thanks to choice of  $z_j, \bar{z}_j$ .

§4. To determine the complex angles  $z_j = \theta_j + i\varphi_j$  and  $\bar{z}_j = \bar{\xi}_j + i\eta_j$  we need four conditions. They are obtained as follows. Transforming at the  $j$ -th

step the matrix  $A_{j-1} = ((a_{km}^{(j-1)}))$  into  $A_j = ((a_{km}^{(j)}))$ , we make vanish in  $A_j$  the two elements

$a_{k_j, m_j}^{(j)}$  and  $a_{m_j, k_j}^{(j)}$  ( $k_j < m_j$ ) located at the inter-

section of the  $k_j$ -th row with the  $m_j$ -th column and at that of the  $m_j$ -th row with the  $k_j$ -th

column. This gives us four <sup>real</sup> equations for the four real unknowns  $\theta_j, \varphi_j, \bar{\xi}_j$  and  $\eta_j$ .

The sequence (S) of transformations of A is cyclic,  $\frac{n(n-1)}{2}$  steps forming a cycle since in a cycle we make vanish one after another all  $\frac{n(n-1)}{2}$  pairs of off-diagonal elements symmetrical with respect to the principal diagonal.

A particular fixed pair  $(a_{pp}, a_{pp})$  vanishes only once during a cycle and it reappears again in the next step, but weakened, its modulus being diminished in an average ratio of  $\frac{1}{2}\sqrt{2}$ . At each step the sum of squares of moduli of diagonal elements increases, while the norm of A is invariant. After a certain number of cycles the moduli of all off-diagonal elements become less than a prescribed upper bound. In other words, in our transformation the off diagonal elements tend uniformly to zero when the number of cycles  $p$  increases. In practice, the diagonalization will be considered as approximately achieved after a sufficiently large number of steps  $N = n(n-1) \cdot p/2$ , where the number of cycles  $p$  naturally depends on the desired accuracy.



The speed of convergence in (11) is determined by P. 9 the fact that — as we will see in the § —, in the average, the moduli of off-diagonal elements decrease in the ratio of one to  $e^{-\frac{1}{2}}$  per cycle, so that after  $p$  cycles they are divided by  $e^{\frac{p}{2}}$ .

If at the beginning the original matrix was divided by the modulus of its largest off diagonal element making thus the moduli of all its off-diagonal elements less than one, then after 46 cycles the average modulus of all off-diagonal elements will be less

$$e^{-23} \approx 10^{-10} \approx 2^{-33}$$

§5. We now describe how the matrix  $A$  is transformed during one step. We suppose that  $A = ((a_{rs}))$  becomes  $C = ((c_{rs}))$ , where

$$C \equiv u'_{km}(\bar{z}). A \cdot u_{km}(z) = u'_{km}(\bar{z}). B$$

with

$$B = ((b_{rs})) = A \cdot u_{km}(z).$$

Substituting  $Ch_{2p}$  and  $Ch_{2q}$  by  $y$  and  $y^*$  respectively, we have for the elements of  $B$ :

$$b_{rs} = a_{rs} \quad \text{for } r, s \neq k, m$$

as well as

$b_{kj} = a_{kj}$  and  $b_{mj} = a_{mj}$ , if  $j \neq k, m$ .  
 But, for  $j \neq k, m$ , we obtain also

$$\sqrt{y} \cdot b_{jk} = a_{jk} \cos Z + a_{jm} \sin Z ; \sqrt{y} \cdot b_{jm} = -a_{jk} \sin \bar{Z} + a_{jm} \cos \bar{Z}$$

$$\sqrt{y} \cdot b_{kk} = a_{kk} \cos Z + a_{km} \sin Z ; \sqrt{y} \cdot b_{km} = -a_{kk} \sin \bar{Z} + a_{km} \cos \bar{Z}$$

$$\sqrt{y} \cdot b_{mm} = a_{mm} \cos Z + a_{mm} \sin Z ; \sqrt{y} \cdot b_{mm} = -a_{mm} \sin \bar{Z} + a_{mm} \cos \bar{Z}$$

The elements of  $C$  are therefore as follows:

$$c_{ij} = b_{ij} = a_{ij} \text{ for } i, j \neq k, m$$

$$(12) \left\{ \begin{array}{l} \sqrt{y}^* \cdot c_{kj} = a_{kj} \cos \bar{Z} + a_{mj} \sin \bar{Z} ; \sqrt{y}^* \cdot c_{mj} = -a_{kj} \sin \bar{Z} + a_{mj} \cos \bar{Z} \end{array} \right.$$

$$\left\{ \begin{array}{l} \sqrt{y} \cdot c_{jk} = a_{jk} \cos Z + a_{jm} \sin Z ; \sqrt{y} \cdot c_{jm} = -a_{jk} \sin \bar{Z} + a_{jm} \cos \bar{Z} \end{array} \right.$$

as well as

$$\left. \begin{array}{l} c_{kk} \cdot \sqrt{y} y^* = a_{kk} \cos Z \cos \bar{Z} + a_{km} \sin Z \cos \bar{Z} + a_{mk} \cos Z \sin \bar{Z} + a_{mm} \sin Z \cdot \sin \bar{Z} \\ c_{km} \cdot \sqrt{y} y^* = -a_{kk} \sin \bar{Z} \cos \bar{Z} + a_{km} \cos \bar{Z} \cos \bar{Z} - a_{mk} \sin \bar{Z} \cdot \sin \bar{Z} + a_{mm} \cos \bar{Z} \cdot \sin \bar{Z} \\ c_{mk} \cdot \sqrt{y} y^* = -a_{kk} \cos Z \sin \bar{Z} - a_{km} \sin Z \sin \bar{Z} + a_{mk} \cos Z \cos \bar{Z} + a_{mm} \sin Z \cos \bar{Z} \\ c_{mm} \cdot \sqrt{y} y^* = a_{kk} \sin \bar{Z} \cdot \sin \bar{Z} - a_{km} \cos \bar{Z} \sin \bar{Z} - a_{mk} \sin \bar{Z} \cos \bar{Z} + a_{mm} \cos \bar{Z} \cos \bar{Z} \end{array} \right\} (13)$$

The formulae (13) yield the important relation

$$(14) \quad |c_{kk}|^2 + |c_{km}|^2 + |c_{mk}|^2 + |c_{mm}|^2 = |a_{kk}|^2 + |a_{km}|^2 + |a_{mk}|^2 + |a_{mm}|^2$$

as well as the second invariant:

P. 11

$$(15) \quad c_{kk}c_{mm} - c_{km}c_{mk} = a_{kk}a_{mm} - a_{km}a_{mk}.$$

The formulae (12) entail moreover the equalities

$$(16) \quad \begin{cases} |c_{jk}|^2 + |c_{jm}|^2 = |a_{jk}|^2 + |a_{jm}|^2 \\ |c_{kj}|^2 + |c_{mj}|^2 = |a_{kj}|^2 + |a_{mj}|^2 \end{cases} \quad (j \neq k, m)$$

which, together with (14), prove that the norm of  $A$  is <sup>an</sup> invariant of our transformation:

$$\|C\| = \sum_{ij} |c_{ij}|^2 = \sum_{ij} |a_{ij}|^2 = \|A\|.$$

§6. Choice of  $z = \theta + ip$  and  $\zeta = \xi + i\eta$ .

Denoting  $\tan z$ ,  $\tan \zeta$  and the conjugate of  $a_{kk}$  by  $t = \tan z$ ,  $\tau = \tan \zeta$  and  $\bar{a}_{kk}$ , we can write the conditions  $c_{km} = 0$ ,  $c_{mk} = 0$  which determine the choice of  $z$  and  $\zeta$  as follows:

$$(17) \quad \begin{cases} a_{km} \tau t - a_{mm} t + a_{kk} \tau - a_{mk} = 0 \\ \bar{a}_{mk} \tau t + \bar{a}_{kk} t - \bar{a}_{mm} \tau - \bar{a}_{km} = 0 \end{cases}$$

Eliminating first  $\tau$  and then  $t$ , two quadratic equations are obtained:

$$(18) (D + A - iE)z^2 + Fz - (D + A + iE) = 0$$

$$(19) (D - A - iE^*)z^2 + F^*z - (D - A + iE^*) = 0$$

where the real constants  $A, D, E, E^*, F, F^*$  are deduced from four complex numbers  $a_{kk}, a_{km}, a_{mk}$  and  $a_{mm}$  as follows. First put

$$\frac{1}{2}(a_{kk} + a_{mm}) = M + i.m$$

$$\frac{1}{2}(a_{kk} - a_{mm}) = P + i.p$$

$$\frac{1}{2}(a_{km} + a_{mk}) = N + i.n$$

$$\frac{1}{2}(a_{km} - a_{mk}) = Q + i.q$$

then form

$$(20) \begin{cases} A = PQ + pq; & B = PM + pm; & C = NQ + nq; \\ D = MN + mn; & F = 2(B - C); & F^* = 2(B + C) \\ E = mQ - qM + (pN - nP) \\ E^* = mQ - qM - (pN - nP) \end{cases}$$

In other words  $D, A, E, E^*$  are such that

$$D + A + iE = \frac{1}{2}[a_{kk}\bar{a}_{km} + a_{mk}\bar{a}_{mm}]$$

$$D - A + iE^* = \frac{1}{2}[\bar{a}_{kk}a_{mk} + \bar{a}_{km}a_{mm}]$$

Solving (18) for  $t$  we can choose the P. 13

• sign before the square root of the discriminant.

Thus, choosing the sign plus:

$$t = \frac{D+A+iE}{R+B-C} \quad (21)$$

with

$$R^2 = (D+A)^2 + (B-C)^2 + E^2,$$

the abbreviation  $R$  denoting the positive square root  $+\sqrt{R^2}$ . We can now compute

$$\tan 2\theta = \tan(z+\bar{z}) = (t+\bar{t})(1-t\bar{t})^{-1} = \frac{D+A}{B-C}$$

$$\operatorname{Ch} 2\varphi = -i \tan(z-\bar{z}) = -i(t-\bar{t})(1+t\bar{t})^{-1} = \frac{E}{R}$$

• Denoting  $(D+A)^2 + (B-C)^2$  by  $\omega^2$  and  $\cos 2\theta$  by  $\delta$ , so that

$\cos 2\theta = \delta$ ,  $(D+A)^2 + (B-C)^2 = \omega^2$ ,  $R^2 = \omega^2 + E^2$ , we have in the general case when  $\omega \neq 0$ :

$$(22) \left\{ \begin{array}{l} \sin 2\theta = \frac{D+A}{\omega} \quad ; \quad \delta = \cos 2\theta = \frac{B-C}{\omega} \\ \operatorname{Sh} 2\varphi = \frac{E}{\omega} \quad ; \quad \gamma = \operatorname{Ch} 2\varphi = \frac{R}{\omega} > 1. \end{array} \right.$$

Designating the sign of a real number  $r$

• by  $\sigma(r)$ , so that  $|r| = r \cdot \sigma(r)$ , we observe that

$$\sigma(\sin 2\theta) = \sigma(D+A) \quad ; \quad \sigma(\varphi) = \sigma(E) \quad \text{and} \quad \sigma(\delta) = \sigma(B-C)$$

The numerical value  $\delta$  of  $\cos 2\theta$  does not

determine  $\theta$  and we add now the condition  $|\theta| < \frac{\pi}{2}$ . Thus, choosing as  $\theta$  an acute angle we have  $\sigma(\cos \theta) = 1$  and therefore  $\sigma(\sin \theta) = \sigma(\sin 2\theta) = \sigma(D+A)$ . Finally,  $z = \theta + i\varphi$  is determined completely since we know the four real numbers  $s_1, c_1, s_2, c_2$  defined by:

$$(23) \begin{cases} s_1 = \sin \theta = \sigma(D+A) \cdot \left(\frac{1-\delta}{2}\right)^{\frac{1}{2}} & c_1 = \cos \theta = \left(\frac{1+\delta}{2}\right)^{\frac{1}{2}} > 0 \\ s_2 = \frac{\text{Sh } \varphi}{\sqrt{\gamma}} = \sigma(E) \cdot \left(\frac{\gamma-1}{2\gamma}\right)^{\frac{1}{2}} & c_2 = \frac{\text{Ch } \varphi}{\sqrt{\gamma}} = \left(\frac{\gamma+1}{2\gamma}\right)^{\frac{1}{2}} > 0 \end{cases}$$

All four are less than one in absolute value,  $c_1$  and  $c_2$  being always positive.

Replacing  $E, F, \omega, R$  respectively by  $E^*, F^*, \omega^*, R^*$  where  $\omega^{*2} = (D-A)^2 + (B+C)^2$  and  $R^{*2} = \omega^{*2} + E^{*2}$ , we have

$$\begin{aligned} R^{*2} &= (D-A)^2 + (B+C)^2 + E^{*2} = \\ &= (D+A)^2 + (B-C)^2 + E^2 = R^2 \end{aligned}$$

So that  $R^* = R$ . Solving (19) we have

$$\tau = \frac{D-A + iE^*}{\epsilon R + B+C} \quad (24)$$

where  $\underline{\epsilon}$  is the symbol of double sign  $\pm 1$  P.15

● The value of  $\underline{\epsilon}$  cannot be chosen arbitrarily. The relations (17) entail

$$\tau(a_{km}\bar{a}_{mx} + \bar{a}_{mx}a_{km}) + |a_{km}|^2 = \tau(\bar{a}_{km}a_{mx} + a_{mx}\bar{a}_{km}) + |a_{mx}|^2.$$

Observing that  $|a_{km}|^2 - |a_{mx}|^2 = 4(NQ + ng) = 4C$ , this can be written as follows

$$\tau.[D - A - iE^*] + C = \tau.(D + A - iE) - C$$

that is, using (21) and (24):

$$\epsilon.R - B = R - B$$

and we see that  $\epsilon = +1$ . Thus,

$$(25) \quad \tau = \frac{D - A + iE^*}{R + B + C}$$

and

$$\tan 2\zeta = \frac{\tau + \bar{\tau}}{1 - \tau\bar{\tau}} = \frac{D - A}{B + C}$$

$$\therefore \text{th } 2\eta = -i(\tau - \bar{\tau})(1 + \tau\bar{\tau})^{-1} = \frac{E^*}{R}.$$

Denoting  $\cos 2\zeta$  by  $\delta^*$ , we obtain

$$(26) \quad \left\{ \begin{array}{l} \sin 2\zeta = \frac{D - A}{\omega^*} \quad ; \quad \delta^* = \cos 2\zeta = \frac{B + C}{\omega^*} \end{array} \right.$$

$$(\omega \neq 0) \quad \left\{ \begin{array}{l} \text{sh } 2\eta = \frac{E^*}{\omega^*} \quad ; \quad \gamma^* = \text{ch } 2\eta = \frac{R}{\omega^*} > 1, \end{array} \right.$$

provided  $\omega^* \neq 0$ .

Choosing again as  $\xi$  an acute angle so that  $|\xi| < \frac{\pi}{2}$  and  $\cos \xi > 0$ , we finally know  $\zeta = \xi + i\eta$  completely.  $\zeta$  is characterized by four real numbers  $s_1^*$ ,  $c_1^*$ ,  $s_2^*$  and  $c_2^*$ . All four are less than one in absolute value,  $c_1^*$  and  $c_2^*$  being positive.

They are given by following expressions:

$$(27) \begin{cases} s_1^* = \sin \xi = \delta(U-A) \left( \frac{1-\delta^*}{2} \right)^{\frac{1}{2}} \\ c_1^* = \cos \xi = \left( \frac{1+\delta^*}{2} \right)^{\frac{1}{2}} > 0 \\ s_2^* = \frac{\text{Sh } \eta}{\sqrt{\gamma^*}} = \delta(E^*) \left( \frac{\gamma^* - 1}{2\gamma^*} \right)^{\frac{1}{2}} \\ c_2^* = \frac{\text{Ch } \eta}{\sqrt{\gamma^*}} = \left( \frac{\gamma^* + 1}{2\gamma^*} \right)^{\frac{1}{2}} > 0 \end{cases}$$

where  $\gamma^* = \text{Ch } 2\eta$ .



### §7. Computation of $c_{km}, c_{mn}$ .

The notations used in this paragraph are as follows. The real part and the coefficient of  $i = \sqrt{-1}$  of a complex number  $\alpha + i\beta = r$  are designated by

$$\alpha = \Re(r) \quad \text{and} \quad \beta = \Im(r),$$

$$\theta + \xi = s, \quad \theta - \xi = d, \quad \varphi + \gamma = \sigma, \quad \varphi - \gamma = \delta,$$

$$M \cos d + Q \sin d = T$$

$$Q \cos d - M \sin d = U$$

$$N \sin s + P \cos s = V$$

$$-P \sin s + N \cos s = W$$

$$m \cos d + q \sin d = t$$

$$q \cos d - m \sin d = u$$

$$n \sin s + p \cos s = v$$

$$-p \sin s + n \cos s = w$$

Using these notations, we rewrite the equations  $c_{km} = 0$ ,  $c_{mk} = 0$  and the expressions of  $c_{kk}$  and  $c_{mm}$  as follows:

$$\frac{1}{2} \sqrt{\gamma\delta} * \Re(c_{km} - c_{mk}) = U \cdot \text{Ch} \delta - v \cdot \text{Sh} \delta = 0$$

$$\frac{1}{2} \sqrt{\gamma\delta} * \Im(c_{km} - c_{mk}) = V \cdot \text{Sh} \delta + u \cdot \text{Ch} \delta = 0$$

$$\frac{1}{2} \sqrt{\gamma\delta} * \Re(c_{km} + c_{mk}) = W \cdot \text{Ch} \delta - t \cdot \text{Sh} \delta = 0$$

$$\frac{1}{2} \sqrt{\gamma\delta} * \Im(c_{km} + c_{mk}) = T \cdot \text{Sh} \delta + w \cdot \text{Ch} \delta = 0$$

Therefore

$$\frac{U}{V} = \frac{-u}{V} = \frac{Sh\sigma}{Ch\delta} ; \quad \frac{W}{t} = \frac{-w}{T} = \frac{Sh\delta}{Ch\sigma}$$

On the other hand, since  $Ch^2\delta + Sh^2\sigma = \gamma\gamma^*$ :

$$(28) \begin{cases} \frac{1}{2}\sqrt{\gamma\gamma^*} R(c_{kk} + c_{mm}) = TCh\sigma - wSh\delta = \gamma\gamma^* \cdot T/Ch\sigma \\ \frac{1}{2}\sqrt{\gamma\gamma^*} Im(c_{kk} + c_{mm}) = WSh\delta + tCh\sigma = \gamma\gamma^* \cdot t/Ch\sigma \\ \frac{1}{2}\sqrt{\gamma\gamma^*} R(c_{kk} - c_{mm}) = V \cdot Ch\delta - u \cdot Sh\sigma = \gamma\gamma^* \cdot V/Ch\delta \\ \frac{1}{2}\sqrt{\gamma\gamma^*} Im(c_{kk} - c_{mm}) = USh\sigma + v \cdot Ch\delta = \gamma\gamma^* \cdot v/Ch\delta \end{cases}$$

Thus, letting  $c_{kk} = c'_{kk} + ic''_{kk}$ ,  $c_{mm} = c'_{mm} + ic''_{mm}$ :

$$(28 bis) \begin{cases} R(c_{kk}) = c'_{kk} = 2\sqrt{\gamma\gamma^*} \cdot [VCh\sigma + TCh\delta] / (\gamma + \gamma^*) \\ Im(c_{kk}) = c''_{kk} = 2\sqrt{\gamma\gamma^*} \cdot [v \cdot Ch\sigma + t \cdot Ch\delta] / (\gamma + \gamma^*) \\ R(c_{mm}) = c'_{mm} = 2\sqrt{\gamma\gamma^*} \cdot [TCh\delta - VCh\sigma] / (\gamma + \gamma^*) \\ Im(c_{mm}) = c''_{mm} = 2\sqrt{\gamma\gamma^*} \cdot [t \cdot Ch\delta - v \cdot Ch\sigma] / (\gamma + \gamma^*) \end{cases}$$

These formulae are reduced ~~now~~ now as follows. In §7 we defined the eight numbers  $s_1, c_1, s_2, c_2, s_1^*, c_1^*, s_2^*, c_2^*$ . Expressing  $Ch\delta, Ch\sigma, T, V, t$  and  $v$  in terms of these

eight numbers, we obtain:

P.19

$$T = M(c_1 c_1^* + s_1 s_1^*) + Q(s_1 c_1^* - c_1 s_1^*)$$

$$V = N(s_1 c_1^* + c_1 s_1^*) + P(c_1 c_1^* - s_1 s_1^*)$$

$$Ch\delta = (c_2 c_2^* - s_2 s_2^*)\sqrt{\gamma\gamma^*}; Ch\delta = (c_2 c_2^* + s_2 s_2^*)\sqrt{\gamma\gamma^*}$$

the expressions for  $t$  and  $v$  being obtained from those for  $T$  and  $V$  replacing in them the capital letters  $M, N, P, Q$  by small one.

The final rules for the computation of  $c_{kk}$ ,  $c_{mm}$  can be formulated as follows. Form the numbers  $G$  and  $H$  defined by

$$G = (M+P)c_1 c_1^* + (M-P)s_1 s_1^* + (N+Q)s_1 c_1^* + (N-Q)c_1 s_1^*$$

$$H = (M-P)c_1 c_1^* + (M+P)s_1 s_1^* - (N-Q)s_1 c_1^* - (N+Q)c_1 s_1^*$$

Then, replacing in the right hand members the capital letters  $M, N, P, Q$  by small ones  $m, n, p, q$ , form the corresponding numbers  $g$  and  $h$ .

$$c'_{kk} = \frac{2 \cdot \gamma\gamma^*}{\gamma + \gamma^*} [G \cdot c_2 c_2^* - H \cdot s_2 s_2^*]$$

$$c''_{kk} = \frac{2 \cdot \gamma\gamma^*}{\gamma + \gamma^*} [g \cdot c_2 c_2^* - h \cdot s_2 s_2^*]$$

$$(29) \begin{cases} c'_{mm} = \frac{2 \cdot \delta \delta^*}{\gamma + \gamma^*} \cdot [H \cdot c_2 c_2^* - G \cdot s_2 s_2^*] \\ c''_{mm} = \frac{2 \cdot \delta \delta^*}{\gamma + \gamma^*} \cdot [h \cdot c_2 c_2^* - g \cdot s_2 s_2^*] \end{cases}$$

Finally, the ~~diagonal~~ off diagonal elements of  $C$  are computed with the aid of formulas

$$\begin{aligned} \sqrt{\gamma^*} \cdot c_{kj} &= a_{kj} \cos \bar{z} + a_{mj} \sin \bar{z} \\ \sqrt{\gamma^*} \cdot c_{mj} &= -a_{kj} \sin \bar{z} + a_{mj} \cos \bar{z} \\ \sqrt{\gamma} \cdot c_{jk} &= a_{jk} \cos z + a_{jm} \sin z \\ \sqrt{\gamma} \cdot c_{jm} &= -a_{jk} \sin z + a_{jm} \cos z, \end{aligned}$$

where

$$\begin{aligned} \sin z &= (s_1 c_2 + i s_2 c_1) \sqrt{\gamma} \\ \cos z &= (c_1 c_2 - i s_1 s_2) \sqrt{\gamma} \\ \sin \bar{z} &= (s_1^* c_2^* + i s_2^* c_1^*) \sqrt{\gamma^*} \\ \cos \bar{z} &= (c_1^* c_2^* - i s_1^* s_2^*) \sqrt{\gamma^*} \end{aligned}$$

Therefore

$$\left. \begin{aligned} c'_{kj} &= (a'_{kj} c_1^* + a'_{mj} s_1^*) c_2^* - (a''_{kj} s_1^* - a''_{mj} c_1^*) s_2^* \\ c'_{mj} &= (a'_{mj} c_1^* - a'_{kj} s_1^*) c_2^* + (a''_{mj} s_1^* + a''_{kj} c_1^*) s_2^* \\ c'_{jk} &= (a'_{jk} c_1 + a'_{jm} s_1) c_2 + (a''_{jk} s_1 - a''_{jm} c_1) s_2 \\ c'_{jm} &= (a'_{jm} c_1 - a'_{jk} s_1) c_2 - (a''_{jm} s_1 + a''_{jk} c_1) s_2 \end{aligned} \right\} (30)$$

as well as

$$c_{kj}'' = (a_{kj}'' c_1^* + a_{mj}'' s_1^*) c_2^* - (a_{mj}' c_1^* - a_{kj}' s_1^*) s_2^*$$

$$c_{mj}'' = (a_{mj}'' c_1^* - a_{kj}'' s_1^*) c_2^* - (a_{mj}' s_1^* + a_{kj}' c_1^*) s_2^*$$

$$c_{jk}'' = (a_{jk}'' c_1 + a_{jm}'' s_1) c_2 + (a_{jm}' c_1 - a_{jk}' s_1) s_2$$

$$c_{jm}'' = (a_{jm}'' c_1 - a_{jk}'' s_1) c_2 + (a_{jm}' s_1 + a_{jk}' c_1) s_2$$

In other words, eight numbers

$$(31) \left\{ \begin{array}{ll} \alpha = a_{jk}' c_1 + a_{jm}' s_1 & \beta = a_{jm}' c_1 - a_{jk}' s_1 \\ \lambda = a_{jk}'' c_1 + a_{jm}'' s_1 & \mu = a_{jm}'' c_1 - a_{jk}'' s_1 \\ \alpha^* = a_{kj}' c_1^* + a_{mj}' s_1^* & \beta^* = a_{mj}' c_1^* - a_{kj}' s_1^* \\ \lambda^* = a_{kj}'' c_1^* + a_{mj}'' s_1^* & \mu^* = a_{mj}'' c_1^* - a_{kj}'' s_1^* \end{array} \right.$$

once computed, the eight numbers  $c_{kj}'$ ,  $c_{mj}'$  etc.

have the following form:

$$\left. \begin{array}{l} c_{kj}' = \alpha^* c_2^* + \mu^* s_2^* \\ c_{mj}' = \beta^* c_2^* + \lambda^* s_2^* \\ c_{jk}' = \alpha c_2 - \mu s_2 \\ c_{jm}' = \beta c_2 - \lambda s_2 \end{array} \right\} (32) \left\{ \begin{array}{l} c_{kj}'' = \lambda^* c_2^* - \beta^* s_2^* \\ c_{mj}'' = \mu^* c_2^* - \alpha^* s_2^* \\ c_{jk}'' = \lambda c_2 + \beta s_2 \\ c_{jm}'' = \mu c_2 + \alpha s_2 \end{array} \right.$$

§8. Discussion of exceptional cases  
when  $\omega = 0$ , or  $\omega^* = 0$ , or  $\omega = \omega^* = 0$ .

The general solution (§§ 6 & 7) holds for  $\omega, \omega^* \neq 0$ . Here we study the special six cases, when  $\omega, \omega^* = 0$ :

I)  $\omega \neq 0, \omega^* = 0$  ; II)  $\omega = 0, \omega^* \neq 0$ ;

III)  $\omega = \omega^* = 0$ , but  $E^2 = E^{*2} \neq 0$ ;

IV)  $\omega = \omega^* = E = E^* = 0$ , but  $nQ \neq 0$ ;

V)  $\omega = \omega^* = E = E^* = n = 0$ , but  $Q \neq 0$ ;

VI)  $\omega = \omega^* = E = E^* = Q = 0$ , but  $n \neq 0$ .

Case I)

In this case  $\omega^* = 0$ , that is,  $B - C = D + A = 0$ , but  $\omega > 0$  so that  $R^2 = E^{*2} = \omega^2 + E^2 > \omega^2 > 0$ . Thus  $|E^*| = R \neq 0$ . The equation for  $\tau = \tan \zeta$  becomes  $1 + \tau^2 = 0$ , since  $F^* = 0$ , and therefore  $\tau = \tan(\frac{\pi}{2} + iy) = \pm i$  which gives

$$(1 \mp \tanh y) \tan \xi \mp i = 0.$$

But  $|\tan \xi \mp i|^2 = \sec^2 \xi \geq 1$  and thus  $\tanh y = \pm 1$ ,  $y = \pm \infty$  and  $\gamma^* = \text{ch } 2y = +\infty$ . Computing  $S_2^*, G_2^*$  as limits for  $\gamma^* \rightarrow \infty$ , we find

that  $c_2^* = \frac{1}{\sqrt{2}}$ ,  $s_2^* = \sigma(E^*)/\sqrt{2}$ , the parameter P. 23

•  $\xi$  being arbitrary in this case. We choose  $\xi = 0$  because otherwise the sign of  $s_1^*$ , which is fixed by  $\delta(s_1^*) = \sigma(D+A)$  would remain indetermined since  $D+A=0$ . Thus, we have usual formulae <sup>(23)</sup> for  $s_1, c_1, s_2, c_2$ , while (27) are replaced by

$$s_1^* = 0, \quad c_1^* = 1, \quad s_2^* = \frac{\sigma(E^*)}{\sqrt{2}}, \quad c_2^* = \frac{1}{\sqrt{2}}$$

In this case (29) become

$$(29') \left\{ \begin{array}{l} c'_{kk} = R(c_{kk}) = \gamma \cdot \sqrt{2} \cdot [G \cdot c_2 - \sigma(E^*) \cdot H s_2] \\ c''_{kk} = \mathbf{Im}(c_{kk}) = \gamma \cdot \sqrt{2} \cdot [g \cdot c_2 - \sigma(E^*) \cdot h s_2] \\ c'_{mm} = R(c_{mm}) = \gamma \cdot \sqrt{2} \cdot [H c_2 - \sigma(E^*) \cdot G s_2] \\ c''_{mm} = \mathbf{Im}(c_{mm}) = \gamma \sqrt{2} [h c_2 - \sigma(E^*) \cdot g s_2] \end{array} \right.$$

with

$$G = (M+P)c_1 + (N+Q)s_1, \quad g = (m+p)c_1 + (n+q)s_1,$$

$$H = (M-P)c_1 - (N-Q)s_1, \quad h = (m-p)c_1 - (n-q)s_1,$$

the other elements of the transformed matrix  $C$  being given by the formulae (32) in which

• that,  $\alpha^* = a'_{kj}$ ,  $\beta^* = a'_{mj}$ ,  $\lambda^* = a''_{kj}$  and  $\rho^* = a''_{mj}$ , so

$$(33) \left\{ \begin{array}{l} \sqrt{2} \cdot c'_{kj} = a'_{kj} + a''_{mj} \cdot \sigma(E^*) ; \quad \sqrt{2} \cdot c''_{kj} = a''_{kj} - a'_{mj} \cdot \sigma(E^*) \\ \sqrt{2} \cdot c'_{mj} = a'_{mj} + a''_{kj} \cdot \sigma(E^*) ; \quad \sqrt{2} \cdot c''_{mj} = a''_{mj} - a'_{kj} \cdot \sigma(E^*) . \end{array} \right.$$

The expressions of  $c'_{jk}, c''_{jk}, c'_{jm}, c''_{jm}$  do not change. P. 24

Case II)

This case does not differ from I):  $\gamma = \infty, E \neq 0$  so that  $s_1^*, c_1^*, s_2^*, c_2^*$  are given by (27) but the expressions (20) are replaced by

$$(23') \quad s_1 = 0, \quad c_1 = 1, \quad s_2 = \sigma(E)/\sqrt{2}, \quad c_2 = 1/\sqrt{2}.$$

The formulae (29) become

$$(29'') \quad \begin{cases} c'_{kk} = \gamma^* \cdot \sqrt{2} [G c_2^* - \sigma(E) \cdot H s_2^*] & ; \quad c'_{mm} = \gamma^* \cdot \sqrt{2} [H c_2^* - \sigma(E) \cdot G s_2^*] \\ c''_{kk} = \gamma^* \cdot \sqrt{2} [g \cdot c_2^* - \sigma(E) \cdot h \cdot s_2^*] & ; \quad c''_{mm} = \gamma^* \cdot \sqrt{2} [h \cdot c_2^* - \sigma(E) \cdot g \cdot s_2^*] \end{cases}$$

where  $G = (M+P)c_1^* + (N-Q)s_1^*$  ;  $H = (M-P)c_1^* - (N+Q)s_1^*$   
 $g = (m+p)c_1^* + (n-q)s_1^*$  ;  $h = (m-p)c_1^* - (n+q)s_1^*$ .

The formulae (32) give  $c'_{kj}, c''_{kj}, c'_{mj}, c''_{mj}$ , while the expressions of  $c'_{jk}, c''_{jk}, c'_{jm}, c''_{jm}$  are replaced by

$$(34) \quad \begin{cases} c'_{jk} \sqrt{2} = a'_{jk} - a''_{jm} \cdot \sigma(E) & ; \quad c''_{jk} \sqrt{2} = a''_{jk} + a'_{jm} \cdot \sigma(E) \\ c'_{jm} \sqrt{2} = a'_{jm} - a''_{jk} \cdot \sigma(E) & ; \quad c''_{jm} \sqrt{2} = a''_{jm} + a'_{jk} \cdot \sigma(E). \end{cases}$$

Case III)

In this case  $\gamma = \gamma^* = \infty$ , so that  $\begin{cases} s_2 = \sigma(E)/\sqrt{2} \\ s_2^* = \sigma(E^*)/\sqrt{2} \end{cases}$   
 $c_1^* = c_1 = 1$  ;  $s_1^* = s_1 = 0$  ;  $c_2^* = c_2 = \frac{1}{\sqrt{2}}$  ;

The formulae (29) do not hold and the expressions of  $c_{kk}, c_{mm}$  are deduced from (28), observing that at the limit for  $\gamma, \gamma^* \rightarrow \infty$ , we have

$$\lim_{\gamma, \gamma^* \rightarrow \infty} \{ (\gamma \gamma^*)^{-\frac{1}{2}} \cdot Ch \sigma \} = c_2 c_2^* + s_2 s_2^* = \frac{1}{2} [1 + \sigma(E E^*)]$$

$$\lim_{\gamma, \gamma^* \rightarrow \infty} \{ (\gamma \gamma^*)^{-\frac{1}{2}} \cdot Ch \delta \} = c_2 c_2^* - s_2 s_2^* = \frac{1}{2} [1 - \sigma(E E^*)]$$



$$\lim_{f, g^* \rightarrow \infty} \left\{ \frac{Sh\sigma}{\sqrt{fg^*}} \right\} = S_2 C_2^* + S_2^* C_2 = \frac{1}{2} [\sigma(E) + \sigma(E^*)]$$

$$\lim_{f, g^* \rightarrow \infty} \left\{ \frac{Sh\delta}{\sqrt{fg^*}} \right\} = S_2 C_2^* - S_2^* C_2 = \frac{1}{2} [\sigma(E) - \sigma(E^*)]$$

Since in the case III)  $\theta = \xi = 0$ , we have  $T = M$ ,  $t = m$ ,  $U = Q$ ,  $u = q$ ,  $V = P$ ,  $v = p$ ,  $W = N$  and  $w = n$ . Therefore, the result is:

$$C'_{kk} = M + P + (M - P)\sigma(E E^*) - (n + q)\sigma(E) + (n - q)\sigma(E^*)$$

$$C''_{kk} = m + p + (m - p)\sigma(E E^*) + (N + Q)\sigma(E) - (N - Q)\sigma(E^*)$$

$$C'_{mm} = M - P + (M + P)\sigma(E E^*) - (n - q)\sigma(E) + (n + q)\sigma(E^*)$$

$$C''_{mm} = m - p + (m + p)\sigma(E E^*) + (N - Q)\sigma(E) - (N + Q)\sigma(E^*)$$

The elements  $G_{ik}$ ,  $C_{kj}$ ,  $G_{jm}$ ,  $C_{mj}$  are given by formulae (33) and (34).

#### Case IV)

Since in this case  $\omega = 0$ ,  $\omega^* = 0$  we see that  $B \pm C = 0$  and  $D \pm A = 0$ . Therefore, in this case all six numbers  $A$ ,  $B$ ,  $C$ ,  $D$ ,  $E$  and  $E^*$  vanish:

$$(35) \quad \left\{ \begin{array}{l} PQ + pg = 0 \\ NQ + ng = 0 \\ mQ - gM = 0 \end{array} \right. \quad \left\{ \begin{array}{l} PM + pm = 0 \\ MN + mn = 0 \\ pN - nP = 0 \end{array} \right.$$

Therefore

$$\frac{m}{M} = \frac{q}{Q} = -\frac{N}{n} = -\frac{P}{p} = \lambda \quad (35)$$

where  $\lambda$  is a real number. No transformation is needed when  $|a_{km}|^2 + |a_{mk}|^2 = 0$ , thus since

$$|a_{km}|^2 + |a_{mk}|^2 = 2(N^2 + Q^2 + n^2 + p^2) = 2(1 + \lambda^2)(n^2 + Q^2) > 0$$

we know that  $n^2 + Q^2 > 0$ ;  $n$  and  $Q$  cannot vanish simultaneously. In the case IV) we suppose that both  $n$  and  $Q$  are different from zero. We deduce from (35) that

$$\frac{a_{kk}}{M + ip} = \frac{a_{mm}}{M - ip} = \frac{a_{km}}{Q + in} = \frac{a_{mk}}{-Q + in} = 1 + i\lambda$$

The conditions  $c_{km} = 0$  and  $c_{mk} = 0$  reduce now to the only equation

$$(Q + in) \sin z \sin \zeta + (M + ip) \cos z \sin \zeta - (M - ip) \cos \zeta \sin z + (Q - in) \cos z \cos \zeta = 0$$

Since we have only one complex equation  $\zeta$  is arbitrary and we take  $\zeta = z$ . Thus  $t = \tau$  is a root of the quadratic equation

$$(36) \quad (Q + in)t^2 + 2ipt + Q - in = 0$$

and

$$(37) \quad t = \frac{R - ip}{Q^2 + n^2} (n + iQ) \quad \text{with } R = \sqrt{Q^2 + n^2 + p^2}.$$

Sow

$$\tan 2\theta = (t + \bar{t})(1 - t\bar{t})^{-1} = \frac{n}{p}$$

$$\operatorname{Ch} 2\varphi = -i(t - \bar{t})(1 + t\bar{t})^{-1} = \frac{Q}{R} \quad \text{which gives}$$

$$(38) \quad \begin{cases} \sin 2\theta = \frac{n}{\sqrt{p^2 + n^2}} & \delta = \cos 2\theta = \frac{p}{\sqrt{p^2 + n^2}} \quad (n \neq 0) \end{cases}$$

$$\operatorname{Sh} 2\varphi = \frac{Q}{\sqrt{p^2 + n^2}} \quad \gamma = \operatorname{Ch} 2\varphi = \frac{R}{\sqrt{p^2 + n^2}}$$

Taking as  $\theta$  an acute angle, we have

$$(38bis) \quad \begin{cases} c_1 = c_1^* = \left(\frac{1+\delta}{2}\right)^{\frac{1}{2}} > 0 & c_2 = c_2^* = \left(\frac{\gamma+1}{2\gamma}\right)^{\frac{1}{2}} \\ s_1 = s_1^* = \epsilon(n) \cdot \left(\frac{1-\delta}{2}\right)^{\frac{1}{2}} & s_2 = s_2^* = \epsilon(Q) \cdot \left(\frac{\gamma-1}{2\gamma}\right)^{\frac{1}{2}} \end{cases}$$

The signs of  $s_1$  and  $s_2$  are perfectly determined because of  $n \neq 0$  and  $Q \neq 0$ . If  $Q = 0$ , then

$\varphi = 0$ . Since  $\gamma = \gamma^*$  and  $\zeta = z$  the expressions (28bis)

$$\text{give } c'_{ik} = T + \gamma V \quad c'_{im} = T - \gamma V$$

$$c''_{ik} = t + \gamma v \quad c''_{im} = t - \gamma v$$

But  $s = 2\theta$ ,  $d = 0$  so that  $T = M$ ,  $t = m$

$$v \cdot \gamma = (p \cos 2\theta + n \sin 2\theta) \gamma = \sqrt{p^2 + n^2} \cdot R / \sqrt{p^2 + n^2} = R$$

$$V \cdot \gamma = (P \cos 2\theta + N \sin 2\theta) \gamma = R (P_p + N_n) / (p^2 + n^2) = -\lambda R = -\frac{Q}{R}$$

Therefore  $c_{kk}$  and  $c_{mm}$  are computed by

$$(39) \begin{cases} c'_{kk} = M - \frac{q}{Q}R & c'_{mm} = M + \frac{q}{Q}R \\ c''_{kk} = m + R & c''_{mm} = m - R \end{cases}$$

which proves that in the case considered

$$c_{kk} + c_{mm} = 2(M + im) = a_{kk} + a_{mm}.$$

The formulae (32) yield  $c_{jk}$ ,  $c_{jm}$ ,  $c_{kj}$  and  $c_{mj}$ .

### Case V

Adding to (35) also  $n=0$  and observing that  $n^2 + Q^2 > 0$ , so that now certainly  $Q \neq 0$ , we obtain from  $NQ + ng = 0$  that also  $N=0$ . The equation (36) becomes

$$Q \cdot t^2 + 2ipt + Q = 0$$

so that (37) gives ( $\zeta = z$ )

$$t = \tan(\theta + iq) = \tan(\xi + iy) = \frac{iQ(R-p)}{Q^2}$$

with  $R = \sqrt{Q^2 + p^2}$ , so that  $Q^2 = R^2 - p^2$ . Thus

$$\tan(\theta + iq) = t = i \cdot \frac{R-p}{Q} = \frac{iQ}{R+p}$$

This complex equation can be written as

$$(40) \sin\theta[(R+p)Chq - QShq] + i\cos\theta[(R+p)Shq - QChq] = 0$$

The two solutions are a) and b):

$$\bullet \quad a) \quad \theta = 0 \quad \text{and} \quad \tanh \varphi = \frac{Q}{R+p} = \sigma(Q) \sqrt{\frac{R-p}{R+p}}$$

$$b) \quad \theta = \frac{\pi}{2} \quad \text{and} \quad \tanh \varphi = \frac{R+p}{Q} = \sigma(Q) \sqrt{\frac{R+p}{R-p}}$$

Since  $|\tanh \varphi| \leq 1$ , we distinguish three subcases V1), V2) and V3) which correspond to  $p > 0$ ,  $p = 0$  and  $p < 0$  respectively. If

$p > 0$  then the solution a) holds, but b) does not hold. When  $p = 0$  both solutions are acceptable. If  $p < 0$  the solution a) does not work and it is the solution b) which holds. But in all three subcases the same values for  $C_2 = C_2 \varphi$  and  $S_2 = S_2 \varphi$  are obtained, namely

$$(41) \quad C_2 = C_2^* = \left[ \frac{1}{2} \left( 1 + \frac{|p|}{R} \right) \right]^{\frac{1}{2}}; \quad S_2 = S_2^* = \sigma(Q) \left[ \frac{1}{2} \left( 1 - \frac{|p|}{R} \right) \right]^{\frac{1}{2}}$$

For  $p \neq 0$  we have  $\gamma = \frac{R}{|p|}$ , while  $\gamma = \infty$

When  $p=0$ . In the case  $p=0$ , the equation P. 30 (40) becomes

$$[S \operatorname{th} \varphi - \sigma(Q) \cdot C \operatorname{th} \varphi] e^{i\theta \sigma(Q)} = 0$$

so that  $\operatorname{th} \varphi = \sigma(Q)$  and  $\varphi = \sigma(Q) \cdot \infty$ , while  $\theta$  is any. If we express the solutions a) and b) valid for  $p \neq 0$  as follows

$$\theta = \frac{\pi}{4} [1 - \sigma(p)] \quad (\sigma(0) = 0)$$

then  $\theta = \frac{\pi}{4}$ , if  $p=0$ . For  $p \geq < 0$  we have now

$$(42) \begin{cases} S_1 = S_1^* = \frac{1}{2} [1 - \sigma(p)], & \text{if } p \neq 0, & S_1 = S_1^* = \frac{\sqrt{2}}{2}, & p=0 \\ C_1 = C_1^* = \frac{1}{2} [1 + \sigma(p)], & \text{if } p \neq 0, & C_1 = C_1^* = \frac{\sqrt{2}}{2}, & \text{if } p=0. \end{cases}$$

The four numbers,  $C_2 = C_2^*$ ,  $S_2 = S_2^*$  are given by (41) for  $p \geq < 0$ . Formulae (39) hold if  $p \neq 0$ .

If  $p=0$ , they become.

$$(43) \begin{cases} C'_{\text{max}} = M - g \cdot \sigma(Q) & C'_{\text{min}} = M + g \cdot \sigma(Q) \\ C''_{\text{max}} = m + Q \cdot \sigma(Q) & C''_{\text{min}} = m - Q \cdot \sigma(Q) \end{cases}$$

and the other elements of  $C$  are given by (32).

### Case VI)

Now  $Q=0$ , but  $n \neq 0$ . The equation (36)

$$(36) \quad n \cdot t^2 + 2p \cdot t - n = 0$$

gives

$$t = \tan(\theta + i\varphi) = \sigma(n) \cdot \left( \frac{R-p}{R+p} \right)^{\frac{1}{2}}, \text{ which}$$

is real. Therefore,  $\varphi = 0$  and

P. 31

$$\tan \theta = \sigma(n) \sqrt{\frac{R-p}{R+p}}$$

which gives  $\delta = \cos 2\theta = p/R$  and thus

$$(38 \text{ ter}) \begin{cases} s_1 = s_1^* = \sigma(n) \left[ \frac{1}{2} \left( 1 - \frac{p}{R} \right) \right]^{\frac{1}{2}}, & s_2 = s_2^* = 0 \\ c_1 = c_1^* = \left[ \frac{1}{2} \left( 1 + \frac{p}{R} \right) \right]^{\frac{1}{2}}, & c_2 = c_2^* = 1 \end{cases}$$

Since  $NQ + nq = 0$ , the vanishing of  $Q$  coupled with  $n \neq 0$  entails  $q = 0$ . The formulae (39) become indetermined. But for  $Q \rightarrow 0$  we

have from  $NQ + nq = 0$  the proportion

$$\frac{q}{Q} = -\frac{N}{n}$$

so that (39) are equivalent to

$$(44) \begin{cases} c'_{kk} = M + \frac{N}{n} R & c'_{mm} = M - \frac{N}{n} R \\ c''_{kk} = m + R & c''_{mm} = m - R \end{cases}$$

which hold in the case VII. Together with (32) the formulae (44) define the transformed matrix  $C$ .

## §9. Convergence of the Diagonalization method and the Speed of convergence.

P.32

At each step the amount  $|a_{mk}|^2 + |a_{km}|^2$  of two offdiagonal terms vanishing in the transformed matrix  $C$  is transferred from the sum of moduli squared of offdiagonal elements

$$\sum_{i \neq j} |a_{ij}|^2$$

of the matrix  $A$  into the sum of moduli squared of diagonal elements of the transformed matrix  $C$ . We have indeed in the relation (14)  $c_{km} = c_{mk} = 0$  so that

$$|c_{kk}|^2 + |c_{mm}|^2 = |a_{kk}|^2 + |a_{mm}|^2 + |a_{km}|^2 + |a_{mk}|^2$$

This fact proves the convergence of our method since the norm is an invariant and remains constant.

But the convergence is not sufficient by itself to insure the practical importance



of diagonalization method in the solution of systems of linear equations. P. 33  
If the convergence speed slows down when the number of steps increases it may be impossible in practice to reduce the moduli of all off-diagonal terms below a prescribed sufficiently small quantity characterizing the desired accuracy.

It is a favourable circumstance that the speed of convergence of our method does not depend on the number of performed steps and remains constant whatever large the number of iterations becomes.

To show it let us study what happens to the modulus  $|a_{kj}|$  of an off-diagonal term  $a_{kj}$  when it is first made to ~~be~~ vanish, the first step being defined by  $c_{kj} = c_{jk} = 0$ , and then reappears again as  $d_{kj}$  after the second step transforming  $C = (c_{ij})$  into

$D = ((d_{ij}))$  is completed. The second step must involve one of two subscripts  $k, j$ , otherwise the element  $c_{kj} = 0$  would remain equal to zero. To fix the ideas, we suppose that the second step makes vanish  $d_{km}$  and  $d_{mk}$ .

Consider now the transforms  $d_{kj}, d_{mj}, d_{jk}$  and  $d_{jm}$  of four elements  $a_{kj}, a_{mj}, a_{jk}$  and  $a_{jm}$ . The succession of our two steps can be represented as follows:

$$a_{kj} \rightarrow 0 \rightarrow d_{kj} \quad ; \quad a_{mj} \rightarrow c_{mj} \rightarrow d_{mj}$$

$$a_{jk} \rightarrow 0 \rightarrow d_{jk} \quad ; \quad a_{jm} \rightarrow c_{jm} \rightarrow d_{jm}$$

$$a_{km} \rightarrow c_{km} \rightarrow 0 \quad ; \quad a_{mk} \rightarrow c_{mk} \rightarrow 0$$

The relations (16) p. 11 give

$$|d_{jk}|^2 + |d_{jm}|^2 = |c_{jm}|^2$$

$$|d_{kj}|^2 + |d_{mj}|^2 = |c_{mj}|^2,$$

where

$$\gamma^* |c_{jm}|^2 = |a_{jm} \cos \xi + a_{km} \sin \xi|^2 = \gamma^* f_2^*(\bar{z})$$

$$\gamma |c_{mj}|^2 = |a_{mj} \cos \zeta + a_{mk} \sin \zeta|^2 = \gamma f_2(\underline{z})$$

Computing  $f_2(z) = |c_{mj}|^2$  and using the polar form of  $a_{mj} = |a_{mj}| \cdot e^{i\alpha_{mj}}$  and  $a_{mk} = |a_{mk}| \cdot e^{i\alpha_{mk}}$ , we find that

$$(45) \quad 2|c_{mj}|^2 = (|a_{mj}|^2 + |a_{mk}|^2) + (|a_{mj}|^2 - |a_{mk}|^2) \cdot \cos 2\theta + 2|a_{mj} \cdot a_{mk}| \left\{ \frac{\sin 2\theta}{\gamma} \cos(\alpha_{mj} - \alpha_{mk}) + \text{th } 2\varphi \cdot \sin(\alpha_{mj} - \alpha_{mk}) \right\}$$

A similar expression gives  $f_1(\bar{\xi}) = |c_{jm}|^2$  and it is obtained from (45) replacing  $\theta, \varphi, a_{mj}$  and  $a_{mk}$  by  $\xi, -\eta, a_{jm}$  and  $a_{km}$  respectively.

Each pair of symmetric elements (symmetric with respect to the principal diagonal) is annihilated and reappears once during a cycle. When the number of cycles increases the average values of  $\cos 2\theta$  and of the braces

$$\frac{\sin 2\theta}{\gamma} \cos(\alpha_{mj} - \alpha_{mk}) + \text{th } 2\varphi \cdot \sin(\alpha_{mj} - \alpha_{mk})$$

tend to zero. Therefore, if the number of cycles is large, it can be assumed that in the average the two last terms in (45) do not contribute to final values of moduli

and these final values are obtained from the approximate expressions

$$|c_{jm}|^2 \cong \frac{1}{2} (|a_{jm}|^2 + |a_{km}|^2)$$

$$|c_{mj}|^2 \cong \frac{1}{2} (|a_{mj}|^2 + |a_{mk}|^2)$$

which thus characterize the (effect of one cycle when a group of cycles is considered. This gives, for one cycle

$$|d_{jk}|^2 + |d_{jm}|^2 + |d_{kj}|^2 + |d_{mj}|^2 \cong$$

$$\cong \frac{1}{2} (|a_{jm}|^2 + |a_{km}|^2 + |a_{mj}|^2 + |a_{mk}|^2).$$

When a global effect on the average modulus  $\mu$  of all  $(n-1)n$  off-diagonal terms

$$\mu = \left\{ \frac{1}{(n-1)n} \sum_{i \neq j} |a_{ij}|^2 \right\}^{\frac{1}{2}}$$

is considered, this result means that  $\frac{n(n-1)}{2}$  times per cycle  $\mu$  is multiplied by

$$\left[ 1 - \frac{2}{(n-1)n} \right]^{\frac{1}{2}}$$

Since four squares of moduli of elements of  $D = ((d_{ij}))$  are equal approximately to only two squares of moduli of  $a_{ij}$ ,  $i \neq j$ .

Therefore, two among  $n(n-1)$  squares p. 37  
 of moduli of off-diagonal terms are lost  
 which means that at each step  $\mu$  is multi-  
 plied by  $\left[1 - \frac{2}{(n-1)n}\right]^{\frac{1}{2}}$ . After the completion  
 of a cycle  $\mu$  becomes

$$\mu \left[1 - \frac{2}{(n-1)n}\right]^{\frac{n(n-1)}{4}} \approx \mu \cdot e^{-\frac{1}{2}}$$

when  $n$  is large, since there are  $\frac{n(n-1)}{2}$   
 steps in each cycle. After  $p$  cycles the  
 mean modulus  $\mu$  of off-diagonal terms  
 decreases  $e^{p/2}$  times. Thus, for instance,  
 after 46 cycles it decreases  $10^{10}$  times since  
 $e^{-23} \sim 10^{-10}$ . If, at the beginning,  $\mu$   
 was less than one or at most equal to one,  
 after 46 cycles  $\mu$  will be less than  $10^{-10}$ .

We see that the speed of convergence  
 is characterized by a constant factor  
 $e^{-\frac{1}{2}}$  and it does not depend on the number  
 of cycles already performed.

## § 10.

In this section we discuss the application of diagonalization method to special classes of matrices. Six such special classes are considered: skew symmetric complex, symmetric complex, general real, skew hermitian, hermitian and real symmetric matrices.

In each of these six particular cases the general procedure undergoes some simplification and the purpose of this section is to describe the simplifications brought in our method by special properties of matrices considered.

### Skew symmetric

If the original matrix  $A$  is skew symmetric then for each pair of elements  $a_{km}, a_{mk}$  we have  $a_{km} + a_{mk} = 0$ , so that  $N = n = 0$ , as well as  $a_{kk} = 0$  and therefore  $M = m = P = p = 0$ . The two remaining quantities  $Q, q$  verify the condition  $Q^2 + q^2 > 0$  since  $a_{km} = -a_{mk} = Q + iq$  cannot vanish. All six numbers  $A, B, C, D, E$  and  $E^*$  vanish and the conditions  $c_{km} = c_{mk} = 0$  are reduced to a single equation  $\frac{1}{\sqrt{2}} \cos(x - z) = 0$ .

$T' \rightarrow \zeta = z + \frac{\pi}{2}$  and  $\varphi = \eta$ . It is easy to prove that the transformation does not preserve the skew symmetry of the original complex matrix. If  $\varphi = \eta$  is finite  $c_{jk} \neq -c_{kj}$  and  $c_{kk} \neq 0, c_{mm} \neq 0$ . We have indeed  $\sin \zeta = \cos z$  and  $\cos \zeta = -\sin z$ . Computing by (12)  $c_{kj}$  and observing that  $\gamma = \gamma^*$ , we have  $(a_{ij} = -a_{ji})$ :

$$\sqrt{\gamma} \cdot c_{kj} = a_{jk} \sin \bar{z} - a_{jm} \cos \bar{z}$$

while

$$-\sqrt{\gamma} \cdot c_{jk} = -a_{jk} \cos z - a_{jm} \sin z$$

To preserve the skew symmetry we should have  $\cos \bar{z} = \sin z$  as well as  $\sin \bar{z} = -\cos z = -\sin \bar{z}$

so that  $\sin \bar{z} = \sin z = 0$  and also  $\cos z = 0$  which is impossible since for finite  $z$  we have  $\sin^2 z + \cos^2 z = 1$ . Taking now  $\varphi = \eta$

infinite,  $\varphi = \eta = +\infty$  for instance, we obtain  $c_2 = c_2^* = s_2 = s_2^* = \frac{1}{\sqrt{2}}$  so that the conditions  $c_{km} = c_{mk} = 0$  are verified:

$$\left[ \frac{\cos(z - \zeta)}{\sqrt{\gamma \gamma^*}} \right]_{\gamma = \gamma^* = \infty} = 0,$$

but now  $\frac{\sin z}{\sqrt{\gamma}} \xrightarrow{\gamma \rightarrow \infty} s_1 c_2 + i c_1 s_2$  at the limit for  $\gamma = \infty$  yields  $(s_1 + i c_1) / \sqrt{2}$ . Forming the analogous expressions for the limits of

$\frac{\sin z}{\sqrt{y^*}}$ ,  $\frac{\cos z}{\sqrt{y}}$  and  $\frac{\cos z}{\sqrt{y^*}}$ , we obtain

$$\sqrt{2} \cdot c_{kj} = a_{kj} (c_1^* + i s_1^*) + a_{mj} (s_1^* - i c_1^*)$$

$$\begin{aligned} \sqrt{2} \cdot c_{jk} &= a_{jk} (c_1 - i s_1) + a_{jm} (s_1 + i c_1) = \\ &= a_{kj} (-c_1 + i s_1) - a_{mj} (s_1 + i c_1) \end{aligned}$$

So that

$$-\sqrt{2} \cdot c_{jk} = a_{kj} (c_1 - i s_1) + a_{mj} (s_1 + i c_1)$$

To preserve the skew symmetry for any pair  $(a_{kj}, a_{mj})$ , we should have at the same time

$c_1 = c_1^*$ ,  $s_1 = -s_1^*$  and also  $s_1 = s_1^*$ ,  $c_1 = -c_1^*$  which is impossible, since it entails  $c_1 = s_1 = 0$ .

Therefore, diagonalizing a skew symmetric matrix, no simplification is possible and in the process of diagonalization the skew symmetry is lost.

### §11. Symmetric complex matrix

To the contrary, the symmetry is preserved. If  $a_{km} = a_{mk}$ , then  $Q = q = 0$  and



$A = C = 0$ , while  $E + E^* = 0$ . We have in [P. 41]  
 this case  $\omega^2 = \omega^{*2} = B^2 + D^2$ . Therefore

$$\sin 2\theta = \sin 2\xi = \frac{D}{\omega} \quad \delta = \delta^* = \cos 2\theta = \cos 2\xi = \frac{B}{\omega}$$

So that  $\theta = \xi$ . But

$$\operatorname{Sh} 2\varphi = \frac{E}{\omega} = -\frac{E^*}{\omega} = -\operatorname{Sh} 2\eta \quad \text{so}$$

that  $\eta = -\varphi$  and  $\zeta = \bar{z}$ . Since  $c_i^* = c_i$ ,  $s_i^* = s_i$   
 and  $a_{ij} = a_{ji}$ , we conclude from (31) that

$$\alpha^* = \alpha, \beta^* = \beta, \lambda^* = \lambda, \mu^* = \mu$$

Moreover, we have  $c_2^* = c_2$  and  $s_2^* = -s_2$ . Thus,

the expressions (32) prove that the symmetry  
 is preserved since  $c_{kj} = c_{jk}$  and  $c_{mj} = c_{jm}$ .

There is no difference with respect to general  
 case insofar as the computation of  $c_{jk}$  and  $c_{jm}$  is  
 concerned, but the expressions of  $c_{kk}$  and  $c_{kk}$   
 can be simplified as follows. The functions  
 $G, g, H$  and  $h$  in (29) become

$$G = M + (\delta \cdot P + \frac{DN}{\omega}) \quad , \quad H = M - (\delta \cdot P + \frac{DN}{\omega})$$

$$g = m + (\delta \cdot p + \frac{Dn}{\omega}) \quad , \quad h = m - (\delta \cdot p + \frac{Dn}{\omega})$$

and the formulae (29) are transformed into:

$$c'_{kk} = \gamma \cdot M + (\delta \cdot P + \frac{DN}{\omega}) \quad c'_{mm} = \gamma \cdot M - (\delta \cdot P + \frac{DN}{\omega})$$

$$c''_{kk} = \gamma \cdot m + (\delta \cdot p + \frac{Dn}{\omega}) \quad c''_{mm} = \gamma \cdot m - (\delta \cdot p + \frac{Dn}{\omega})$$

where  $\gamma = \gamma^* = \frac{R}{\omega}$ . Thus,

$$c_{kk} + c_{mm} = \gamma \cdot (a_{kk} + a_{mm})$$

### §12. Real general matrix

If the elements  $a_{ij}$  of  $A$  are real, then

$$m = n = p = q = E = E^* = \varphi = \gamma = 0, \quad \gamma = \gamma^* = 1,$$

$$z = \theta, \quad \zeta = \xi, \quad c_2 = c_2^* = 1, \quad s_2 = s_2^* = 0, \quad A = PQ,$$

$$B = PM, \quad C = NQ, \quad D = MN, \quad \omega = \omega^* = R = (A^2 + B^2 + C^2 + D^2)^{\frac{1}{2}}$$

so that  $R = R_1 \cdot R_2$ , where

$$R_1^2 = M^2 + Q^2; \quad R_2^2 = N^2 + P^2.$$

Let us denote by  $\underline{s}$  and  $\underline{d}$  the sum  $z + \zeta = \theta + \xi = s$  and the difference  $z - \zeta = \theta - \xi = d$ .

Then the conditions  $c_{km} = c_{mk} = 0$  give

$$\tan s = \frac{N}{P} \quad \tan d = \frac{Q}{M}$$

and therefore

$$(46) \quad \frac{\sin s}{N} = \frac{\cos s}{P} = \frac{1}{R_2}; \quad \frac{\sin d}{Q} = \frac{\cos d}{M} = \frac{1}{R_1}$$

This yields first

43

$$\delta = \cos 2\theta = \cos(s+d) = \frac{MP - NQ}{R_1 R_2}$$

$$\delta^* = \cos 2\xi = \cos(s-d) = \frac{MP + NQ}{R_1 R_2}$$

and then

$$(47) \quad s_1 = \epsilon \left( \frac{1-\delta}{2} \right)^{\frac{1}{2}} \quad ; \quad c_1 = \eta \left( \frac{1+\delta}{2} \right)^{\frac{1}{2}}$$

$$s_1^* = \epsilon^* \left( \frac{1-\delta^*}{2} \right)^{\frac{1}{2}} \quad ; \quad c_1^* = \eta^* \left( \frac{1+\delta^*}{2} \right)^{\frac{1}{2}}$$

where  $\epsilon, \epsilon^*, \eta, \eta^* = \pm 1$  are the signs of  $s_1, s_1^*, c_1, c_1^*$  to be determined in accordance with (46), that is, in accordance with

$$\sigma(\sin s) = \sigma(N), \quad \sigma(\cos s) = \sigma(P), \quad \sigma(\sin d) = \sigma(Q), \quad \sigma(\cos d) = \sigma(M).$$

Using the trigonometric relations such as  $\sin 2\theta = 2 \sin \theta \cos \theta = \sin(s+d)$  or  $2 \sin \theta \sin \xi = \cos d - \cos s$ , we deduce the following relations between the signs of  $M, N, P, Q$  and  $s_1, c_1, s_1^*, c_1^*$ :

$$(48) \quad \begin{cases} \epsilon \eta = \sigma(\sin 2\theta) = \sigma(MN + PQ) \\ \epsilon \epsilon^* = \sigma(\sin \theta \sin \xi) = \sigma(MR_2 - PR_1) \\ \epsilon \eta^* = \sigma(\sin \theta \cos \xi) = \sigma(NR_1 + QR_2) \end{cases}$$

Taking  $\epsilon = \sigma(MN - PQ)$ , we obtain

$$(49) \quad \begin{cases} \eta = \sigma(M^2 N^2 - P^2 Q^2) \\ \epsilon^* = \sigma(NR_1 + QR_2) \\ \eta^* = \sigma(MR_2 - PR_1) \end{cases}$$

as it can be seen using the identity P. 44

$$(50) \quad (MN + PQ)(MR_2 + PR_1) \equiv (NR_1 + QR_2)(R_1R_2 + MP - NQ)$$

and observing that  $|MP \pm NQ| < R_1R_2$ . Changing in it the sign of  $M$  or of  $P$  or of  $Q$ , we deduce other identities which are useful in checking the validity of the choice of signs in (49). In checking that (49) gives the true signs to  $\sin s$ ,  $\cos s$ ,  $\sin d$  and  $\cos d$ , it is useful to observe that the inequalities  $|MN| \geq |PQ|$  entail the corresponding inequalities  $|N| \cdot R_1 \geq |Q| \cdot R_2$  and  $|M| \cdot R_2 \geq |P| \cdot R_1$ .

It is essential to observe that the choices

$$\epsilon = \sigma(MN - PQ) \quad \text{and} \quad \eta = \sigma(M^2N^2 - P^2Q^2)$$

become meaningless if  $|MN| = |PQ|$ . Therefore, we suppose that  $|MN| \neq |PQ|$ .

We now simplify the rules of sign, separating the cases  $|MN| > |PQ| \geq 0$  and  $|PQ| > |MN| \geq 0$ .

If  $|MN| > |PQ| \geq 0$ , then  $\eta = +1$ ,  $\epsilon^* = \sigma(N)$ , while  $\eta^* = \sigma(M)$ . The value of  $\epsilon$  depends on the sign of  $MN$ : if  $MN < PQ \leq 0$ , then  $\epsilon = -1$ , if  $MN > PQ \geq 0$ , then  $\epsilon = +1$ . Thus, we conclude that  $\epsilon = \sigma(MN)$ .

If  $|PQ| > |MN| \geq 0$ , then  $\eta = -1$ ,

$\epsilon^* = \sigma(Q)$ ,  $\eta^* = -\sigma(P)$ , while  $\epsilon = -\sigma(PQ)$ .

But nothing is changed, if we replace all four signs  $\epsilon, \eta, \epsilon^*, \eta^*$  by opposite ones since it corresponds to the addition of  $\pi$  to  $\theta$  and to  $\xi$ , so that the conditions are verified since  $\sin s, \cos s, \sin d, \cos d$  do not change. We <sup>will</sup> use therefore in the case  $|PQ| > |MN| \geq 0$  the signs  $\epsilon = \sigma(PQ)$ ,  $\eta = 1$ ,  $\epsilon^* = -\sigma(Q)$  and  $\eta^* = \sigma(P)$ . To summarize:

	If, then	$\epsilon =$	$\epsilon^* =$	$\eta =$	$\eta^* =$
(51)	$ MN  >  PQ  \geq 0$	$\sigma(MN)$	$\sigma(N)$	+1	$\sigma(M)$
	$ PQ  >  MN  \geq 0$	$\sigma(PQ)$	$-\sigma(Q)$	+1	$\sigma(P)$

If  $|MN| = |PQ|$ , we distinguish between two entirely different cases  $|MN| = |PQ| = 0$  and  $|MN| = |PQ| > 0$ . The first one is a singular case and it will be studied separately. The case  $|MN| = |PQ| > 0$  will be discussed now, subdividing it into two subcases:  $MN = PQ \neq 0$  and  $MN = -PQ \neq 0$ .

Case  $MN = PQ \neq 0$

The identity (50) with a change of sign in  $Q$  proves that in this case  $NR_1 = QR_2$  and also  $PR_1 = MR_2$ . Therefore  $\sigma(NQ) = \sigma(MP) = 1$  and  $\delta^* = \cos 2\xi = (MP + NQ)R_1^{-1}R_2^{-1} = 1$ , so that  $\xi = 0$  or  $\xi = \pi$ . Both values of  $\xi$  give the same result since, if  $\xi = 0$ ,  $s = d = \theta$  and therefore

$$(\xi = 0) \quad \eta^* = 1, \quad \epsilon = \sigma(s) = \sigma(N), \quad \eta = \sigma(\epsilon) = \sigma(M)$$

while  $\xi = \pi$  gives  $\theta = s - \pi = d + \pi$  and

$$(\xi = \pi) \quad \eta^* = -1, \quad \epsilon = -\sigma(N), \quad \eta = -\sigma(M)$$

and changing the signs into opposite ones ( $s_1^* = 0$  and therefore  $\epsilon^*$  does not matter)

we obtain the same result. Multiplying  $\eta^* = 1$ ,  $\epsilon = \sigma(N)$ ,  $\eta = \sigma(M)$  by  $\eta = \sigma(M)$  we see that it is equivalent to

$$\epsilon = \sigma(MN), \quad \eta = +1, \quad \eta^* = \sigma(M)$$

or to

$$\epsilon = \sigma(PQ), \quad \eta = +1, \quad \eta^* = \sigma(P)$$

Since  $MN = PQ$  and  $\sigma(MP) = 1$ .

This analysis shows that both rules deduced in 1) and 2) hold for  $MN = PQ \neq 0$  as it should be since the case  $MN = PQ$  is a limiting case for 1) as well as for 2) when  $|MN| \rightarrow |PQ|$ . The apparent contradiction between  $\sigma(N) = \sigma(Q)$  and the values  $\sigma(N), -\sigma(Q)$  for  $\epsilon^*$  is resolved observing that, for  $MN = PQ$ ,  $s_1^* = \sin \xi$  vanishes so that  $\epsilon^*$  has no meaning.

Case  $MN = -PQ \neq 0$

Now  $NR_1 + QR_2 = 0$  and  $\sigma(MP) = -\sigma(NQ) = +1$ .

Therefore  $\delta = \cos 2\theta = +1$  and  $\theta = 0$  or  $\theta = \pi$ .

If  $\theta = 0$ , then  $s = -d = \xi$  so that  $s_1^* = \sin s$ ,

$c^* = \cos s$  and thus  $(s_1 = 0)$ :

( $\alpha$ )  $\eta = \sigma(c_1) = 1$ ,  $\epsilon^* = \sigma(s_1^*) = \sigma(N)$ ,  $\eta^* = \sigma(c_1^*) = \sigma(M)$

while  $\theta = \pi$  yields  $\xi = \pi - d$  and thus

$\eta = \sigma(c_1) = -1$ ,  $\sigma(s_1^*) = \epsilon^* = -\sigma(N)$ ,  $\sigma(c_1^*) = \eta^* = -\sigma(M)$

Multiplying this by  $-1$  we come back to to ( $\alpha$ ), while  $\sigma(NQ) = -1$  and  $\sigma(MP) = +1$  give an equivalent system of signs, namely

$$\eta = 1 \quad \epsilon^* = -\delta(Q) \quad \eta^* = \delta(P) \quad \text{P.48}$$

Again we see that both rules 1) and 2) hold. ●  
 Thus, the table (51) is justified <sup>also</sup> for  $|MN| = |PQ|$   
 provided that  $MNPQ \neq 0$ .

### Singular Case $MN = PQ = 0$

Since  $a_{km}^2 + a_{mk}^2 \neq 0$ ,  $N$  and  $Q$  cannot vanish together and  $N^2 + Q^2 \neq 0$ . At least two of four quantities  $M, N, P, Q$  vanish, if  $MN = PQ = 0$ , but it may happen that three of them vanish in which case we will have  $N \neq 0$  or  $Q \neq 0$ . ●

Therefore, we have in all three sub-cases when two quantities vanish, namely

$$1) N = P = 0, Q \cdot M \neq 0 \quad \therefore (a_{kk} = a_{mm}, a_{mk} = -a_{km})$$

$$2) M = Q = 0, N \cdot P \neq 0 \quad \therefore (a_{mm} = -a_{kk}, a_{km} = a_{mk})$$

$$3) M = P = 0, N \cdot Q \neq 0 \quad \therefore (a_{kk} = a_{mm} = 0)$$

and two cases when three of them vanish:

$$4) M = P = N = 0, Q \neq 0: \left. \begin{array}{l} (a_{km} + a_{mk} = 0) \\ (a_{kk} = a_{mm} = 0) \end{array} \right\}$$

$$5) M = P = Q = 0, N \neq 0: (a_{km} = a_{mk})$$



As we will show in all these five P. 49

● Subcases  $s_1^* = 0$  and  $c_1^* = 1$ , while  $\sigma(s_1)$  and  $\sigma(c_1)$  are given in the following table:

	1) $NQ \neq 0, N=P=0$	2) $NP \neq 0, M=Q=0$	3) $NQ \neq 0, M=P=0, Q \neq 0$	4) $M=N=P=0, N \neq 0, M=P=Q=0$	5)
$\sigma(s_1)$	$\sigma(Q)$	$\sigma(N)$	$\sigma(Q)$	$\sigma(Q)$	$\sigma(N)$
$\sigma(c_1)$	$\sigma(M)$	$\sigma(P)$	$(c_1=0)$	$(c_1=0)$	$(c_1=0)$

To justify this table we observe that in the four cases 1), 2), 4) and 5) only one of the two conditions  $\tan s = \frac{N}{P}$ ,  $\tan d = \frac{Q}{M}$  remains, the other becoming indetermined. Therefore, in these four cases one of two parameters  $\xi, \theta$  can be chosen arbitrarily. We choose  $\xi = 0$ , so that  $s_1^* = 0, c_1^* = 1$ . In the case 3) we obtain  $\cos s = 0$  and  $\cos d = 0$  so that the two conditions reduce to

$$\cos \theta \cdot \cos \xi = 0 \quad ; \quad \sin \theta \cdot \sin \xi = 0.$$

We choose again the solution in which  $\sin \xi = 0$  and take  $\xi = \frac{\pi}{2}(1 - \sigma(NQ)) = 0$  or  $\pi$  according to  $NQ > 0$  or  $NQ < 0$ , which gives  $c_1^* = \sigma(NQ)$  and  $s_1^* = 0$ . In the cases 1), 2), 4) and 5) we have  $s = d = \theta$ , so that in the case 1)  $\tan \theta = \tan d = \frac{Q}{M}$  yields

● Case 1)  $s_1 = \frac{Q}{R_1}, \quad c_1 = \frac{M}{R_1}$

The case 2) gives likewise  $\tan \theta = \frac{N}{P}$ : P. 50

$$\text{Case 2)} \quad s_1 = \frac{N}{R_2} \quad ; \quad c_1 = \frac{P}{R_2}$$

In the case 4) only  $Q$  does not vanish, therefore  $\cot \alpha = \cot \theta = \frac{M}{Q} = 0$  and thus  $c_1 = \cos \theta = 0$ .

Taking  $\theta = \frac{\pi}{2} \sigma(Q)$ , we justify  $s_1 = \sigma(Q)$ ,  $c_1 = 0$ .

In the case 5) only  $N$  is different from zero. Thus,  $\cos \theta = \cos \theta = 0$  again and, taking  $\theta = \frac{\pi}{2} \sigma(N)$  we have  $s_1 = \sigma(N)$ ,  $c_1 = 0$ .

Remains the case 3) in which  $\xi = \frac{\pi}{2} [1 - \sigma(NQ)]$  gives first  $s_1^* = 0$  and  $c_1^* = \sigma(NQ)$ . To verify the second equation  $\cos \theta \cdot \cos \xi = 0$ , we must have  $c_1 = \cos \theta = 0$  and we take  $\theta = \frac{\pi}{2} \sigma(N)$ , so that

$$s_1 = \sigma(N), \quad c_1 = 0, \quad s_1^* = 0, \quad c_1^* = \sigma(NQ)$$

Multiplying these four numbers by  $\sigma(NQ)$ , we obtain the equivalent system

$$\text{Case 3)} \quad s_1 = \sigma(Q), \quad c_1 = 0, \quad s_1^* = 0, \quad c_1^* = 1$$

So that  $s_1^* = 0$ ,  $c_1^* = 1$  hold in all five cases.

The values of  $s_1$  and  $c_1$  are:

	1)	2)	3)	4)	5)
$s_1 =$	$Q/R_1$	$N/R_2$	$\sigma(Q)$	$\sigma(Q)$	$\sigma(N)$
$c_1 =$	$M/R_1$	$P/R_2$	0	0	0

We have seen that the values and P. 51  
 signs of  $s_i, c_i, s_i^*, c_i^*$  are given by formulas  
 (47) and the table (51), if — which is the gene-  
 ral case —  $|MN| \geq |PQ|$  or  $|MN| = |PQ| \neq 0$ .

In the singular case  $MN = PQ = 0$  they are  
 given by the table (52).

The elements of  $C$  now are computed using  
 the following formulae (see formulae (32)):

$$\left. \begin{aligned} c_{jk} &= \alpha = a_{jk} \cdot c_i + a_{jm} s_i \\ c_{jm} &= \beta = -a_{jk} s_i + a_{jm} c_i \\ c_{kj} &= \alpha^* = a_{kj} c_i^* + a_{mj} s_i^* \\ c_{mj} &= \beta^* = -a_{kj} s_i^* + a_{mj} c_i^* \end{aligned} \right\} (32')$$

as well as

$$c_{kk} = R_1 + R_2 = \sqrt{M^2 + Q^2} + \sqrt{N^2 + P^2}$$

$$c_{mm} = R_1 - R_2 = \sqrt{M^2 + Q^2} - \sqrt{N^2 + P^2}$$

The last expressions are justified, observing  
 that (29) become  $c_{kk} = G$ ,  $c_{mm} = H$

with  $G = M \cos d + P \cos s + N \sin s + Q \sin d = T + V = R_1 + R_2$

$H = M \cos d - P \cos s - N \sin s + Q \sin d = T - V = R_1 - R_2$

Since  $M \cos d + Q \sin d = R_1$ ,  $P \cos s + N \sin s = R_2$ .

### §13. Real symmetric matrices

P. 52

If a real matrix is symmetric, then the symmetry is preserved because it can be considered as a particular case of a complex symmetric matrix. Since for all pairs of off-diagonal elements  $a_{km} - a_{mk} = 2Q = 0$ , we have  $\tan d = \frac{Q}{M} = 0$  and therefore can take  $d = 0$ ,  $\xi = \theta$ . The angle  $2\theta = s$  is then obtained from  $\tan s = \frac{N}{P}$ :

$$\frac{\sin 2\theta}{N} = \frac{\cos 2\theta}{P} = \frac{1}{R_2} \quad (R_2^2 = N^2 + P^2)$$

$$\text{and } \delta = \cos 2\theta = \frac{P}{R_2}.$$

Taking  $|\theta| < \frac{\pi}{2}$  so that  $\sigma(c_1) = \sigma(\cos \theta) = +1$ , we have  $\sigma(s_1) = \sigma(\sin \theta) = \sigma(\sin 2\theta) = \sigma(N)$  and thus

$$c_1^* = c_1 = \frac{1}{\sqrt{2}}(1 + \delta)^{\frac{1}{2}}$$

$$s_1^* = s_1 = \frac{1}{\sqrt{2}}(1 - \delta)^{\frac{1}{2}} \sigma(N) \quad (N \neq 0)$$

Therefore,

$$c_{kk} = |M| + \sqrt{N^2 + P^2}$$

$$c_{mm} = |M| - \sqrt{N^2 + P^2}$$

$$c_{jk} = c_{kj} = a_{jk} c_1 + a_{jm} s_1$$

$$c_{jm} = c_{mj} = -a_{jk} s_1 + a_{jm} c_1$$

The diagonalization yields the characteristic roots of the original matrix since  $u'(\xi) = u'(\theta)$ .

We saw that in general the skew symmetry is destroyed by our transformation. But if the original skew symmetric matrix is hermitian, then its skew symmetry is preserved.

If  $a_{kk}$  and  $a_{mm}$  are purely imaginary, then  $M = P = 0$  and  $a_{kk} + a_{mm} = 2im$ ,  $a_{kk} - a_{mm} = 2ip$  give  $a_{kk} = i(m+p)$ ,  $a_{mm} = i(m-p)$ . The skew symmetry further means, if the matrix is hermitian, that  $a_{km} = Q + in$  and  $a_{mk} = -Q + in$  so that  $N = g = 0$ . No transformation is needed if  $Q^2 + n^2 = 0$ . We have therefore  $Q^2 + n^2 > 0$  by hypothesis. To check that the skew symmetry is preserved, we observe that  $H = G = 0$  so that  $c'_{kk} = c'_{mm} = 0$  which means that the diagonal elements of the transformed matrix  $C$  are purely imaginary as they should be. We have, indeed,  $\zeta = z$  since the vanishing of  $c_{mm}$  entails that of  $c_{mk}$ . Therefore,  $\theta = \xi$ ,  $\varphi = \eta$ ,  $c_1^* = c_1$ ,  $s_1^* = s_1$ ,  $c_2^* = c_2$ ,  $s_2^* = s_2$ . Computing  $\alpha^*$ ,  $\beta^*$ ,  $\lambda^*$ ,  $\mu^*$  in (31) we find that

$$\alpha^* = -\alpha, \beta^* = -\beta, \lambda^* = \lambda, \mu^* = \mu \quad \text{P.54}$$

So that (32) yields without difficulty

$c_{kj} = -\bar{c}_{jk}$  and  $c_{mj} = -\bar{c}_{jm}$  and this completes the proof of the fact that  $C$  is also skew symmetric hermitian, if  $A$  is.

A skew symmetric hermitian matrix is normal. Therefore, diagonalizing it with the aid of right unitary factor  $u(z)$  and left unitary factor  $u'(\bar{z}) = u'(\bar{z}) = u^{-1}(z)$  preserves its characteristic roots. Thus, our procedure yields also the characteristic roots (eigen-values) of  $A$ .

Forming the equation in  $t = \tan z$ , we obtain the equation (36)

$$(36) \quad (Q + in)t^2 + 2ipt + (Q - in) = 0$$

so that we are in the exceptional case IV, with its ramifications V and VI (see §8).

All formulas established for these cases IV, V and VI are valid and for  $c_{kk} = ic''_{kk}$  and

$c_{mm} = ic''_{mm}$  we obtain from formulas (39)

$$c''_{kk} = m + R, \quad c''_{mm} = m - R \quad \text{with } R = \sqrt{Q^2 + p^2 + n^2},$$

The other elements of  $C$  being given by P. 55  
the formulas (32).

### §15. Hermitian Matrix.

The matrix  $A = (a_{ij})$  is hermitian when  $a_{mn} = \bar{a}_{nm}$ . Therefore, for a hermitian matrix  $Q = m = n = p = 0$ ,  $\frac{1}{2}(a_{kk} + a_{mm}) = M$ ,  $\frac{1}{2}(a_{kk} - a_{mm}) = P$ ,  $\frac{1}{2}(a_{km} + a_{mk}) = N$  and  $\frac{1}{2}(a_{km} - a_{mk}) = iq$ .

We take again  $\zeta = z$ , so that  $\theta = \xi$  and  $\varrho = \eta$  and  $c_2 = c_2^*$ ,  $s_2 = s_2^*$ ,  $c_1 = c_1$ ,  $s_1 = s_1$ ,  $\delta = \delta^*$ ,  $\gamma = \gamma^*$ .

The characteristic roots are preserved, so that diagonalizing a hermitian matrix we find its characteristic roots.

To prove that our transformation does not destroy the hermitian character of a matrix, we observe that this time  $\alpha^* = \alpha$ ,  $\beta^* = \beta$ ,  $\lambda^* = -\lambda$  and  $\mu^* = -\mu$ , so that the formulae (32) give  $c_{kj} = \bar{c}_{jk}$  and  $c_{mj} = \bar{c}_{jm}$ . Moreover,  $g = h = 0$  and therefore the diagonal elements of  $C$  are real numbers as they should be in a hermitian matrix.

The equation (18) for  $t = \tan z$  is as follows P. 56

$$(N + iq)t^2 + 2Pt - (N - iq) = 0$$

Since, multiplied by  $-i$ , it becomes of the same type as (36), namely

$$(q - iN)t^2 - 2iPt + (q + iN) = 0$$

we obtain the solution in the same form:

$$\tan 2\theta = \frac{N}{P}$$

$$\operatorname{ch} 2\varphi = -\frac{q}{R}$$

$$\sin 2\theta = \frac{N}{\omega}$$

$$\operatorname{sh} 2\varphi = -\frac{q}{\omega}$$

$$\delta = \cos 2\theta = \frac{P}{\omega}$$

$$r = \operatorname{ch} 2\varphi = \frac{R}{\omega},$$

where  $\omega = \sqrt{N^2 + P^2}$  and  $R = \sqrt{N^2 + P^2 + q^2}$ . This gives

$$c_1^* = c_1 = \left[\frac{1+\delta}{2}\right]^{\frac{1}{2}}; \quad c_2^* = c_2 = \left[\frac{r+1}{2r}\right]^{\frac{1}{2}}$$

$$s_1 = s_1^* = \sigma(N) \left[\frac{1-\delta}{2}\right]^{\frac{1}{2}}; \quad s_2^* = s_2 = -\sigma(q) \left[\frac{r-1}{2r}\right]^{\frac{1}{2}}$$

This solution presupposes  $N \neq 0$ , so that  $\omega \neq 0$  and  $\sigma(N)$  has a meaning. Insofar as  $\sigma(q)$  is concerned, if  $q$  vanishes  $\sigma(q)$  is no more needed because together with  $q$  also  $p$  vanishes so that  $s_2 = 0$ , if  $q = 0$ .



Equations (28<sup>63</sup>) give very simple expres- P. 57  
 sions for  $c_{kk}$  and  $c_{mm}$ , namely because in our  
 case  $Q=0$ ,  $T=M$  and  $V = N \sin s + P \cos s = \omega$ :

$$(53) \begin{cases} c_{kk} = \gamma V + T = M + \gamma \omega = M + R \\ c_{mm} = T - \gamma V = M - \gamma \omega = M - R \end{cases}$$

It may happen that  $N=0$ . In this par-  
 ticular case  $q$  cannot vanish since  $N^2 + q^2 =$   
 $= |a_{km}|^2 = |a_{mn}|^2 \neq 0$ . For  $N=0$ ,  $q \neq 0$  the

equation (18) becomes

$$t^2 - \frac{2iP}{q} t + 1 = 0$$

So that

$$t = -i\delta(q) \left\{ \frac{R-P}{R+P} \right\}^{\frac{1}{2}}$$

which can be written also as follows:

$$(R+P) \left[ \text{th}\varphi + \delta(q) \sqrt{\frac{R-P}{R+P}} \right] \cos\theta + i \left[ \text{th}\varphi + \delta(q) \sqrt{\frac{R+P}{R-P}} \right] \delta(q) \sin\theta \frac{(R-P)}{R+P} = 0$$

Since  $|\text{th}\varphi| \leq 1$ , we have, as in the case  
 V, to separate the subcases  $P \geq 0$ . If  $P$   
 is positive, then  $\text{th}\varphi = -\delta(q) \sqrt{\frac{R-P}{R+P}}$  and  $\sin\theta = 0$

while  $\text{th}\varphi = -\delta(q) \sqrt{\frac{R+P}{R-P}}$  and  $\cos\theta = 0$ , if  $P$   
 is negative. Taking  $\theta = 0$  or  $\theta = \frac{\pi}{2}$

respectively and, observing that  $\gamma = R/|P|$  for  $P \neq 0$  and  $\gamma = \infty$  for  $P = 0$ , we obtain for  $P \geq 0$ : P.58

$$s_2^* = s_2 = -\sigma(q) \left[ \frac{1}{2} \left( 1 - \frac{|P|}{R} \right) \right]^{\frac{1}{2}} ; c_2^* = c_2 = \left[ \frac{1}{2} \left( 1 + \frac{|P|}{R} \right) \right]^{\frac{1}{2}}$$

If  $P$  vanishes,  $\theta$  becomes indetermined since the equation for  $t$  gives  $t^2 + 1 = 0$ , so that  $\theta = -\sigma(q)$ . We take  $\theta = \frac{\pi}{4}$  if  $P = 0$  and thus, if  $P \neq 0$ :

$$s_1^* = s_1 = \frac{1}{2} [1 - \sigma(P)]$$

$$c_1^* = c_1 = \frac{1}{2} [1 + \sigma(P)] , \text{ while } c_1 = s_1 = \frac{\sqrt{2}}{2} , \text{ if } P = 0.$$

Formulae (53) hold for  $N = 0$ , whatever is the value of  $P \geq 0$ . If  $P$  vanishes they become

$$c_{kk} = M + |q| ; c_{mm} = M - |q|$$

Since for  $P = 0, N = 0$  we have  $R = |q|$ .

The off-diagonal elements of the transformed matrix  $C$  are given, as always, by formulae (32) in which for  $P \neq 0$

$$2(\alpha + i\lambda) = a_{jk} + a_{jm} + (a_{jk} - a_{jm}) \cdot \sigma(P)$$

$$2(\beta + i\mu) = -a_{jk} + a_{jm} + (a_{jk} + a_{jm}) \cdot \sigma(P)$$

while for  $P = 0$

$$(\alpha + i\lambda)\sqrt{2} = a_{jk} + a_{jm} \text{ and } (\beta + i\mu)\sqrt{2} = a_{jm} - a_{jk}$$

$$\text{Thus, } c'_{kj} = c'_{jk} = \alpha c_2 - \lambda s_2 ; c''_{jk} = \lambda c_2 + \beta s_2 = -c''_{kj}$$

$$c'_{mj} = c'_{jm} = \beta c_2 - \lambda s_2 ; c''_{jm} = \lambda c_2 + \alpha s_2 = -c''_{mj}$$

February 15, 1954\*

E. Kogbetliantz

Thus,  $c_{kj}$ ,  $c_{mj}$ ,  $c_{jk}$ ,  $c_{jm}$  are given by:

$$c'_{kj} = c'_{jk} = \alpha c_2 - \mu s_2 ; c''_{jk} = -c''_{kj} = \lambda c_2 + \beta s_2$$

$$c'_{mj} = c'_{jm} = \beta c_2 - \lambda s_2 ; c''_{jm} = -c''_{mj} = \mu c_2 + \alpha s_2.$$

This terminates the study of particular matrices.

### § 8. (Continued) . Case VII).

In the singular case VII)  $\omega = \omega^* = E = E^* = 0$  as well as  $n = Q = 0$ , but  $N^2 + q^2 > 0$ . The definitions of  $\omega^2 = (D+A)^2 + (B-C)^2$ ,  $\omega^{*2} = (D-A)^2 + (B+C)^2 = 0$  show that in this case  $A=B=C=D=0$ . Therefore, the formulae (20) p. 12 prove that the case VII) takes place only when the four elements  $a_{kk}$ ,  $a_{km}$ ,  $a_{mk}$ ,  $a_{mm}$  verify the following five con-

ditions:  $pq=0$ ,  $qM=0$ ,  $PM+pm=0$ ,  $MN=0$ ,  $pN=0$ .  
(54)

It is easy to prove that (54) implies  $p=M=0$ . First, if  $N \neq 0$  the last two conditions give  $p=M=0$ . Let now  $N=0$ ,  $q \neq 0$ . Then first two conditions give the same result. Therefore, in the case VII) we have always  $p=M=0$ .

$$\begin{aligned}
 a_{kk} &= P + iz & a_{km} &= N + iq \\
 -a_{mm} &= P - iz & a_{mk} &= N - iq \quad (N^2 + g^2 > 0)
 \end{aligned}$$

Eq. (13) yield now only one equation, namely

$$P \sin(z+\zeta) - N \cos(z+\zeta) = i [z \sin(z-\zeta) - g \cos(z-\zeta)]$$

Therefore, two conditions are to be chosen at will. We choose  $\theta = \varphi$ ,  $\xi = \eta$  so that  $\zeta = z$  and the equation becomes

$$\boxed{P \sin 2z - N \cos 2z = -ig} \quad (55)$$

We observe that our choice of  $z = \zeta$  presupposes that  $P^2 + N^2 > 0$  since  $g$  cannot vanish, if  $N = 0$ .

Therefore, we will have to study the case  $N = P = 0$  separately. If  $P^2 + N^2 > 0$ , then (55) gives:

$$P \sin 2\theta \operatorname{Ch} 2\varphi - N \cos 2\theta \operatorname{Ch} 2\varphi = 0$$

$$P \cos 2\theta \operatorname{Sh} 2\varphi + N \sin 2\theta \operatorname{Sh} 2\varphi = -g.$$

The first equation gives

$$\sin 2\theta = N/\omega \quad ; \quad \cos 2\theta = P/\omega$$

with  $\omega^2 = P^2 + N^2 > 0$ . Then, from the second one:

$$\operatorname{Sh} 2\varphi = -g/\omega$$

Denoting now  $\omega^2 + g^2 = P^2 + N^2 + g^2$  by  $R^2 = \omega^2 + g^2$ , we have:

$$\gamma = \gamma^* = \operatorname{Ch} 2\varphi = R/\omega \quad ; \quad \delta = \delta^* = \cos 2\theta = P/\omega$$

Since  $\sigma(\varphi) = -\sigma(\psi)$  and  $\sigma(\theta) = \sigma(N)$ ,

we obtain

$$c_1^* = c_1 = [\frac{1}{2}(1+\delta)]^{\frac{1}{2}} ; s_1^* = s_1 = \sigma(N) \cdot [\frac{1}{2}(1-\delta)]^{\frac{1}{2}}$$

$$c_2^* = c_2 = [\frac{1}{2}(1+\gamma^{-1})]^{\frac{1}{2}} ; s_2^* = s_2 = -\sigma(\psi) [\frac{1}{2}(1-\gamma^{-1})]^{\frac{1}{2}}$$

If  $g=0$  then  $\sigma(\psi)$  is not needed since  $s_2$  vanishes for  $g=0$ . Now the formulae (29) yield

$$C_{kk} = R + iz ; C_{mm} = -R + iz$$

since  $G = -H = \omega$ ,  $g = h = z$  so that

$$C'_{kk} = -C'_{mm} = R \text{ and } C''_{kk} = C''_{mm} = z.$$

If  $N=P=0$  then six numbers vanish:

$$M = p = n = Q = N = P = 0$$

and therefore  $q_{kk} = q_{mm} = iz$  and  $q_{km} = -q_{mk} = iq$ ,

$q \neq 0$ . The conditions  $c_{km} = c_{mk} = 0$  are now

$$z \cdot \sin(z-\zeta) = g \cdot \cos(z-\zeta)$$

and we choose two supplementary conditions as follows:

$$\varrho = \eta \text{ and } \xi = -\theta, \text{ that is}$$

$$\zeta = -\bar{z}.$$

With these supplementary conditions P. 62

$$z \sin 2\theta = q \cos 2\theta$$

and therefore the conditions  $c_{km} = c_{mk} = 0$  determine only  $\theta$ , the common value of  $\varphi = \eta$  being arbitrary. We choose  $\varphi = \eta = 0$ , while  $\delta = \cos 2\theta = z \cdot [z^2 + q^2]^{-\frac{1}{2}}$  and  $\sigma(\sin \theta) = \sigma(q)$ .

Therefore

$$c_j^* = c_j = \left[ \frac{1}{2} (1 + \delta) \right]^{\frac{1}{2}}; \quad s_j^* = s_j = \sigma(q) \cdot \left[ \frac{1}{2} (1 - \delta) \right]^{\frac{1}{2}}$$

The elements of  $C$  are:

$$c_{kj} = \alpha^* + i\lambda^*$$

$$c_{jk} = \alpha + i\lambda$$

$$c_{mj} = \beta^* + i\mu^*$$

$$c_{jm} = \beta + i\mu$$

where the right-hand members are defined in 31) p. 21. Moreover

$$c_{kk} = c_{mm} = i \cdot [z^2 + q^2]^{\frac{1}{2}}$$

S. Kogbetliantz

March 15, 1954.

APPLICATIONS OF A CATHODE RAY TUBE READ-  
OUT DEVICE FOR THE IBM 701 ELECTRONIC  
DATA PROCESSING MACHINE

Paul Armer

P-509

8 April 1954

---

The **RAND** Corporation  
1700 MAIN ST. • SANTA MONICA • CALIFORNIA

SUMMARY

A Cathode Ray Tube Readout Device for the 701 is described from the user's viewpoint. Its application to a particular problem is discussed. This problem involves a large amount of output and is unusual in that it is the inverse of the data reduction problem. More general uses are also touched upon.



APPLICATIONS OF A CATHODE RAY TUBE READOUT DEVICE  
FOR THE IBM 701 ELECTRONIC DATA PROCESSING MACHINE

A good deal has been written and a great deal more has been said about the problem of data reduction. By data reduction I refer to the process where instruments are read, the readings are recorded, and the recorded information is then processed to yield a history of what took place. Some of the problems involved are instrumentation, data recording and data processing. The instrumentation problem is frequently very difficult and the data recording often very tedious. The measurements are usually made in an analog fashion so an analog-to-digital conversion is required in order to process the measurements on a digital computer. The converter may be built right into the instrument, as it is in a digital voltmeter, so that the data recorded is digital in nature rather than analog, or the data may be recorded in an analog way as it is on an oscillograph trace.

All of this is said in the way of introduction to a problem which I feel will be, at least, of academic interest to those concerned with data reduction. The solution to the problem involves some hardware which I feel will be of interest to most of you. That facet of the problem which makes it unusual is that it is essentially the inverse of the data reduction problem in that we want to end up producing an instrument reading rather than starting with one. Instead

of an analog-to-digital conversion, we have a digital-to-analog conversion to make; the computing comes first, not last.

The specific problem with which we were faced was that of training a radar operator. For the sake of brevity I will simplify the problem somewhat. One way to attack the problem is to dream up a synthetic track of an aircraft where the coordinates in space are a function of time and then somehow get the track onto the scope seen by the radar operator. Let's take a short look at the computational problem involved. Suppose one wants to present six tracks to the operator as a training problem which is to last one hour. Since the presentation changes with each rotation of the radar antenna, we will need to know just where each aircraft is for each rotation. If the antenna in our training problem makes one rotation per minute, we'll need positions for sixty rotations for six aircraft, yielding 360 sets of information for our problem. From a computational standpoint, we can completely specify a given track by assuming that every track is made up of straight line segments and giving the coordinates and time of the end points.

This "turning point" information for each track is fed into the 701, which then computes, by linear interpolation, the coordinates of the aircraft for each rotation of the radar antenna. Whether or not the radar actually "sees" the

aircraft at a specific point in space is a function of the local topography. The 701 is programmed to make the decision as to whether or not the aircraft does appear on the scope at each point along its track. Since the see-no see decision is a function of the z coordinate, this coordinate must be computed even though only x and y are displayed.

To those of you familiar with radar, it will be obvious from the above that I am talking about a PPI scope - the plan position indicator. Such a scope presents only a plan picture of the area around the antenna. No height information is presented.

When the coordinates for each aircraft have been computed for each antenna rotation, it is necessary to order this information with respect to time. Normally, all the points for a given track are computed before going onto the next track. Since the presentation is chronological, a sorting operation (or merging if you prefer) is required to give a time ordering rather than an ordering by track. Once this is accomplished, it is necessary to get the information out of the computer and eventually onto the training scope.

Just how we were to output this information and store it prior to its presentation on a training scope was the big problem. Techniques existed for getting information from film to the cathode ray tube (CRT) of a radar so film appeared to be a desirable medium for recording and storing the infor-

mation to be displayed.

At this point, the only hurdle remaining was to get the information on the film. The obvious answer was to photograph a CRT with the coordinate information displayed on it. CRT's had been used as output devices for computers by other people, notably MIT and the University of Illinois. Now all that was required was to get the hardware built. We went to IBM with our problem and they undertook to construct a CRT readout device for us. It was installed on our 701 early in April.

Before I start to give some of the details of this unit I would like to explain that because of our particular application, it was necessary to get as much accuracy out of the system as the state of the art would allow. This accuracy is probably greater than most of you would require for your own purposes.

The unit actually has two CRT's; one, a seven inch tube in what is called the "recording unit", is the tube which is photographed. The other tube is for visual observation and is a whopping 21 incher. This is called the "display unit".

The recording unit works something like this. Information for the point to be plotted must be in electrostatic storage. Each of the coordinates is specified by ten binary digits. The x coordinate normally appears in the left address and the y coordinate in the right address although pro-

vision is made, under control of a switch, for reading them from the rightmost 20 bits of the word. When a copy order is given, after the CRT has been selected, this information is passed via the Multiplier-Quotient Register (MQ) to two buffer storage units called the deflection registers. The digital information in these registers goes through two digital-to-analog converters to become the appropriate deflection voltages for the CRT and then the beam is turned on. All this makes it sound quite simple while in reality it wasn't, due, principally, to the accuracy we were after. For example, normal CRT tubes aren't very orthogonal nor very linear. To satisfy our needs for better tubes, we had DuMont build some special ones for us.

Since ten bits go into each coordinate, we have a 1024x1024 raster. This raster is about three inches on a side on the seven inch tube. The spot diameter is about .012 inches and consequently is about four times as large as the spacing between points on the raster. This may seem a little strange but we hope to reduce the spot size at which time the tenth bit in the deflection information will become more meaningful. Also with this extra bit it is possible to draw/<sup>a</sup>continuous smooth curve by calling for the displaying of adjacent points.

I might make some general (I purposely emphasize the word general) statements about accuracy. The short term accuracy of the recording unit is of the order of .1% of full

scale. Over a period of hours, it is conceivable that the accuracy may deteriorate to as much as  $\frac{1}{2}\%$  but to date has shown very little tendency to actually do so. Those of you familiar with analog equipment will understand the reasons for this - for this equipment is an analog device. The display unit should be good to about 3% of full scale. The greater error in the display unit is due to the characteristics of the larger tube.

Several sense instructions have been added in order to shorten the loop required for displaying a number of points. With these sense instructions the total time for displaying one point is approximately 400 microseconds. This time includes the time taken to carry out all the orders in the loop. The actual time consumed in the displaying of a single spot is about 200 microseconds. Persistence of the spots on the recording tube is of the order of a few microseconds, and unless regenerated, can scarcely be detected by the eye. Spots on the display tube will persist for about 20 seconds. With this long persistence, it isn't necessary to worry, if you have a good many points to display, about the first point plotted dying out before the last one is displayed. At 400~~us~~ per point, some 50,000 points could be displayed in 20 seconds.

The movie camera, which photographs the recording unit CRT, is not under the control of the computer. The speed of the camera is variable up to twenty frames per second and, if

set for that speed, will advance one frame every 50 ms no matter what the 701 is doing. In order to synchronize the camera with the displaying of points by the 701, the status of various features of the camera is reported to toggles in the 701. The state of these toggles can then be sensed by sense instructions. Five sense instructions have been added for use with the CRT. Three of these are associated with the camera and the CRT, two of them are internal to the 701. The two which are internal to the 701 are of general interest. One looks at the  $2^{-1}$  position of the MQ and the 701 skips an instruction if it finds a "1" present. The other is identical to the first except that it looks at the  $2^{-2}$  position in the MQ.

I'd like to discuss some other uses we intend to make of the CRT unit. Since it is logically the same as a high speed point plotter connected directly to the computer, it has the obvious applications of a point plotter. In addition to the plotting of results, routines can be devised to display alphanumeric characters. However, as long as all the points for each character must be generated within the machine, the output speed is rather slow, being of the order of 50 characters per second. This is about one-sixth the speed of a 407 printer.

A good many computing problems are of such a nature that there is a man in the feedback loop; that is, results for one run are obtained on the computer, these are analyzed by the

human, the problem is then changed, possibly only to the extent that new parameter values are chosen, and then the problem goes back on the machine. The problem goes around this loop several times before it is completed. For many of these problems, the economics of the situation require that the analysis of the results be carried out apart from the machine so that expensive machine time is not wasted.

However, I believe that for some of these problems, it is economically feasible to have the human "think on his feet" at the machine. We have been doing problems in this fashion on our analog equipment for a good many years. Calculus of variation problems are an example of this kind of problem.

To do this well, two things are required. First, it must be possible to present the results rapidly, particularly if the output is more than a very few numbers, and to present them in such a fashion that they may be rapidly assimilated by the human being in the loop. Secondly, there must be an easy way for the individual to communicate his decisions back to the computer.

I believe the CRT fills the first requirement pretty well but the second has yet to be met satisfactorily for the 701. The need for such a facility for the CPC gave rise to such gadgets as the Gizmo of the University of Wisconsin and other such "parameter boards". The situation is different



here, however. For the CPC, one was usually setting in a predetermined pattern - it was just much easier to use the parameter board than to feed in cards with the appropriate information. The problem was to get the cards in the appropriate places in the program deck. On the 701, the same thing can easily be accomplished by code. However, if the numbers to be entered are not predetermined, it's not so easy.

It is possible to get these non-predetermined numbers into the machine via the console or via cards keypunched on the spot. But neither of these methods works out very well. Mr. Eugene Jacobs of RAND came up with an improvement over the above schemes which uses the card reader. Impulses from the digit emitter on the reader are taken to a rotary switch and then back to the Calculate Entry hubs. One switch is required for each digit to be entered. Through the use of selectors, such a board can still be used to read cards normally. When entering numbers in this fashion, blank cards with an appropriate x punch are used.

Nat Rochester of IBM has suggested yet another, though similar, scheme which uses the echo impulses from the printer back to the computer to get information from a Gizmo-like setup into the machine. However, neither of these methods quite satisfies me for I've been spoiled by all the potentiometers (pots) I can set on our analog computer. I'd like to have pots, either analog or digital, to set on the 701.

I'd like to go even a little further with this pot idea and ask that these pot settings be addressable in the same way as any memory position. If I had several such addressable pots, then I'd also like some additional analog outputs such as meters and maybe even another CRT display unit.

SORTING ON THE 701

by

D. T. Blum and P. Fagg

With contributions by:

L. B. Stinnett

H. D. Avram

N. D. Belnap

T. E. McCool

29-1013-0(BF)

This paper attempts to describe the effort of our 701 programming group on sorting.

It is well to state, in the beginning of this paper, that there was a great deal of skepticism displayed by those involved in initial planning of programs for our 701 concerning the feasibility of utilizing a "mathematically conceived" electronic computer for data-handling problems; one of these problems was sorting. However, it was felt that it was important to program a sort of some kind, exploiting all possible time-saving mechanisms in programming. Unfamiliarity with 701 programming, inexperienced personnel, and lack of development time hampered a truly thorough search of the problem, and we are still involved in that program now, over a year since our machine was delivered.

Much effort is yet to be expended in the future. We have adapted programs of other people, have written original programs, and have varied our operational procedure according to the job on hand. We have found, as yet, no set rule for application to all jobs which require sorting. However, we feel that we have proven the feasibility of sorting on the 701, and are processing data utilizing sorting programs at the present time.

## THE BIT SORT

The "Bit Sort" is a 701 program designed to sort information in binary form bit by bit. It has the advantage of allowing sorting on any particular number of bits, rather than on a specified number of half or full words.

In theory, the bit sort is actually a special case of multi-bit or address sort. It performs on the 701 what a non-existent two pocket sorter could perform on cards punched with data in the binary system (ie., only 1's or 0's). A description of the bit sort in terms of the two pocket sorter, with the analogous 701 terms in parenthesis, follows below.

First, the data cards are loaded into the hopper of the two pocket sorter (the binary data is loaded onto magnetic tapes). Then, starting at some specified column (bit location), all cards (full words) with a 0 in that location are routed to one pocket (tape), and those with a 1 are routed to another pocket (tape). At the end of one run (pass), the cards are reloaded in the sorter hopper so as to preserve the effects of the previous passes (the tapes are read backward in the proper order) and the process is repeated until the data has been sorted on desired columns (bits).

An actual example showing the steps involved in this particular bit sort is given later.

The particular bit sort in use was written by T. E. McCool, and has the following characteristics:

1. Without any manual changing of the tapes, it will sort a maximum of one full tape of data, and the original data may be on either tape 400 or 401 or both as specified by positive constants in two locations.
2. The initial data tape can consist of any number of unit records of 200 full words or less, but always with an even number of full words per unit record. Each unit record is comprised of "sorting units" of two full words, the first containing indicative information, and the second full word containing the data to be sorted. The number of sorting units in different unit records can be variable so long as it does not exceed 100. A similar program could be written also allowing flexible sorting unit format, but we do not have one available.
3. At present it is set to sort on all 35 bits of the full word, although with minor modifications or a calling sequence it could be adapted to sort any block of bits within the full word.

The actual program is written in symbolic, and it has been assembled to use the following locations:

	Decimal	Octal
Program Storage	300-632	0454-1170
"0" Storage blocks	800-1199	1440-2257
	1200-1599	2260-3076
"1" storage blocks	1600-1999	3100-3717
	2000-2399	3720-4537

Total storage 1933 decimal locations.

A step by step example of the use of the bit sort on binary information follows:

The tapes with the initial data (tapes 400 or 401 or both), must be positioned at the end of file before the bit sort can begin, as the program only reads backwards.

Assume a file of 7 unit records on tape 400, which are to be sorted successively on bits 1, 2, 3, 4, and 5 from the right of the full word. Instead of showing the full sorting units, consisting of one full word of indicative followed by a full word of sorting material only the bits actually being sorted are shown. In sorting, these units are preserved as complete entities.

INITIAL DATA ON TAPE 400

<u>bits</u>	54321	
	00100	Beginning of file
	11111	
	11011	
	10111	
	00111	
	10100	
	10010	End of file

In the first pass, tape 400 is read backwards from the end of file condition, a full unit record at a time (note that in any pass, an entire unit record, not a sorting unit, is either written or read at one time). The data in each unit record is examined on bit 1, and all the sorting units with a "0" in bit 1 are stored in the 0 storage block in electrostatic memory, while the "1" data is stored in the 1 storage block. The contents of these storage blocks in electrostatic memory are written only on the appropriate tapes after 200 sorting units have been accumulated, or at the end of the sorting pass. The first pass takes the original data, which may have been in variable length unit records, and puts it into unit records containing 200 sorting units, except for the last unit record, which may be short.

For any odd pass, data is read backwards from tape 400 and then 401, the data with a zero in the bit being sorted, is written on tape 402, and the data with a one is written in tape 403.

The results of the first pass are given below:

<u>"0" Tape (402)</u>	<u>"1" Tape (403)</u>	
1	1	Beginning of File
10010	00111	
10100	10111	
00100	11011	
	11111	End of file

### SECOND PASS

On the second, and all even passes, the "1" tape from the preceding pass (403) is read backward before the "0" tape (402). This sequence is necessary to preserve the effects of previous sorting passes. The new "0" data is written on tape 400, and the "1" data on tape 401, as indicated below:

At the end of each even pass, the data is in descending sequence according to the sorted bits when reading forward from tape 401 and then tape 400.

<u>"0" Tape (400)</u>	<u>"1" Tape (401)</u>
2	2
00100	11111
10100	11011
	10111
	00111
	10010

### THIRD PASS

In the third pass, and all odd passes after the first, the "0" tape (400) is read backward first, the "1" tape (401) is read backward next, the new "0" data is written on tape 402, and the new "1" data is written on 403. This reading sequence preserves the effect of the previous passes.

At the end of each odd pass, the data is in ascending sequence according to the sorted bits when reading forward from tapes 402 and then 403.

<u>"0" Tape (402)</u>	<u>"1" Tape (403)</u>
3	3
10010	10100
11011	00100
	00111
	10111
	11111

The results for the 4th and 5th passes are shown below:

#### FOURTH PASS

<u>"0" Tape (400)</u>	<u>"1" Tape (401)</u>
4	4
10111	11111
00111	11011
00100	
10100	
10010	

#### FIFTH PASS

<u>"0" Tape (402)</u>	<u>"1" Tape (403)</u>
5	5
00100	10010
00111	10100
	10111
	11011
	11111

For each pass after the first, the basic rules and the results for each pass are outlined below:

#### ODD PASS

1. Read back "0" tape (400), then "1" tape (401)
2. Write resulting "0" data on tape 402, "1" data on tape 403
3. The results at the end of this pass will be in ascending sequence when reading forward from tape 402 and then 403.

#### EVEN PASS

1. Read back "1" tape (403) then "0" tape (402).
2. Write resulting "0" data on tape 400, "1" data on tape 401.
3. The results at the end of this pass will be in descending sequence when reading forward from tape 401 and then 403.

Note that data with zero in the bit being sorted is always written on either tape 400 or 402 and data with a one in the corresponding bit is always written on either tape 401 or 403.

The final status of the data at the end of the sort depends only on whether there have been an even or an odd number of bits sorted on.

#### TIMING

A full tape, or about 1100 unit records of information, can be sorted on a full word (ie., 35 passes), in about 3 hours, or about 5 minutes/bit/full tape. This is equivalent to roughly 0.3 sec/unit-record/bit. For sizeable quantities of data, the total time is about twice the reading time required, remembering that in effect only one bit is read during each pass. On the average, for each unit record that is read from the tape, one unit record is written on either the "1" or "0" tape. Actually the zero or one data is written whenever the storage block (which is equivalent in size to a full unit record) becomes filled.

Added to the double reading time required is the time consumed in Writing End of File, in Reading Back from the End of File condition, in Rewinding past the Beginning of File, and in Writing the Beginning of File. This time is completely independent of the number



of unit records, and totals about 3.6 seconds/bit, or 2 minutes/full word.

Expressed mathematically, the total sorting time may be expressed by the equation:

$$\text{Total time (minutes)} = \text{number bits being sorted} (.06 + \text{number of 200 word-unit records} \times .005)$$

Note that by dealing with unit records usually containing 100 words to be sorted, rather than having a separate unit record for each sorted word, the sorting time is reduced considerably, the strain on the tapes is decreased, and the amount of information which can be stored on the tape is increased perhaps eight fold.

## USE

The "bit sort" is exceedingly useful in several operations. It is the only sort we have which is well adapted to block sorting large quantities of data (ie., of breaking it up into blocks, not of sorting individual blocks). It is equally good for sorting information using only a few bits in the sorting word, and it is flexible enough to be used as a normal sort within other programs. One of its main disadvantages has been the operational necessity of starting over when tape trouble occurs. Also, unless the original material is duplicated and saved on another tape, it may be lost. Use of the "Sense Tape" order should reduce this difficulty, but will not eliminate it.

## OPERATIONAL EXPERIENCE

The "bit sort" has been used successfully many times, but it has encountered tape trouble on many occasions for several reasons.

1. Inadequate storage of the tapes, if tape performance can be improved considerably by special storage of the tapes. No special care is given to our tapes other than keeping them in the 701 air conditioned room.
2. The "bit sort" is hard on the tapes, and most trouble within the tape units of the 701 will show up as operational difficulties. For sorting on more than a few bits, the bit sort may read and write on the tapes more than some of the other sorting programs, and therefore may be more subject to tape errors.

The possibility of either losing the original data, or having to restart the sorting operation if machine trouble occurs, must be considered.

## ADDRESS OR MULTI-BIT SORT

The "address sort" is a 701 program designed to sort information stored within the electrostatic memory.

The input is a number of units of data. The address sort does not actually move the data being sorted; rather, its output is a file of addresses with the address of the lowest valued data first, and the address of the highest value data last. It does this by a method akin to that employed by the standard IBM sorter. That is, the addresses of the sorting units are distributed into various pockets (blocks of electrostatic memory), according to the value of groups of bits in successively more significant fields (that is, in minor, then intermediate, then major fields), until the sort is completed.

The method employed in sorting on a particular group of bits is basically as follows: A run is made through all the data in order to count the number of units of data in each category. There will be two categories, where  $n$  is the number of bits being sorted each pass ( $n$  is usually 5, 6, or 7). " $n$ " bits is equivalent to a single character. A "pocket" (block of electrostatic memory) is then allocated or set up for each category, and its size is determined by the number of units of data in this category, as found in the counting described above. It is then necessary to make an additional run through the data in order to put the addresses of the units into the pockets provided for them. The operation is similar to that of the IBM sorter, except that variable pocket sizes are pre-determined by the counting, because it is inconvenient to "empty" a pocket when it gets full, as is done on the sorter.

Consider the following example which indicates the steps involved in address sorting some data. The 701 category will be complete if instead of considering sorting by columns, groups of bits or characters had been sorted each pass.

The data to be sorted by means of its addresses consists of six two digit numbers, and it is known that the value of these digits ranges from 0 to 3. Nine locations of associated material follow each two digit number location. In other words, the sorting block consists of 10 locations, and it is to be sorted on the data in the first location. The addresses or locations are followed by the letter A to distinguish them from the data.

LOCATIONS OF DATA TO BE SORTED	DATA TO BE SORTED	LOCATIONS OF INFORMATION ASSOCIATED WITH THE DATA TO BE SORTED
0 A	21	1 A - 9 A
10 A	10	11 A - 19 A
20 A	30	21 A - 29 A
30 A	12	31 A - 39 A
40 A	01	41 A - 49 A
50 A	23	51 A - 59 A
	Col. 2    ↗    ↖    Col. 1	

## FIRST PASS

Each pass consists of three major steps:

1. Accumulate the frequency of the characters in each category for the column being sorted. (ie., the number of 0's, 1's, and 2's in column 1). This entails one run through the data.

<u>LOCATION OF COUNTER</u>	<u>WHAT IT COUNTS</u>	<u>RESULTS (on col. 1)</u>
100 A	zeros	2
101 A	ones	2
102 A	twos	1
103 A	threes	1

2. COMPUTE THE INITIAL STORAGE ADDRESSES FOR EACH CATEGORY  
(Starting at location equals 200 A)

Stores addresses of data with 0's in col. 1 beginning at loc. 200 A

Stores addresses of data with 1's in col. 1 beginning at loc. 202 A  
(200 A plus number of 0's)

Stores addresses of data with 2's in col. 1 beginning at loc. 204 A  
(200 A plus number of 0's and 1's)

Stores addresses of data with 3's in col. 1 beginning at loc. 205 A  
(200 A plus number of 0's, 1's, and 2's.)

3. SEQUENCE THE ADDRESSES OF THE DATA ACCORDING TO THE SORT ON Col. 1

Examine the first column of the original data directly again and store the address of the data in the appropriate category and location as determined in step 2. After storing an address in any category, increase the storage address for that category 1 location.

<u>LOCATION OF SORTED ADDRESSES STORED BY THE FIRST PASS</u>	<u>ADDRESS STORED (in parentheses is the column the address refers to )</u>
200 A	10 A (address of 0 in col. 1)
201 A	20 A (address of 0 in col. 1)
202 A	0 A (address of 1 in col. 1)
203 A	40 A (address of 1 in col. 1)
204 A	30 A (address of 2 in col. 1)
205 A	50 A (address of 3 in col. 1)

## SECOND AND ALL FOLLOWING PASSES

The major difference between the first and the succeeding passes is that the succeeding passes must preserve the effect of the previous passes. The effect of this shows up in

step 3. Step 1 and step 2 can be done as before, using the run through the unsorted data. The actual sequencing in step 3 must be done through reference to the addresses just sorted and then to the data, rather than referring directly to the unsorted data.

1. COUNT the data in each category as before, except that the sort is now being made on a different column (ie., col. 2 in this example).

<u>LOCATION OF COUNTER</u>	<u>WHAT IT COUNTS</u>	<u>RESULTS (on col. 2)</u>
100 A	zeros	1
101 A	ones	2
102 A	twos	2
103 A	threes	1

Note that the counter locations and what they count are fixed, but the counters themselves are cleared to zero before each counting run.

2. COMPUTE THE INITIAL STORAGE ADDRESSES FOR EACH CATEGORY

There are two blocks of storage for the addresses: The X block (starting at address 200 A in this example), and the Y block (starting at 300 A). These blocks are alternated each pass, so that the Y storage locations are used for all the even numbered passes, and the X storage locations are used for all the odd numbered passes, such as the first one.

Start storing address of data with 0's in col. 2 in loc. 300 A  
 Start storing address of data with 1's in col. 2 in loc. 301 A  
 (300 A plus the number of 0's)  
 Start storing address of data with 2's in col. 2 in loc. 303 A  
 (300 A plus the number of 0's and 1's)  
 Start storing address of data with 3's in col. 2 in loc. 305 A  
 (300 A plus the number of 0's, 1's, and 2's)

3. SEQUENCE THE ADDRESSES OF THE DATA ACCORDING TO THIS SORT AND THE PREVIOUS SORTS (For this example, according to the major sort on col. 2 and the minor sort on col. 1)

Pick up the data that the first address in X or Y (whichever was stored in the last pass) refers to. Examine the data on the column being sorted, and store the address of this data in the appropriate category and location as determined in step 2. After storing an address in any category, increase the storage address for that category 1 location. Pick up the next address sorted during the previous pass, then the data it refers to, and continue in this fashion until the pass is completed. Note the extra step introduced here to preserve the effects of the previous passes. In the first pass:

- a. The original data is referred to.
- b. The addresses of this data are stored in the appropriate locations.

In all other passes:

- a. The addresses sorted in the previous pass are referred to.
- b. The original data to which these addresses refer is examined.
- c. The addresses of this data are stored in the appropriate locations.

<u>LOCATIONS OF SORTED ADDRESSES STORED BY THE FIRST PASS</u>	<u>ADDRESS STORED (in parentheses is the data the address refers to)</u>
300 A	40 A (address of 01)
301 A	10 A (address of 10)
302 A	30 A (address of 12)
303 A	0 A (address of 21)
304 A	50 A (address of 23)
305 A	20 A (address of 30)

At the end of this pass, continue in the same manner as for the 2nd pass until all the columns to be sorted have been completed.

The file of addresses is now in order according to the data it refers to, and an exit is made from the address sort. From this point on, the programmer can either rearrange the actual data into sort within electrostatic memory, store the data in sort on the tapes or drums, work with the sorted addresses, or treat it in some other manner. Programming to use the addresses is not difficult, although some storage problems may arise in actually moving the data within electrostatic memory.

#### PROGRAM CHARACTERISTICS

This "address" or "multi-bit" sort was written by N. Belnap, and must be supplied with the parameters listed below. A sorting unit is defined as one complete unit of information to be sorted together with the associated indicative information. It is comparable to an IBM card, and consists of a specified number of full words. A sorting block consists of a group of sorting units, each with the same format.

1. Number of full words in the sorting unit, including the non-sorted or indicative information.
2. Number of full words within the sorting unit to be actually sorted. (Note that as the program is now written the sorted material must come just within the sorting unit; also the program sorts on groups of bits of full words. Therefore all numbers are sorted on their absolute bit value.)
3. "n", the number of bits to be sorted each pass. 2 counter locations must be set aside in memory. n = 5 through 7 is a reasonable value, although

12 is theoretically possible. Increasing the number of bits being sorted does not necessarily decrease the sorting time proportionally, as more time is required to clear the counters and to set up the initial category addresses.

4. Number of passes to be made per full sorting word.
5. Number of non-sorted bits in the right of each word. The combination of 3, 4, and 5 means that the bits to be sorted on within each full word can be specified, but these bits must be the same for all the full words being sorted within the sorting unit.
6. Number of sorting units per block.
7. Address to transfer to when the program is completed.

In the accompanying listing, the program sorts information in the following form:

1. 4 full words per sorting unit.
2. Sort on 1st three full words
3. Sort 7 bits per pass.
4. 5 passes per full word.
5. No unsorted bits in the right of the full word.
6. 500 sorting units per block.
7. Transfers to symbolic location 50.00.00 at end of program.

## STORAGE

The actual program is written in symbolic, and has been assembled to use the following locations:

	Decimal	Octal
Program Storage	200 - 437	310 - 665
Counter Storage	0 - 127	0 - 177
X storage block	3000 - 3499	5670 - 6653
Y storage block	3500 - 3999	6654 - 7637
Data	1000 - 2999	1750 - 5667

If the maximum storage space is to be used, and the data is to be rearranged within electrostatic memory after the address sort, it might prove helpful to reassemble and use blocks of memory other than those utilized now.

## TIMING

For sorting data within electrostatic memory, the address sort is very efficient and is the only general program we have available. The special cases of a very small quantity of data, or data without indicative information could probably be processed more quickly by some other program, but the address sort will handle them.

If  $P$  = number of passes needed to complete the sort (ie., the total number of bits to be sorted divided by the number of bits sorted in one pass)

$U$  = Number of sorting units per sorting block.

then the total address sorting time  $T$  is given approximately by the expression:

$$T = 2 P U / 1000 \text{ seconds}$$

If  $n$  is called the number of bits sorted each pass, then a more exact expression for the total sorting time is:

$$T = \frac{1.76 + P(1.76 + .76n + 1.92 U)}{1000} \text{ seconds}$$

For example, to address sort 200 sorting units on three full words, seven bits per pass, and to carry along one or two full words of indicative information, would take about  $2PU/1000 = 2(15)(200)/1000 = 6$  seconds.

## USE

The address sort is the only general purpose sort for handling data within electrostatic memory that we have available, and therefore it is used extensively.

The address sort is well adapted to sort blocks of data on the tapes or the drums, provided that each block does not exceed the capacity of electrostatic memory. The blocks will be sorted independently of each other, and therefore machine failure should normally result in loss of only a few seconds or a few minutes machine time. If, however, the original data tape is damaged in processing, an indefinite amount of time is lost.

A revised version of this program is now being written making possible the following variations by the use of appropriate parameters:

1. Size of sorting unit and sorting field
2. Size of block
3. Location of sorting field within the unit
4. Size of character to be sorted on each pass
5. Location and number of characters in each word.

## OPERATIONAL EXPERIENCE

The address sort by itself utilizes only electrostatic memory, which has proven itself to be quite reliable compared to the other components of the 701. As a result of this factor and the rapidity or short duration of the sort it has not encountered any machine operating difficulties.

The address sort has not been tested extensively on single or multi-tape sorting, although it will be in the near future. It is expected to perform more reliably than any other type of tape sort, due to the minimum of tape references. The data tapes are read once only, and the sorted data is written once. Tape breaks, etc. may cause considerable trouble, but the possibility of tape trouble is minimized.



## BUCHHOLZ SORT

The "Buchholz" sort is an adaptation of an IBM program which was written for an earlier version of the 701.

The Buchholz sort is a merge sort, which merely means that it is patterned after a collator rather than a sorter. Instead of sorting data into "pockets" by some means, it sorts by building "sequences". In its present form, it is limited to sorting a maximum of one full tape of data, and each sorting unit must be in a separate unit record, identical in format with the other sorting units. It is an ascending sort.

In theory, the Buchholz sort is nearly identical with the standard 519 IBM collator as it is used to merge and sequence check. There is one major change. Normally, the collator is wired so that when a step-down condition occurs, either the collator stops or the out of sequence card falls into a different stacker. In the 701 version of the collator, a sequence change rather than a single step-down condition by itself results in a change in procedure.

A single step-down condition by itself does not constitute a sequence break. A sequence break consists of the following conditions:

1. Step-down in both primary and secondary feeds.
2. Step-down in one feed, and no cards in the other feed.
3. No cards in either feed (this is really an end of one merge pass).

In the 701 collator stackers are employed alternately. The stacker being used is reversed in each sequence break, so that alternate sequences go to alternate stackers. In this way each new sequence including the sequence breaking card, goes into the alternate pocket. At the end of a pass, the cards from each stacker are kept separated and are reloaded into the primary and secondary feeds in preparation for another merge pass. On the initial pass, only one feed is utilized. However, on the 701 version of the collator, the effect is that this one feed supplies cards to both the primary and secondary.

In the simplified flow chart this modification on the change of sequence is indicated by the block which alternated the tapes upon which the data is being written. The purpose of alternating the output tapes is to have the merged sequences about equally divided in number in two different files ready for the next pass, also to speed up the sorting.

With a few minor changes, the following explanation and example of the merge sort are taken from Buchholz's own description, IBM Report 11.010.241, April 25, 1951. The simplified flow chart shows most of the logic employed.

Four tapes are used, two tapes serving as the input and two tapes serving as the output. The roles of the pairs being interchanged after each pass. The memory location of the two unit records will be referred to as A and B. In addition to storing in A and B, it is necessary to store the control word of the last preceding unit record which was written on one of the two output tapes. This control word will be called P, and it starts at 0, and is reset to 0, for each pass.

Tapes  $T_1$  and  $T_3$  are possible inputs for memory locations A, and tapes  $T_2$  and  $T_4$  for B. During one pass  $T_1$  and  $T_2$  will be feeding unit records to A and B, respectively, with  $T_3$  and  $T_4$  acting as output tapes. During the next pass, A will receive information from  $T_3$  and B from  $T_4$ ,  $T_1$  and  $T_2$  being the output tapes. During the first pass only, the single tape  $T_1$  supplies unit records to both memory locations A and B.

In order to merge pairs of sequences from the input tapes into longer sequences on one of the output tapes, it is necessary to make a three-way comparison between A, B, and P. The comparison determines whether A or B should be written next, or whether a new sequence must be started. The process is perhaps best explained by an example. Let each control represent an entire unit record. Decimal rather than binary numbers are chosen for convenience.

A set of 13 "unit records" are assumed to be distributed on tapes  $T_1$  (400) and  $T_2$  (401) in an arbitrary manner as indicated below. These unit records fall into 7 sets of subsequences as indicated by the solid lines.

The object is to merge these sequences into larger ones, and to distribute them alternately onto tapes  $T_3$  (402) and  $T_4$  (403) starting with  $T_3$ , as indicated below.

MERGE DATA BEFORE PASS		MERGE DATA AFTER PASS	
$T_1$	$T_2$	$T_3$	$T_4$
45	01	01	08
53	53	45	13
67	13	53	30
08	30	53	83
02	27	67	
15	86	02	
	83	15	
		27	
		86	

The step-by-step procedure by which the result is obtained is given below. The data in parenthesis is unchanged from the previous step. The blank columns indicate unused decisions or data unreferrred to.

CONTENTS OF			DECISIONS		SWITCH OUTPUT	COMMENTS
A	B	P	Write A	Write B	TAPES	
		reset to				
45	01	00		X		
(45)	53	01	X			
53	(53)	45	X			
67	(53)	53		X		
(67)	13	53	X			step-down in B
08	(13)	67			X	<u>step-down in A</u>
(08)	(13)		X			
02	(13)	08		X		step-down in A
(02)	30	13		X		
(02)	27	30			X	<u>step-down in B</u>
(02)	(27)		X			
15	(27)	02	X			
END	(27)	15		X		end of A
RUN	{	86		X		
		83		X		
OUT		86			X	<u>step-down in B</u>
		(83)		X		
		END				end of B
		<u>83</u>				

To begin with, the first unit records from  $T_1$  (45) and  $T_2$  (01) are stored at locations A and B. They are compared in step 1, and the lower one, 01 in B, is written on  $T_3$ . It is replaced by the next record from  $T_2$ , 53, as shown on line 2. The control work of the outgoing record is stored in P. Both A and B are greater than P; this means that the sequence is not yet ended. A and B are compared again in step 2; this time A is smaller and it is written on  $T_3$ . In step 3, A and B are equal; either A or B could be written. An arbitrary convention is followed of always writing A when they are equal. B is written on the following line 4.

At step 5, it is found that the new B (13) is smaller than P. Thus B must belong to a new sequence. This condition is referred to as a "step-down" in B. Note that a single step-down by itself does not constitute a break in sequence. No more records from B can be added to the present sequence. But A is still greater than P, and it is now written. In step 6, however, there is also a step-down in A. Thus the sequence is ended, as neither A nor B can be added to the present sequence, and the output is switched to tape  $T_4$  to start a new sequence. In step 7, A and B are again compared, and A being the lower, it is written on  $T_4$ . P is effectively 0 for each start of a new sequence, although this result may be achieved by by-passing the comparison with P, rather than actually resetting it to 0. In step 7, A and B are again compared, and A being the lower, it is written on  $T_4$ . This continues until at step 10, there has been a step-down in both A and B. The output is switched back to  $T_3$ , and the merging continues.

At step 13, Tape  $T_1$  has come to an end, and hence there is nothing more in A. The unit records remaining in B and on  $T_2$  must still be run out to the output tapes. This is done by a simple comparison of P and B. So long as B is greater or equal to P, B is written on the current output tape. When B is less than P, the output tapes are switched and the process continues. When  $T_2$  also comes to an end, this pass is finished. A new pass may then begin using  $T_3$  and  $T_4$  as the input.

The end of sorting is indicated when no switching of output tapes occurs for an entire pass. If the sorted data is on tape  $T_1$  (400), the program stops; otherwise it rewrites it onto 400. This rewrite option, which may consume 5 minutes or so for a full tape, may be eliminated if desired.

#### CHARACTERISTICS OF PRESENT PROGRAM

This merge sort was written by W. Buchholz of IBM, and our adaptation for the 701 was written by H. Avram. The actual program was written in symbolic, and has been assembled into storage locations  $300_8$  through  $753_8$ , or a total of  $299_{10}$  half word locations.

The program must be supplied with the data indicated for the following locations, either manually or by a calling sequence:

$302_8 + N$	Number of full words in a unit record = N.
$303_8 + k$	Control word or word to be sorted upon is $k^{\text{th}}$ word of the unit record
$304_8 -$	First address of unit record A
$305_8 -$	First address of unit record B
$306_8 +$	Address where transfer to at end of merge.

The original file of unsorted data must be on tape 400, and the file of sorted data will end on tape 400.

The program stops at the end of the first pass, allowing the original data tape to be taken off and replaced, so that the original data need not be lost unless tape 400 breaks while being read the first time.

It is possible, with minor modifications which have been made and tested, but are not in the accompanying listing, to sort on either half word of the specified full word, instead of the full word.

Sorting non-positive numbers has not been tried, but the theory indicates that the numbers would be sequenced in the following normal order:

The largest valued negative number to - zero followed by plus zero and the positive numbers in ascending order.

This merge sort does not employ the Read Tape Backward order, which would result in considerable time saving. The Buchholz sort is written very compactly, is not easily modified and as we have not used it extensively, not too much effort has been expended in revising it.

## TIMING

Unlike many sorting programs whose operational time varies in general only with the amount of data being sorted, the time for the merge sort varies with the quantity of the data and the type of sort it is in already. In other words, the number of passes required to merge the data varies with the type of data. The maximum and minimum sorting time result when the data is in sort in reverse order, and in sort in ascending order respectively. If the data is random, the sorting time is one-half the maximum time. The number of passes required to complete the sort, and the respective times required are given below:

<u>TYPE OF DATA</u>	<u>NUMBER OF MERGE PASSES REQUIRED TO COMPLETE THE SORT</u>	<u>TIME REQUIRED (in seconds)</u>
In ascending sort	1	$2N (.0055n + .012)$ sec.
In reverse sort	next even integer which is = to $(\log_2 N)$	$2N (.0055n + .012) (\log_2 N)$ sec. even int.
In random sort	1/2 next even integer which is = to $(\log_2 N)$	$N (.005n - .012) (\log_2 N)$ sec. even int.

where  $N$  = number of unit records in the file  
and  $n$  = number of words in each unit record

As an example, a file of 1000 unit records in random order with 10 words per record requires 11 minutes to sort by merging.

If the tape were read backward instead of rewind, the above example would take a little less than 9 minutes.

The " $\log_2$ " term for a reverse sort arises from the fact that  $Q$  merge passes will suffice to merge roughly  $2^Q$  different step-down conditions, and for data in reverse sort  $N$ , the number of unit records also equal the number of step-down conditions. so that  $2^Q$  would =  $N$ , or  $Q$ , the number of passes would =  $\log_2 N$ .

When sorting considerable information, there is a severe disadvantage in putting each sorting unit into a separate unit record. This arises from the 1" length of the unit record gap, compared to the 1/16" required per full word. Thus, for any sorting unit consisting of less than 16 full words, more tape space is required for the unit record gap than for the data; for a two word sorting unit, 1/9th of the tape is used for data, and 8/9ths for the unit record gap. By allowing each unit record to contain a number of sorting units, such as is done in the bit sort, the tape is used more efficiently, and the machine time is correspondingly reduced.

## USE

The Buchholz sort is best adapted to sorting a moderate amount of data on tape on one or more full words. So far the Buchholz sort has not yet been used in production, but it has been tested on data and has operated successfully. It is being incorporated in a program which is not yet fully completed.

## OPERATIONAL EXPERIENCE

No true operating experience is available yet. It is worth mentioning that if something happens to tape 400 during the first pass original data may be lost. If, at any time after the first pass, something happens to the tape being read, and the original data tape has been saved, the program will have to be started again. If, after the first pass, something happens to the tape being written upon, the sort can be continued approximately from the point at which the trouble has occurred.

## INITIAL SORT

This is an original program, written as a companion program to the "merge sort" by L. B. Stinnett of our 701 group.

It is a "merge sort", as was the Buchholz sort, but this program merges information into "blocks" of data, sorting these blocks in on tape in a form which is usable by the "merge-sort" program for the completion of the merging. The program merges the data in an ascending order, and attempts to take advantage of random sequences.

It accepts information in the form of four words per sorting unit, and stores the information as two unit records of 100 sorting units each in a sorting block (that is each block of 200 units of data is in sort). It also fills out the unit records to an even number.

Two unit records at a time are read from tape, sorted, and written on another tape. 1600 words of electrostatic storage are set aside for data, twice the size of the data being sorted. Each unit record is sorted with ES, transferring sorted information between two sets of storage locations until each unit record is in sort. The two sorted unit records are then merged as the data is written on magnetic tape.

The program operates as follows: the first sorting unit of unit record 1 and the last sorting unit of the same unit record are compared in storage block A. The smaller of the two is stored in block B; the sorting unit adjacent to it is used for the next comparison; and the procedure is repeated. Where in block B the data is stored depends whether the unit stored previously is smaller or larger than the one being stored. If the ascending sequence is kept, the data is stored adjacent to the last location used; if the ascending sequence is broken, a new sequence is begun in block B at the far end of the block. The sequence therefore, are present in both ascending and descending order, within the storage block B. The procedure is repeated until the last piece of data is compared against itself; this condition is recognized as the end of block A processing, and the processing of block B now proceeds in the identical manner, transferring the newly merged data from block B to block A. The storing of the complete block of 100 sorting units without a change of sequence is recognized as the end of the single block sort. This is accomplished for two blocks, and the two blocks are then merged as they are recorded on magnetic tape in the form of two 400-word unit records.

An example is given below, using a configuration of a ten-unit block.

The original data appears as follows:

Block A:

loc. (1)	3	loc. (6)	6
(2)	8	(7)	7
(3)	2	(8)	6
(4)	6	(9)	3
(5)	5	(10)	1

1. The contents of locations (1) and (10) are compared,  $3_{(A1)}$  and  $1_{(A10)}$ .  $1_{(A10)}$  is stored into Block B, location (1).
2.  $3_{(A1)}$  and  $3_{(A9)}$  are compared.  $3$  is compared with  $1_{(B1)}$  and stored into Block B, location (2), and also location (3).
3.  $8_{(A2)}$  and  $6_{(A8)}$  are compared.  $6_{(A2)}$  is compared with  $3_{(B3)}$  and stored into (B4).
4.  $8_{(A2)}$  and  $7_{(A7)}$  are compared.  $7_{(A7)}$  is compared with  $6_{(B4)}$  and stored into (B5).
5.  $8_{(A2)}$  and  $6_{(A6)}$  are compared.  $6_{(A6)}$  is compared with  $7_{(A2)}$ . A new sequence is begun by storing  $6_{(A6)}$  into (B10).
6.  $8_{(A5)}$  and  $5_{(A5)}$  are compared; then  $5_{(A5)}$  with  $6_{(B10)}$ . A new sequence is begun by storing  $5_{(A5)}$  into (B6).
7.  $8_{(A2)}$  and  $6_{(A4)}$  are compared; then  $6_{(A4)}$  with  $5_{(B6)}$  and stored into (B7).
8.  $8_{(A2)}$  and  $2_{(A3)}$  are compared; then  $2_{(A3)}$  and  $6_{(B7)}$ . A new sequence is started in (B9) with  $2_{(A3)}$ .
9.  $8_{(A2)}$  is compared with  $8_{(A2)}$ . The end of the first merge is recognized;  $8_{(A2)}$  is stored in (B8).
10. The data appears in B block as follows:

(1)	1	(6)	5
(2)	3	(7)	6
(3)	3	(8)	8
(4)	6	(9)	2
(5)	7	(10)	6



11. Following the same procedure, starting with  $1_{(B1)}$  and  $6_{(B10)}$  and storing into the A block, the following sequence is found in A at the end of the merge:

A	(1)	1 then B	(1)	1 and A	(1)	1
	(2)	3	(2)	2	(2)	2
	(3)	3	(3)	3	(3)	3
	(4)	6	(4)	3	(4)	3
	(5)	6	(5)	6	(5)	5
	(6)	5	(6)	6	(6)	6
	(7)	6	(7)	8	(7)	6
	(8)	8	(8)	7	(8)	6
	(9)	7	(9)	6	(9)	7
	(10)	2	(10)	5	(10)	8

In this example, it can be seen that no advantage is gained by taking into account at only one point of the comparing routine, the ordering (by random) of the information. It is felt that a new sort should be written to take advantage of all types of runs in the data.

In order to use the program, the data must be written on tape and left at the end of file condition. There are four full words per sorting unit, three of which are compared, the last for identification. There are one hundred units per unit record. A calling sequence is used which contains the number of tapes (one or two), the address of the even tape to be read from, the total number of unit records and the number of words needed is the last record to make it a complete four hundred words.

The program has been used in two recurrent jobs. The length of time varies with the initial order of the data. Observed timing has ranged from 25 to 55 seconds (with an average of 45 seconds) per each 200 units sorted.

The instructions and constants occupy 548 half-word locations. The data occupies the last 1600 full word locations of electrostatic storage. The program reads from one or two tapes and writes on two tapes.

Normal tape difficulties have been encountered in the use of this program, and the data tapes both before and after the sort have been retained for safeguarding time expended in using this sort. The inflexibility of this particular program together with the poor timing achieved make it questionable as to its future use unless quite thoroughly revised.

## MERGE - SORT

This program was written as a companion program to the "initial sort", but can be used for ordering large amounts of information, given a preliminary sort by any method; the information, however must be written on the tapes in a prescribed form for the merge-sort. This program requires the data to be stored on two tapes in a unit record size of 400 full words. This unit record is further broken down into sorting units of four words, in which the sorting field may be 1-3 consecutive words of the sorting unit, and the remaining words indicative information.\* The "sorting block", or the number of sorting units already in ascending order is assumed by the merge-sort program to be 2 unit records--or 200 sorting units. The number of unit records on each tape must be even and not exceed 512. The initial processing preliminary to use of this program, which leaves the information in this form, will now be called "Pass 1"

"Pass 1" must also provide the following information:

1. The number of tapes (1 or 2) on which the information is to be recorded at the end of the merge-sort.
2. The address of the first of the 2 "write" tapes; that is, the tapes on which information is to be recorded at the beginning of the program.
3. The number of unit records on Tape A (first data tape.)
4. The number of unit records on Tape B (second data tape).

The program merges pairs of sorting blocks of information (one block from each tape) into a single new sorting block of twice the size, normally. The program also handles "short blocks" which may occur any time data tapes contain a number of unit records not equal to a powers of 2. On each pass, the information is read from two tapes (A and B), called the "read tapes", merged, and written on two tapes, the "write" tapes,  $A^1$  and  $B^1$ ; The complete  $A^1$  tape is written first, then the  $B^1$  tape. The "write" tapes  $A^1$  and  $B^1$  then become "read" tapes A and B, and vice-versa. This continues until all information makes up a single sorting block, or all the information is in sort. The final information can be written, under program control, on one tape (if the length permits) or two tapes. The use of sense switch 1 controls whether the data is to be rewritten in ascending order, if further processing makes this rewrite step necessary.

\* Our usage has been primarily with the first three words as the sorting field; the fourth has indicative information.

The program utilizes the "Read Backward" tape instruction to eliminate rewinding between passes and minimize operation time. All calculations necessary for a pass are made prior to that pass. All merging of information is done within the write and "write-copy" time of the program. The information is sorted alternately in descending and ascending order, the even passes being descending and the odd ascending (counting the initial sort as the first pass). The number of passes required,  $p$ , can be calculated from the number of unit records,  $N$ , where  $p^1 = \log_2 N$  and  $p = p^1$  if integral, or the next larger integral value, and  $N = \text{number of sorting units}/100$ . The program calculates  $p$  and comes to a stop after the  $p^{\text{th}}$  pass.

This program has been written by H. Avram in symbolic and assembled into absolute locations. Practically speaking, the program occupies all 2048 words of internal storage; the locations used for data manipulation are the last 1600 full words; the remaining locations are used for the sort proper, a drum recording and calling sequence for the tape rewrite routine, and a tape check routine which rereads the data tape check sense if desired. The program allows for locations  $\text{0000} - \text{0070}(8)$ , being used by FEJ 035, the loading program (or a calling sequence not written as yet), but erases these routines and relocates a portion of itself in  $\text{0000} - \text{0064}(8)$ .

The merge-sort program as it now stands is limited by the inflexibility of the unit-record size of four words. Sorting non-positive data has not been tried. The advantages of speed in sorting data in the proper format by use of this program makes it probable that further effort will be spent in making it more flexible, as well as writing another initial sort and a calling sequence for companion use.

The program itself must be supplied with the data indicated for the following locations:

- (1)  $1514_{(8)}$  number of tapes for final recording
- (2)  $1515_{(8)}$  address of "write" tape 1
- (3)  $1516_{(8)}$  number of unit records on "read" tape A
- (4)  $1517_{(8)}$  number of unit records on "read" tape B.
- (5)  $1264_{(8)}$  " $\text{001174}_{(8)}$ " to be inserted if original data tapes are to be retained.

(These tapes must be physically removed from the units.)

### Timing

The formula used for estimating the time for sorting on this program arrived at empirically is:

$$t \text{ (seconds)} = N \left[ 1.468 - (N-1) (.003274) \right]$$

where it does not include Pass 1 and  $N = \text{number of unit records}$ .

## Use and Operational Experience

The merge-sort has been used operationally in four large sorting programs, in each case with the "initial sort" as the companion program. It has been operated successfully on moderate amounts of data, but tape troubles have prevented truly successful effort for complete running on a regular schedule. One of these jobs, however, is a recurrent job on which our best success has been achieved. Tapes are normally saved from the initial sort, and should operation be impaired by tape errors or breakage, normal procedure is to repeat processing from the "initial sort" completed tape. Since this program is quite rapid, this procedure seems satisfactory at present.

Some thought has been given to rewriting the merge-sort to design greater flexibility into the program with a minimum of effort at the time of usage.

## TAPE - MERGE

This program is used in order to merge a large volume of data which has been sorted by single tapes or pairs of tapes by the use of the merge-sort.

Since of the four tape units available two must be used for reading, and two for writing, the program involves much operator handling of the magnetic tapes. The first of the two pairs of tapes are merged and written on the two write tapes. The program comes to a stop at the point at which either (or both) of the read tapes is exhausted as data, with an indication of which tape is to be replaced. (A sense switch is used to inform the program if the last of a set of tapes is being loaded). The next tape of that set replaces the used tape, and the program continues.

The program also stops if both write tapes have been used, and both tapes are then replaced by blank tapes. The program has been designed so that no unnecessary stops are made (as the end of the last tape of a set). There is no limit to the number of tapes that are to be merged, and any number,  $n_1$ , of tapes in set 1 can be merged with any number,  $n_2$ , in set 2.

However, the use of the program involves much external handling both of magnetic tapes and sense switches. This means that capable, alert operators are required for successful running of the program.

### Operational Experience

This program has not been used operationally up to the present time. It is felt advisable to avoid its use, if possible, since tape failures or operator errors can easily ruin a large volume of completed machine runs.

### Timing

The program time consumed is approximately the read plus the write time. Operator handling of the tapes must be added to this basic program time.

## AN EXAMPLE OF A LARGE SCALE SORTING OPERATION

An intriguing problem involving a relatively massive amount of 701 sorting is in process. It is mentioned as an example, as it effectively illustrates how several sorting methods could be combined.

One of the purposes of the following attack is to reduce the amount of material which could be lost at one time due to machine difficulties. It is also designed to use a comparatively small number of tapes, and to keep the problems of operation at a reasonable level.

The actual sorting should involve about 60 tapes of data, and five different sorting programs may be used altogether. This treatment assumes that only about 80 tapes are available. The successive steps will probably be as follows:

1. Load about 20 tapes of data in duplicate (ie., write the data onto 2 tapes simultaneously). Included is a tape check "re-reading" routine to be used for testing the duplicate tapes after they are made from the original data. This is to guarantee proper recording of data on the magnetic tapes before any sorting is attempted.
2. Block sort the 20 tapes into 8 blocks of about 3 tapes each. This will be done in three passes by a modified version of the bit sort. The length of the unit record will be 400 full words.
3. Sort the 400 full word unit records on each tape independently, so that each unit record is now in sort within itself. This will be done by the address sort, and the resulting tapes will be duplicated either during or after this step.
4. Merge all the unit records within a given block (note that before this step there are 8 blocks of about 3 tapes each). This will be done either by the "merge-sort" described earlier, or by a new merge sort now being written, followed by the tape merge sort. At the end of this step, each block will be in sequence. By arranging the blocks in the proper order as determined by the original bit sort, all 20 tapes will now be in sort.

These four steps will then be repeated on the two other groups of 20 tapes.

5. As a final step, the three groups of 20 tapes will be merged into a single sequence in one operation, and also sequence checked. This is the only major step which will not be backed up by duplicate tapes from the previous step, but it only requires one pass on each tape.

## SUMMARY

The entire field of sorting programs is still very young, as is the field of magnetic tape handling. Our experience has indicated that for small volume jobs, (those which can be handled in electrostatic storage), the multi-bit sort is efficient in terms of time and storage; in addition, it is extremely flexible for varying sizes of sorting field, indicative information, and selection of these fields without alteration of data.

For jobs which require storage on magnetic tapes, repetitive multi-bit sorts, though flexibility remains, do not provide the fastest method. It is felt that the following type of processing should be set up.

Large volumes of data exceeding capacity of two tape loads should be block-sorted (as Step 1) into blocks not exceeding two tapes of data. Thus external tape handling would be taken care of in the initial processing stages. All processing from this point on would involve an individual pair of tapes. Some jobs might practically be run (after sorting) on this piece-meal basis, really in the form of many small jobs.

After block-sorting, by either a bit or multi-bit method, each pair of tapes is sorted by use of multi-bit sort into blocks of 200 sorting units and recorded on tapes in the proper form for a merge-sort. The third step is the merge-sort to order each pair of tapes as a unit.

This procedure, in general, is planned for the current job with a large volume of sorting (over 3 million sorting units) previously described. At present, no practical experience on this sorting problem is available.

Consideration is being given to rewriting the merge-sort to make it more flexible for field sizes. It is felt that a maximum of approximately 400 words per unit record is a necessary limitation, regardless of the size of the sorting unit. This comes about because it is necessary to allocate electrostatic storage for four unit records in order to minimize the timing (to utilize all the "write" time for making the comparisons and selections). The program itself easily will use the remaining 800-odd half-words.

It is planned to design the program so that the sorting fields used can be flexible in terms of size (full or half words) and location (assignment of minor, intermediate, major); as a consequence, flexibility in the indicative information field will result.

It is further planned to use, as the initial sort companion program, a general sort of the "address" type (multi-bit sort). Some experimental programming will be done on further development of the present "initial sort" to take complete advantage of random sequences in the data. However, it is expected that this may not prove to be efficient or flexible enough to warrant its use in place of the multi-bit sort.

One other program is afoot to provide more effective operation in sorting programs. This is the incorporation of a sequence-checking routine to be used at the option of either the programmer or the operator. This routine will make possible the checking of the partial sort at the end of any selected pass within the processing of the data. It is felt that optional checking is a desirable feature.

Many of the problems incurred in the operation of our 701 have revolved around the use of the magnetic tapes. Air-conditioning, dust-filtering, and humidity-control problems have aggravated these tape troubles and have muddled the picture enough so that we cannot determine whether physical conditions or machine conditions are primarily responsible for the difficulties. Tape breakage is one large recurrent problem; tape checks, thought to be due to dust and also faulty tapes, are also prevalent. These types of tape checks are not susceptible to elimination by use of the tape sense order, as any number of rereads will consistently yield another tape check. Long time operation with magnetic tapes seems to be impractical. Measures have been taken to correct improper physical conditions, with special attention given to tape storage and humidity control.

It is sincerely hoped that improved sorting programs and magnetic tape operations will be clearly seen in the near future in our 701 operations.



ABSTRACT

NUMERICAL WEATHER PREDICTION

Joseph Smagorinsky  
U.S. Weather Bureau

The physical background and historical development of numerical weather prediction is summarized.

The nature of the mathematical problem and the work-load for a digital computer are then described.

ABSTRACT

NUMERICAL WEATHER PREDICTION PROGRAMMING

on the IBM 701

William P. Heising  
IBM

This paper will treat the programming of a specific numerical weather weather prediction problem of moderate difficulty. The overall layout of the program will be given together with specific attention given to checking and restarting methods and the form of the output.

NUMERICAL WEATHER PREDICTION ON THE IBM 701

W. P. Heising

IBM  
Washington, D.C.

*see Proc. E. Joint Conf  
conference Dec '53  
for refs.*

The Numerical Weather Prediction Unit, a joint group supported by the Weather Bureau, the Navy, and the Air Force, has been formed to perform numerical weather prediction operationally. In order to evaluate the IBM 701 as a possible computer for numerical weather forecasting, a test was made on the 701 using the mathematical system just outlined by Dr. Smagorinsky.

Lt. Commander Albert Stickle, U. S. Navy, Major Herbert Zartner, U. S. A. F., and I were assigned to program, test, and demonstrate three 24-hour forecasts. As none of us had previously prepared any problems for the 701, and the other two programmers had not even seen the 701 manual, it was a formidable task to accomplish this work in less than two months.

The main part of the computations involved the solution of Poisson's equation (modified by Helmholtz terms), and an integration step involving the evaluation of Jacobians. For a  $19 \times 19$  grid using three vertical levels in the atmosphere, this becomes:

Given:  $q_{ijk}^{\tau-1}; \phi_{ijk}^{\tau-1}; q_{ijk}^{\tau-2}; \phi_{ijk}^{\tau-2}$  and  $f_{ij}$

$i, j = 0, 1, 2, \dots, 18 \quad k = 0, 1, 2.$

a. Integration of  $q$ .

$\frac{\partial q}{\partial t}$   $q_{ijk}^{\tau} = q_{ijk}^{\tau-2} + \frac{c}{f_{ij}} \left[ J(f_{ijk}^{\tau-1}, \phi_{ijk}^{\tau-1}) + J(f_{ij}, \phi_{ijk}^{\tau-1}) \right]$

for  $k = 0, 1, 2$  and  $i, j \neq 0$  or  $18$ .

b. First approximation to Poisson Equation solve by extrapolation:

$\phi_{ijk}^{\tau} = 2\phi_{ijk}^{\tau-1} - \phi_{ijk}^{\tau-2}$

c. "Poisson Equation" solved by Liebmann iteration:

$\phi_{ijk}^{\tau} = \phi_{ijk}^{\tau-1} + 1/3 \left[ \left( \nabla_{ij}^2 \phi_{ijk}^{\tau-1} \right) + \left( \sum_{k=1}^2 \beta_{kl} \phi_{ijl}^{\tau-1} \right) - q_{ijk} \right]$

For  $i$  or  $j \neq 0$  or  $18$

Iterate and take  $\phi_{ijk}^{\tau} = \phi_{ijk}^{n\tau}$  when  $|\phi_{ijk}^{\tau} - \phi_{ijk}^{n-1\tau}| < 2^{-13}$  for all  $i, j, k$ .

Computations were performed on a  $19 \times 19$  square grid at three vertical levels in the atmosphere. Half word cells are used throughout the computations without rounding. All data blocks consisted of 364 half words, 2 half words for a check sum of the IBM K02 subroutine type, 361 half words for one level of the  $19 \times 19$  grid and one zero half word.

Computations use two variables at up to three different time points, hence 3 time points  $\times$  2 variables  $\times$  3 levels/variable  $\times$  364 half words per variable per level indicate the overall active storage requirement to be 6552 half words. Drums or tapes must be used, and the drums were chosen in this case. Three time steps of  $q$ , and  $\phi$  (6552 half words) were kept on drums at all times and one time step of data (2184 half words) plus  $2 \times 364$  half words of working storage (total 2912 half words) were assigned from electrostatic storage for data, thus leaving space for 1184 instructions, or slightly more than  $1/4$  of E.S. storage.

Of course after each time step, what had been  $\phi^{\tau-1}$  and  $q^{\tau-1}$  became  $\phi^{\tau-2}$  and  $q^{\tau-2}$  from the programmer's point of view. Rather than actually reading from the drum and rewriting in other locations on the drum 4368 half words, it was quicker to cyclically permute the drum addresses in the program itself to eliminate data shuffling completely.

Approximately 950 instructions are needed to perform and check the numerical step by step integration of  $\phi$  and  $q$ . In addition, after each six (half hour) time steps, the following is to be printed:

(a)  $\phi_{ijk}^{\tau}$  All values except where  $i$  or  $j = 0$  or 18

(b)  $dp/dt$

$$w_{ijk + \frac{1}{2}}^{\tau} = \delta_k \left[ (\phi_{ijk + 1/2}^{\tau+1} - \phi_{ijk-1/2}^{\tau+1}) - (\phi_{ijk+1/2}^{\tau-1} - \phi_{ijk-1/2}^{\tau-1}) \right] +$$

$$\frac{c}{f_{ij}} J (\phi_{ijk-1/2} - \phi_{ijk+1/2}) \quad k = \frac{1}{2}, \frac{3}{2}$$

The output calculation and printing require 730 instructions, hence they must be called from the drum before each use. The printing of a two digit row identification ( $j$ ) and 17 3-digit fields was accomplished using P04 slightly modified, with complete echo checking including sign.

The printing was arranged in a nearly square array, 0.70 inch vs 2/3 inch vertically. Consideration is being given to printing directly on a weather map.

The entire problem (except M10 used for decimal-to-binary conversion of initial data) required about 1700 instructions including about 400 from standard subroutines (K02 and P04). SO2 Assembly required about 30 minutes, and debugging required about 2 machine hours (spread over two days).

Three 24-hour weather predictions were made during the demonstration, and required about three hours. Occasionally K02 check sum stops from drum operation were encountered, but the "restart procedures" handled these with little time loss.

Two types of procedures were set up for handling error conditions. Since a 24-hour weather prediction required nearly an hour, it was undesirable to start over at the beginning unless the error occurred in the first few minutes. One procedure used a transfer of control to a routine in the program to restart that particular time step, disregarding all data in electrostatic storage, and working from the information on the drum. Usually the error did not recur, indicating the error had occurred in computation or in E.S. storage.

In case the error recurred, (indicating something had been written incorrectly on the drum), a different procedure was used.

A complete record of past integrations was kept on the tape, and in the second re-starting procedure the tape information was read back and reloaded on the drums, and then computation was resumed.

Each time step involves two main computations, integration and Liebmann iteration for the solution of the Poisson equation. The two computations are of comparable complexity and time (for one execution). The integration is performed twice (includes a check run) while the Liebmann iteration is performed  $n + 1$  times ( $n$  times for convergence, one additional for checking).

On the basis of a similar computation on the I.A.S. machine, it was estimated that 4-10 iterations of the Poisson equation might be required for convergence, and accordingly particular emphasis in programming for speed was given to the Poisson equation.

The actual machine run showed that only 1-2 iterations were required for convergence, and that consequently the integration took more time than the Poisson equation solution, a rather surprising development.

The value of extrapolation of  $\phi$  to obtain a first guess for the Poisson equation was shown by the fact that at  $\tau = 1$ , (where two previous time steps were not available), four iterations instead of one or two were required for convergence. Use of the previous solution, instead of extrapolation as first approximation throughout, would have lengthened the solution time 6-8 minutes.

Improvements contemplated in the present program are being made on the following lines:

- a. Tighten up the time integration (q) computation in view of its previously unsuspected importance on the running time.
- b. Use of a different type of check sum (in place of K02) to permit simultaneous reading (or writing) and checking.
- c. Restart procedures to be completely automatized.
- d. Master program on tape (instead of cards) to eliminate card input of instruction at the start of the problem.
- e. Output will be actual vertical velocity (instead of  $dp/dt$  a related quantity).
- f. Certain linearizing approximations are being eliminated, introducing slightly more complexity and higher accuracy.
- g. A different set of three vertical pressure levels is to be used in the future; the principal effect on the program will be that the relaxation factor in the Poisson equation will differ at different levels.
- h. Thought is being given to the inclusion of "smoothed" orography.

THE IMPROVED 701

*later called  
IBM 704*

Gene M. Amdahl  
IBM Engineering Laboratory  
Poughkeepsie, N. Y.

The Engineering Laboratory at IBM Poughkeepsie has, for some time, been engaged in a project of improvement of the 701. At the time the 701 was first planned and constructed, it was necessary to keep the machine logic as simple as was reasonable in order that the engineering and production schedules could be made as short as possible.

Since the 701's have been in use by ourselves and our customers, it has been possible to study the uses to which these machines have been applied, and consequently, to determine a number of logical additions which would be of general usefulness.

The improvements which are, at present, undergoing engineering development may be grouped under six headings, each of which will be discussed in greater detail later:

- 1) Operation code modification to make space available for the improvements.
- 2) Special instructions to reduce programming labor and running time.
- 3) Speeded-up multiplication and division.
- 4) An indexing system for automatic address modification.
- 5) A changed drum system with greatly improved access time and information transfer rate.
- 6) Built-in floating point operations.

The descriptions of these features are still subject to modification.

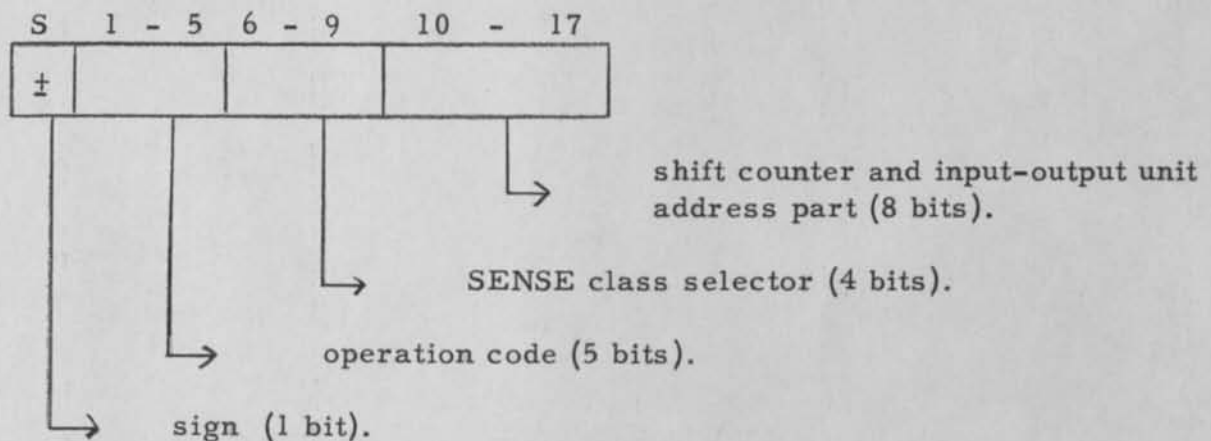
Modification of the Operation Code

In order to provide the necessary new instructions for the improved 701, the operation code had to be revised and consolidated. The means for

doing this were two-fold: 1) the signs of the instructions in many cases were not being employed by the machine; and 2) many instructions were not utilizing their address parts efficiently.

More efficient use of the sign of the instruction may be exemplified by our new handling of conditional transfers. The present conditional transfers will be obtained by a corresponding instruction having a positive sign, but if given a negative sign, the condition for transfer will be inverted. For example, plus transfer on overflow will cause a transfer if the overflow trigger is on, and minus transfer on overflow will cause a transfer if this trigger is off.

Efficient use of operation codes was attained by assigning operation codes to only those instructions which required a memory address for execution. Instructions such as input and output operations and shift instructions were grouped together under the single operation code SENSE, with the word divided as below.



In these instructions an eight bit address part is sufficient for specifying the input or output unit or the required amount of shifting. The four remaining bits of the address part are used to specify which input-output operation or shift is desired. The normal SENSE instructions are also included in this same instruction, and are designated by a particular value of the four bit portion and the proper eight bit designation. In order to use this consolidation efficiently, an additional STORE ADDRESS instruction is included, which allows the storage of the rightmost eight bits of the address part. This permits storing an input-output address or the amount of a shift into a SENSE instruction. The new STORE ADDRESS instruction shares an operation code with the present twelve bit STORE ADDRESS, but is designated by a minus sign.



## Special Instructions

The special instructions have been added primarily for the purposes of simplifying logical manipulations, including table lookup and producing check sums.

The purely logical instructions will include: full word logical "and" to memory; full word logical "or" to memory; full word logical "and" to the accumulator; full word logical "or" to the accumulator; and an MQ ring shift to the left, the sign included.

There are others provided also, such as: complement accumulator magnitude; set accumulator positive; set accumulator negative; change accumulator sign; set accumulator to zero; test rightmost bit of accumulator and skip; test MQ sign and skip; test P position of accumulator and skip; and test divide check (now to be called the MQ overflow trigger) and skip. This latter instruction permits the 701 to continue calculating even though a division was incapable of being carried out. In addition to the above, the status of the sense lights may be determined by a SENSE and SKIP.

The instructions for aiding table lookup are: compare MQ with accumulator and transfer if MQ is low; and compare memory with accumulator and skip if memory is high or equal. The instruction "compare MQ with accumulator" permits table lookup in data stored on the magnetic drum without requiring this data to be brought into memory, and allows this to be performed at the full new drum speed.

The computation of check sums is greatly facilitated by the new instruction END AROUND CARRY ADD, which adds the sign bit of a word in memory into the P position of the accumulator. The carries from the P position are reintroduced into position 35 of the accumulator. The sign of the accumulator is ignored and unchanged by this operation. With this instruction the check sum may be accumulated with no intervening manipulations.

## Speeded-up Multiplication and Division

The operations of multiplication and division at present require 38 machine cycles or 456 micro-seconds. By means of paralleling sub-operations and changing some of the logic, it has been possible to reduce the time required to 20 machine cycles or 240 micro-seconds. A corresponding reduction in the number of succeeding instructions which will not require forced regeneration cycles from 12 down to 6 is necessary. The actual multiplication and division operations are carried out at double present speeds.

## Indexing System

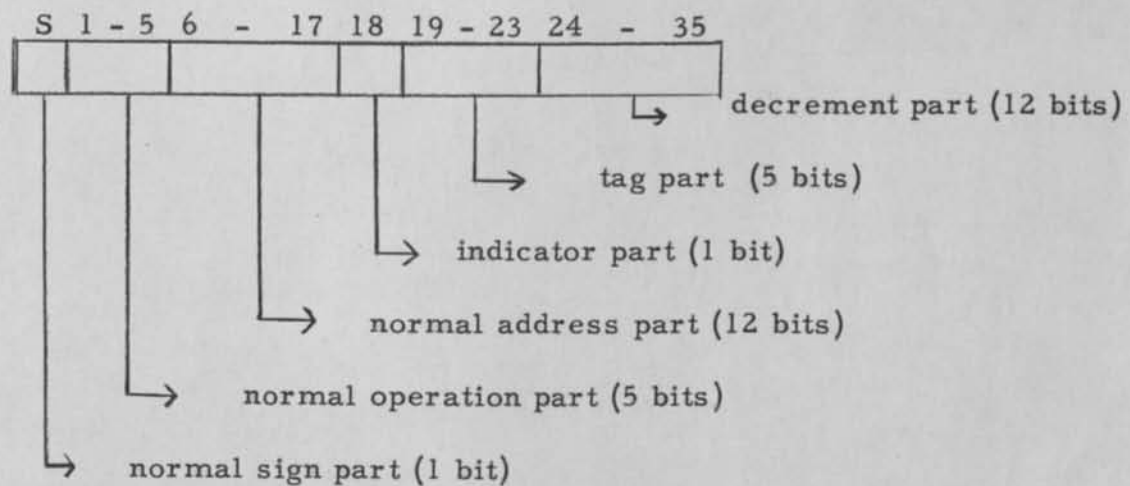
The indexing system provides automatic address modification which substantially reduces the amount of "red tape" in a program. On a typical problem the combination of indexing and speeded up multiplication and division seems to approximately double the speed of the 701.

Indexing is best described by means of an example, such as multiply and accumulate:

1.0	LOAD INDEX REGISTER;	1.1 + 01	2N
1.2	LOAD MQ	$A_0 + 2N$	; 1.3 + 01
1.4	MPY	$B_0 + 2N$	; 1.5 - 01
1.6	ADD	C	
1.7	STORE	C	
1.8	TR ON INDEX 1.2		; 1.9 + 01
1.10	rest of program		

The iteration loop above requires 348 microseconds to carry out compared to 696 for the shortest possible iteration loop on the 701. The storage requirements for the program are also less, being 10 half words compared to a minimum of 16 on the 701.

The more interesting features of the indexing system are shown in this program. For example, indexed instructions are full words, where the left half word is the normal instruction and the right half word contains indexing information. The allocation of digits in the full word instruction is shown below:



The sequence of operations when using index registers are essentially as follows: 1) the desired index register is loaded with the range of memory

addresses to be traversed during the iteration; 2) the instructions which are to have their address parts modified are executed, and in the process of execution they are reduced by the contents of the specified index register; and 3) the contents of the index register is reduced by the desired decrement and the loop repeated if the iteration is not completed.

There are a few features in this programmed example which are still unexplained. The tag part (or operation part of the right half word) specifies which index register is being used in any given instruction. The indicator part (or sign of the right half word) specifies whether or not to continue using full word instructions (or equivalently, whether or not to remain in the "indexing mode"). An example of this is 1.5 (or the right half word of 1.4), where the minus sign causes the calculator to go back to half word operation. The execution of an indexing instruction, such as 1.0 or 1.8 causes the calculator to return to the indexing mode.

The instruction TRANSFER ON INDEX performs the following sequence of operations: 1) the difference between the decrement and the original contents of an index register is generated; 2) this difference is tested to see if it is greater than zero; 3) if the result is greater than zero, the index register is indexed (that is, its contents are replaced by the difference) and if zero or negative, the index register retains its original contents; and 4) the calculator transfers to the address specified if the register is indexed and the instruction is positive (this being a conditional transfer, if it were negative, the condition for transfer would be inverted).

There are four more full word instructions in addition to TRANSFER ON INDEX, which control the index register contents. These are all addressless (SENSE class) instructions. They are: 1) load index register with the decrement part; 2) load index register with the complement of the decrement part; 3) load index register with the accumulator address part; and 4) put the algebraic difference of the index register and decrement part into the accumulator sign and address part.

There are also four half word instructions governing indexing: 1) an addressless instruction to force entering the indexing mode; and 2) a set of three instructions (one for each index register) which cause the complement of their address part to be entered into the proper index register.

### Higher Speed Drum System

The magnetic drum system has been revised in three ways. The first two decrease the access time and the third increases the word transfer rate.

The access time has been improved for reading by introducing electronic switching between drums. The electronic switching requires only one-half a millisecond compared to the present thirty milliseconds for relay switching. In the case of writing, relays are still used, but the switching time has been cut in half, from thirty down to fifteen milliseconds.

The access time has further been decreased by reducing the time required for index searching. This has been accomplished by two improvements: 1) the drum counter continues counting drum timing pulses from the last drum selected, even though no longer selected; and 2) the two logical drums on the same physical drum now share a common timing and index track, consequently, the drum counter will remain in synchronism even though switching between these two logical drums occurs.

The information transfer rate has been increased from the present 800 words per second to 10,000 words per second. This has been accomplished by reducing the interlace on the drum from 128 words down to 8 words. This interlace change is slightly too great to permit programmed word transfer even with indexing, so the drum has been physically slowed by about 20%.

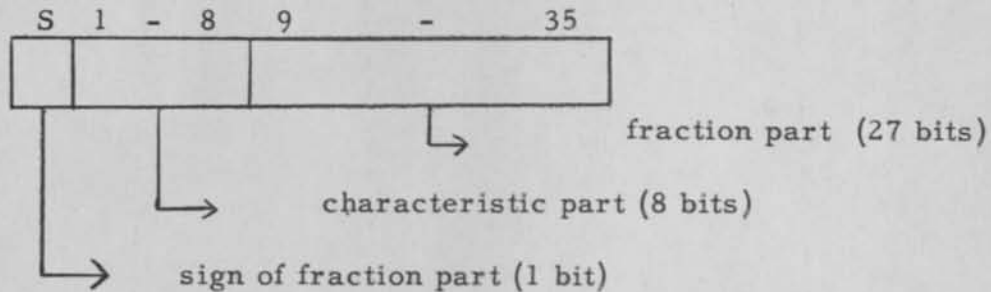
Keeping the drum counter continually connected to the timing track does not permit the operation SET DRUM to be performed, but a corresponding operation LOCATE DRUM ADDRESS has been introduced. The execution of this instruction causes the address part of the instruction to be continuously compared with the drum counter contents until equality exists, at which time the calculator proceeds to the next instruction.

Since the word transfer rate is 10,000 words per second, the time between copies (including a COPY instruction) is only 100 microseconds, hence there is time for only one instruction in addition to the COPY instruction. This additional instruction will normally be a TRANSFER ON INDEX instruction with the transfer address being the address of the COPY instruction. When performing table lookup on the drum, this additional instruction would be COMPARE MQ WITH ACCUMULATOR, with its transfer address again being the address of the COPY instruction.

### Floating Point

The problem of aiding the performance of programmed floating point operations was investigated quite intensively. The results of the investigation were that half measures gave little real advantage to the programmer, therefore, the decision was made to "build in" the set of floating point operations: add, subtract, multiply, and divide.

In order to be able to perform these operations with single instructions, it was necessary to combine the exponent and fraction parts in a single full word, much as in the Los Alamos Dual System. The arrangement of digits within a floating point word is shown below.



The characteristic part consists of 8 bits and contains the exponent of 2 plus 128. The fraction part is 27 bits long and when in the proper, or normalized form, has a 1 in its leftmost position. The binary point is considered to be just to the left of the fraction part.

When performing floating point operations, the original operands are brought into the accumulator or MQ by the normal operations of RESET AND ADD or LOAD MQ respectively. The floating point operation is then initiated by giving the proper floating point instruction. At all times during the floating point operations the arithmetic registers manipulate the information with the same division of digits as shown for the floating point word. The results may be stored by STORE or STORE MQ. Conditional transfers are still directly applicable.

If properly normalized numbers are used as operands, properly normalized results will be produced, with the exceptions of a zero result, or if normalization is inhibited. If the fraction part of a result is zero, the complete floating point word is made zero automatically. The inhibition of normalization is controlled by the programmer, and if the calculator is instructed to go into the non-normalizing mode, left shifts to remove leading zeros are inhibited in the operations of addition, subtraction and multiplication. All right shifts, to take care of carries as in addition, are still required to take place to prevent loss of information.

Characteristics which exceed the range of 0 through 255 will turn on the accumulator overflow trigger or the MQ overflow trigger (formerly divide check trigger) if they are the characteristics of the accumulator fraction part or MQ fraction part respectively. In the case of the accumulator characteristic, exceeding the range by getting too small will cause 1's to appear in both overflow positions P and Q. If too large, only the P position will contain a 1. The MQ register has no corresponding positions, so no determination of the direction of excess is directly available.

The floating point operations have been considered from the standpoint of double precision operation, and certain features have been included to expedite them. For example, all possible information is retained, and with the proper characteristic, so that it is readily usable.

The operations of addition and subtraction in floating point require 7 machine cycles, or 84 microseconds, to complete. If more than 10 shifts are needed to align the operands or more than 4 required to normalize the result, additional cycles will be required.

The operations of multiplication and division require 17 and 18 machine cycles respectively, or 204 and 216 microseconds.

### Program Revisions

For the use of our Poughkeepsie group, we are writing several programs to aid us in the conversion of our old 701 programs so that they can be run on our improved 701. The conversion essentially converts from the present instruction code to the revised instruction code. Two of these will be discussed very briefly.

The first of these programs accepts programs punched on binary cards, and punches out a new set of binary cards with the converted program. Prior to reading of the binary cards, a list of the addresses of those constants which should not be converted (i. e. which look like instructions but are not) must be supplied to the program. There are two items still unconverted which may be taken care of by octonary cards at this point: 1) putting the proper sign on those STORE ADDRESS instructions which refer to input-output or shift instructions; and 2) correcting input-output addresses which are stored alone as constants.

The second program is an assembly program which takes properly marked 701 assembly cards and assembles a converted program which will run on the improved 701. The proper marking consists of noting essentially the same information on the assembly card as was needed for the first program described.

In cases like the drum copy loops, these sections of the programs require a revision, for indexing must be used to copy from the high speed drum.

Amdahl: improved 701.

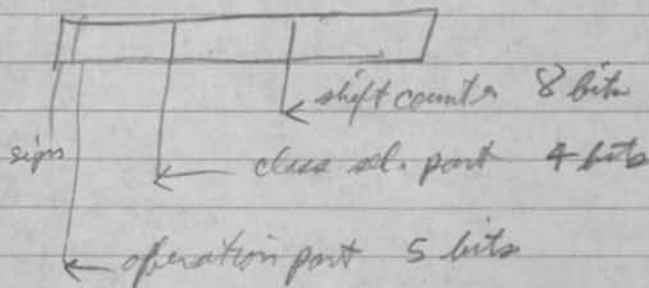
1. operation code modifications to make room.
2. reds - ?
3. mult & div speeded up.
4. indexing system.
5. changed drums.
6. built in f.p.

1. modifications:

use of signs of instructions

+TNOV if OV                      +TNO if 0  
-TNOV is not OV                  -TNO if non 0

Input-output instrs. all sense instrs.



new store addr instr - STA (right 8 bits only)

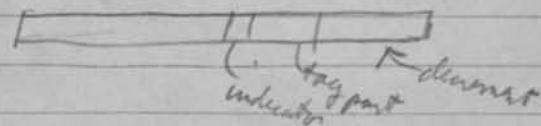
Logical: 1. full wd and, or to mem.

mult & div: - reduced to 20 machine cycles. (reduced no. of free games 12 to 6)

indexing:

example mult & accumulate.

- 1.0 Load index register +01 2H ← full wd. instr →
- 1.2 Load MA  $A+2N$
- 1.4 MPY  $B+2N$
- 1.6 add C
- 1.7 store C
- 1.8 Tn on index 1.2



higher drum.

- decrease access time (electronic switches).

$\frac{1}{2}$  ms instead of 30 ms.

- drum counter continues counting.

- 2 drums on same unit are synchronized.

- 128 interlace to 8

order: "locate drum addr."

10,000 wd/sec transfer rate.

(copy & one other order possible.)

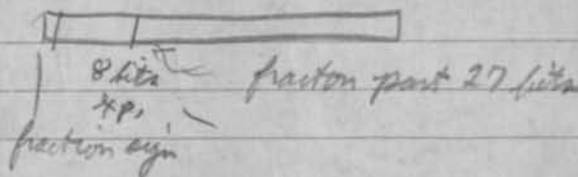
fl. point opns:

exp. & fraction

7 machine cycles	add	17
84 $\mu$ s	sub	18
+ additional	copy	?
if n	div.	?

204  $\mu$ s

206  $\mu$ s



can inhibit normalizing.

overflow of xp. will turn on

MA OV.

or acc OV

for double precision: - all data is retained

conversion routines: old instr cards  $\rightarrow$  new instr.  
assembly



OPERATIONAL CHARACTERISTICS OF THE 701A

John W. Backus  
Applied Science Division  
International Business Machines Corporation  
590 Madison Avenue  
New York 22, New York

29-1013-0(OC)

May 4, 1954

## OPERATIONAL CHARACTERISTICS OF THE 701A

IBM's efforts to improve its Electronic Data Processing equipment depend on two related areas of research:

1. Research in component development.
2. Research in logical design.

Research in component development produces the units which logical design research seeks to integrate into the most effective calculating or data processing machine possible. The 701A is the result of logical design changes in the 701. With additional equipment for control and storage of information the 701A makes use of the components of the 701 from two to twenty times more effectively during most calculations.

Although externally the 701A completely resembles the 701, new internal equipment and circuits provide many new and powerful features which make the 701A a faster, more economical, and convenient calculating tool than has previously been available. From an operating viewpoint, the 701A has briefly the following features in addition to those of the 701:

1. Automatic Floating Point Operation.
2. Automatic Address Modification.
3. Higher Speed Transmission of Information to and from Magnetic Drums.
4. 240 Microsecond Multiply and Divide time.
5. New Operations.

In the following material an attempt will be made to describe only those features of the 701A which are not to be found in the 701 or are different from corresponding features of the 701. This information is of a preliminary nature and may be subject to change.

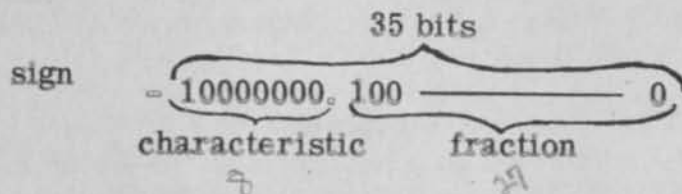
Before taking up any of these features in detail, it is first necessary to understand that the 701A has many more distinct operation codes than the 701. This is achieved by using two instruction forms, one identical to the present instruction form; the other having a ten-bit operation code (including sign bit) and an eight-bit address part as follows:

	S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
(A)	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
(B)	x	0	0	0	1	0	x	x	x	x	x	x	x	x	x	x	x	x

*3 types of  
OPN codes*  
*1. Small*  
*2. ordinary*  
*3. large*

Note that type (B) instructions all have the same number in positions 1 through 5 and that type (A) instructions may never have this number as an operation part, thus there is never any confusion about which type an instruction belongs to. All instructions which refer to electrostatic storage or to drum location are of type (A), most other instructions are of type (B).

Automatic floating binary point operation is achieved by four new type A operations: FLOATING ADD, FLOATING SUBTRACT, FLOATING MULTIPLY, and FLOATING DIVIDE. Floating binary information is represented in a full word by a sign, a characteristic, and a fraction. The characteristic is a positive integer between 0 and 255 inclusive and is equal to the exponent of the number increased by 128. The characteristic is given by bits 1 - 8 of a full-word. The fraction is given by bits 9 - 35 of the full word and is regarded as a 27-bit proper fraction. The sign indicated by the sign bit is associated with the fraction. Thus the floating binary point representation of  $-1/2$  on which the 701A can operate automatically is:



This representation of floating point numbers gives an equivalent decimal precision of more than 8 digits in the fraction and an equivalent decimal exponent range of better than  $10^{\pm 38}$ . A zero which results from a floating point operation is represented by a zero characteristic and a zero fraction with the sign determined as in fixed point arithmetic. A floating point number as described is said to be in normal form if the first bit of its fraction is a 1, or if the number is a floating point zero. All floating point operations refer to full words only.

When FLOATING ADD or FLOATING SUBTRACT is executed the floating point number in the accumulator is added to the contents of the specified full-word memory location. The floating point sum appears in the accumulator. In order to facilitate double precision operations a less significant portion of the sum appears in the MQ register in floating point form.

A floating point sum or difference is formed automatically as follows by the execution of FLOATING ADD or FLOATING SUBTRACT:

1. The MQ register is cleared.
2. The number specified in electrostatic storage is placed in the memory register, with inverted sign when subtracting.
3. If the number with the smaller exponent is in the memory register, the contents of the memory register and the accumulator are interchanged.
4. The fraction in the accumulator is shifted right a number of positions equal to the difference in the characteristics. Bits shifted out of the low order end of the accumulator enter position 9 of the MQ and continue to be shifted right in the MQ.

5. The sign of the MQ is made to correspond with that of the accumulator. (The sign of the number with the smaller exponent.)
6. The characteristic in the memory register is placed in positions 1 - 8 of the accumulator.
7. The fraction in the memory register is added algebraically to the fraction in the accumulator (positions 9 - 35).
8. If there is a carry out of position 9 into 8, positions 9 - 35 of both the accumulator and the MQ are shifted 1 position right and a 1 is inserted in position 9 of the accumulator.
9. If the fraction in the accumulator is zero, the accumulator is cleared.
10. The characteristic in the accumulator, reduced by 27, is placed in positions 1 - 8 of the MQ register unless the accumulator contains zero, in which case zero is placed in positions 1 - 8 of the MQ register.
11. If the floating point number in the accumulator is not in normal form, the fraction in the accumulator only is shifted left and the characteristic in the accumulator is adjusted until a one appears in position 9.

The programmer can cause step 11. to be done or to be omitted in the execution of FLOATING ADD and FLOATING SUBTRACT operations. This is controlled by a "normalizing indicator". If the indicator is on, step 11. is performed for all floating point additions and subtractions. If the normalizing indicator is off, step 11. is omitted. A pair of instructions, ENTER NORMALIZING MODE and LEAVE NORMALIZING MODE turn the normalizing indicator on and turn it off respectively.

In using floating point operations it is desirable to be able to detect the occurrence of a result with too large or too small an exponent, i.e. greater than +127 or less than -128. Too large an exponent corresponds to a characteristic greater than 255, too small an exponent to a negative characteristic. In all floating point operations, if the number appearing in the accumulator after the operation is executed has too large an exponent, the corresponding characteristic appears in position p, 1 - 8 and the overflow indicator is on. If such a number in the accumulator has too small an exponent, the ones'-complement-plus-one of the corresponding characteristic appears in position q, p, 1 - 8 and the overflow indicator is on. If the number in the MQ register after a floating point operation should have too large an exponent, only the low order 8 bits of the corresponding characteristic appear in positions 1 - 8. The MQ overflow indicator is turned on to indicate this. Similarly, if a number with too small an exponent should appear in the MQ as a result of a floating point operation, the low order 8 bits of the one's-complement-plus-one of the corresponding characteristic appear in positions 1 - 8 and the MQ overflow indicator is turned on.

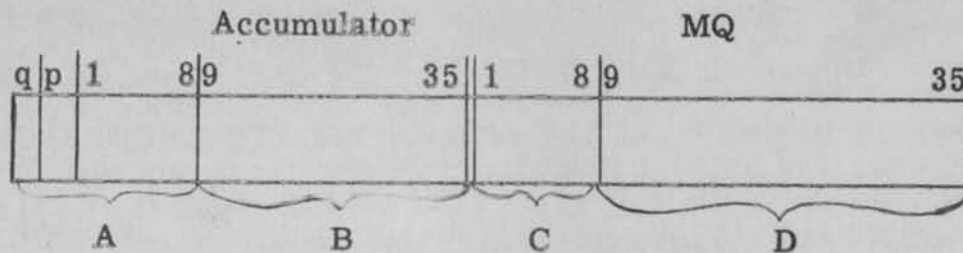
A few illustrations make clear how floating point addition and subtraction operate. For simplicity, assume that the characteristic is 3 bits instead of 8 and the fraction is 3 bits instead of 27. Assume that the accumulator has 6 positions plus two overflow positions, q and p and that the MQ has 6 positions. The normalizing indicator is on.

Operation	Contents of Memory Location			MQ after Operation						
	S	char	fraction	S	char	fraction	MQ after Operation			
1. FL ADD	+	100	100	+	00	100	off	+	111	110
2. FL ADD	+	100	111	+	00	101	off	+	010	000
3. FL ADD	+	100	101	+	00	101	off	+	010	100
4. FL ADD	-	101	111	+	00	110	off	+	011	110
5. FL ADD	-	000	010	+	00	100	off	-	011	100
6. FL SUB	-	111	110	-	00	111	off	-	100	100
7. FL ADD	-	111	111	-	01	000	on	-	101	100
8. FL ADD	+	001	100	-	00	000	on	+	101	000
9. FL SUB	-	000	100	-	11	111	on*	-	101	000
10. CLEAR & ADD	+	100	100	+	00	100	on	-	101	000
11. FL SUB	+	100	100	+	00	000	on	+	000	000

\*The overflow indicator would have been turned on at this point if it had previously been off.

Line 8 illustrates the fact that incorrect results are obtained if it is attempted to utilize for further calculations a result in the accumulator which has too large or too small an exponent.

The execution of FLOATING MULTIPLY causes a double precision floating point product of the number in the MQ register and the specified number in electrostatic storage to appear in the Accumulator and MQ register in the following way:



Where A and B are the characteristic and fraction of the most significant part of the product and C and D constitute the least significant part. C is always precisely 27 less than A unless A is less than 27 or greater than 255; in the former case, C is the ones-complement-plus-one of A-27, in the latter case C equals A-27 modulo 256. If the fixed point product of the fractions of the factors is less than one-half, and the normalizing indicator is on, A is the sum of the two factor exponents plus 127; if this product is not less than one-half, A is the sum of the two factor exponents plus 128. This means that if the two factors are in normal form, the number in the accumulator will be in normal form, provided the normalizing indicator is on. If it is off, A is always the sum of the factor exponents plus 128.

The execution of FLOATING DIVIDE produces a floating point quotient and remainder as a result of dividing the normal number in the accumulator by the normal number in the specified electrostatic storage location. The number originally in the MQ register is not regarded as part of the dividend. The quotient appears in the MQ register and the remainder in the accumulator. If the dividend is A, the divisor B, the quotient Q and the remainder R, the following relationship holds: if the most significant floating point part of the product  $Q \cdot B$  is denoted by QB and the least significant part by  $(QB)'$ , then the following operations result in A being in the accumulator and a number with a zero fraction in the MQ register:

CLEAR AND ADD	L( $(QB)'$ )
FLOATING ADD	L (R)
FLOATING ADD	L (QB)

If the divisor is not in normal form and its fraction is less than or equal to one-half the fraction of the dividend, the division will produce an incorrect result and the MQ overflow indicator will be on. Division of a normal dividend by a normal divisor always produces a normal quotient provided its exponent is in the allowable range. The position of the normalizing indicator does not affect the execution of FLOATING DIVIDE.

The execution of FLOATING ADD or FLOATING SUBTRACT will each require 7 basic machine cycles or 84 microseconds provided the exponents of the operands differ by 10 or less and normalization does not require more than 4 shifts.

If a and b are the exponents of the operands and  $a \geq b$  and c is the exponent of the result the exact number of basic cycles, n, required is:

$$n = 5 + r + s$$

$$r = \text{smallest positive integer containing } \frac{a - b + 2}{12}$$

$$s = \text{smallest positive integer containing } \frac{a - c}{4} \text{ if } a - c < 27$$

$$s = 1 \text{ if } a - c \geq 27$$

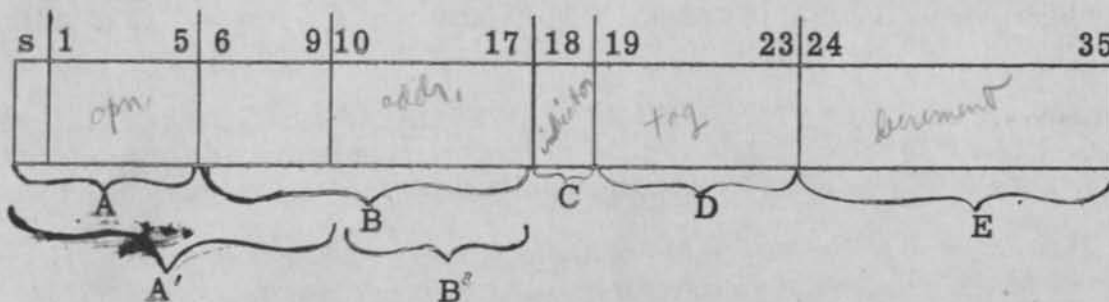
The execution time of FLOATING ADD and FLOATING SUBTRACT is not changed when they follow multiplication or division operations. The execution of FLOATING MULTIPLY or FLOATING DIVIDE requires 17 and 18 basic cycles or 204 and 216 microseconds respectively.

The six instructions following either of these operations may be executed in less than their normal time according to the principles pertaining at present to the twelve instructions following MULTIPLY or DIVIDE.

The automatic address modification feature of the 701A makes it possible to carry out so-called "housekeeping" operations involved in a problem with a minimum number of instructions and at a maximum speed. The problem of programming such operations is reduced accordingly.

The basic purpose of automatic address modification or "indexing" is to permit changing the addresses of many instructions by executing a single instruction. The indexing principle is to have information accompanying each instruction which may designate one of several special registers, called index registers. Then as each instruction is obtained for execution, the contents of the indicated register is applied as an increment or decrement to the address portion of that instruction before it is executed. Thus, the address part of an instruction as it is executed may differ from the address part of the instruction as it is stored in memory. If we call the address part of an instruction as it is executed its "effective address" then, of course, the effective addresses of all instructions which designate a certain index register are changed simply by changing the contents of that register.

Indexing is achieved in the 701A by allowing the machine to execute instructions in either of two modes: the "normal mode" or the "indexing mode". An "indexing indicator" determines which mode is to be used. The manner in which this indicator is turned on and off is described in subsequent paragraphs. When the indexing indicator is off, instructions are interpreted and executed as half-words in the normal way. When it is on, however, instructions are interpreted as "indexed instructions". Indexed instructions occupy a full word and the form of such instructions is as follows:



Where:

- A = operation part (6 bits)
- A' = operation part (10 bits)
- B = address part (12 bits)
- B' = address part (8 bits)
- C = indicator part
- D = tag part
- E = decrement part

The sign, the operation, and address parts of an indexed instruction have the same significance as they do in an ordinary half word instruction, thus the first half word of an indexed instruction may be any 701A instruction. The indicator part determines whether the indexing indicator is to be turned off after executing a given indexed instruction, a 1 denoting that it should be turned off, a zero that it should not be turned off. Thus, an indexed instruction with an indicator part of 1 signifies that subsequent instructions are to be executed in the normal way (as half words). The purpose of the tag part is to specify which, if any, of the three 12-bit index registers is involved in the interpretation or execution of the indexed instruction. A zero tag part indicates no index register is to be involved, 1 specifies index register 1, 2 specifies index register 2, 4 specifies index register 4. The tag part may have only these values. There is no index register 3. The decrement part has significance only for those indexed instructions which operate on the contents of an index register.

Most indexed instructions, called modifiable instructions, may have their address parts modified during interpretation if the tag part specifies an index register. Such instructions, without the second half word, may also be executed in the normal mode. There are a few instructions, however, which are peculiar to indexing which must always be executed in the indexing mode, therefore, if one of these instructions is encountered when the indexing indicator is off, it will be turned on. Since these instructions are always interpreted as indexed instructions they must occupy a full word. All of these latter instructions either change the information in an index register or obtain information from such a register. Since the tag parts in such cases are used to specify which index register is to be operated upon, the contents of the specified index register is not used to modify the address part of such an instruction, hence these are called non-modifiable indexed instructions.

There are a few other non-modifiable instructions which do not operate upon the index registers and which may be executed either as normal or indexed instructions but which do not have their address parts modified by an index register even though such modification is specified. These are: HALT AND TRANSFER, LOCATE DRUM ADDRESS (similar to the 701 operation SET DRUM), and SENSE-type instructions which are not related to indexing.

The interpretation of a modifiable indexed instruction takes place as follows:

1. The sign and operation part (6 bits) are placed in the instruction register.



2. If the tag part is zero the address part (12 bits) is placed in the instruction register.
3. If the tag part is 1, 2, or 4 the difference of the address part minus the contents of the specified index register is placed in the address portion of the instruction register provided the difference is positive. If it is negative, the one's-complement-plus-one of the difference (12 bits) is placed in the instruction register.

The instruction which results in the instruction register by the above steps is then executed in the normal way. The time required to interpret and execute an indexed instruction is the same as that required for the resulting normal instruction.

Since the interpretation of a modifiable indexed instruction modifies the instruction as if it had a 12-bit address part, care must be taken when an instruction with an 8-bit address part is to have its address modified automatically to assure that bits 6 - 9 of the instruction are not altered in the modification.

The operations involved in non-modifiable indexed instructions fall into three categories. In the first category are two operations which serve to insert information into an index register. These operations have SENSE type operation codes, that is, they have ten-bit operation codes and the remaining eight bits of the left half word are also used to completely distinguish the particular operation. The execution of LOAD INDEX REGISTER causes the number equal to the decrement part (bits 24 - 35) of the instruction to be placed in the index register specified by the tag part. The execution of ACCUMULATOR TO INDEX REGISTER causes the number given by bits 6 - 17 of the accumulator to be placed in the index register specified by the tag part. The decrement part has no significance.

The next category of non-modifiable indexed instructions consists of one instruction, also a SENSE type instruction, which serves to obtain the contents of an index register for use in a calculation or for storage. The execution of COMPARE INDEX REGISTER causes the signed difference of the contents of the index register specified by the tag part minus the number equal to the decrement part to appear in positions 6 - 17 of the accumulator. The other positions of the accumulator are zero.

The third category consists of two instructions which are perhaps the most powerful instructions in the 701A. Each of these instructions is designated by a six bit operation code (including sign), hence the address part may designate a location in memory. The execution of TRANSFER ON INDEX results in the reduction of the contents of the index register specified by the tag part by an amount equal to the decrement part and a transfer of control to the address given by the address part, provided such reduction would not give a zero or negative result. If the decrement part is larger than or equal to the contents of the specified index register, TRANSFER ON INDEX is equivalent to NO OPERATION, that is, the contents of the index register remain unchanged and the transfer does not take place. TRANSFER ON NO INDEX performs the same functions as TRANSFER ON INDEX except that transfer of control takes place on the opposite condition, i. e. when the

reduction would result in a zero or negative quantity.

These instructions are unique in that they are always executed as indexed instructions and will cause subsequent instructions to be executed in the indexing mode until an instruction with an indicator part equal to 1 is encountered:

LOAD INDEX REGISTER  
ACCUMULATOR TO INDEX REGISTER  
COMPARE INDEX REGISTER  
TRANSFER ON INDEX ←  
TRANSFER ON NO INDEX

There is one other instruction, a SENSE-type instruction, which will turn the indexing indicator on. ENTER INDEXING MODE is executed as a half word instruction and in addition to turning on the indexing indicator will cause the following indexed instruction to be obtained from the next higher full word location, regardless of whether ENTER INDEXING MODE itself occupies an even or odd address.

Finally, there are three instructions which place information in the index registers and may also have their address parts modified automatically. These instructions have 6-bit operation codes and may be executed as normal, half-word instructions or as indexed instructions. The execution of LOAD INDEX REGISTER 1, COMPLEMENT as a normal instruction causes the number equal to the one's-complement-plus-one of the address part to be placed in index register 1. One of the principal purposes of such an instruction is to provide a convenient, rapid method of specifying the address of a parameter to a subroutine. Thus, to specify the address X it is only necessary to give LOAD INDEX REGISTER 1, COMPLEMENT with address part X and to have indexed instructions in the subroutine which are to make reference to the specified location have address parts equal to zero and tag parts equal to 1. The corresponding two remaining instructions in this group are: LOAD INDEX REGISTER 2 COMPLEMENT and LOAD INDEX REGISTER 4 COMPLEMENT. When any of these instructions are executed as indexed instructions, they are interpreted by the same procedure used for other modifiable indexed instructions, i.e. the address part is reduced by the contents of the index register specified by the tag part, even if the tag part specifies the register which is to be loaded. The one's-complement-plus-one of the reduced address is then placed in the index register called for by the operation code.

All of the instructions described above with reference to indexing require 4 basic cycles for their execution unless a MULTIPLY, DIVIDE, FLOATING MULTIPLY or FLOATING DIVIDE is among the preceding 6 instructions, in which case 2 basic cycles are required.

The new timing conditions for Magnetic Drums in the 701A are as follows:

1. 48 microseconds are available between successive COPY instructions (in addition to the time required to execute

the COPY's themselves).

2. 48 microseconds are available between LOCATE DRUM ADDRESS (replaces SET DRUM) and the first COPY.
3. Consecutively-addressed drum locations pass under the reading heads at the rate of 10,000 per second, consequently COPY's are executed at this rate.
4. The execution of the first COPY will be completed 100 microseconds after the execution of LOCATE DRUM ADDRESS.
5. Any amount of time is available for programming between the execution of a READ or WRITE referring to drums and the interpretation of LOCATE DRUM ADDRESS.
6. The elapsed time,  $T$ , between the execution of READ or WRITE (referring to drums) and the execution of LOCATE DRUM ADDRESS is given by the following formula:

$$T = \max(r_1 + r_2, t) + r_3$$

$$r_1 = 15 \text{ milliseconds for WRITE}$$

$$r_1 = .6 \text{ milliseconds for READ}$$

$$r_2 = 0 \text{ if the drum selected is on the same physical drum last selected}$$

$$r_2 = \text{The elapsed time between the execution of the READ or WRITE and the appearance of a timing mark at the timing track reading head if the drum selected is not the one previously selected. In this case } r_2 \text{ will have an average value of 12.5 milliseconds and a maximum of 25 milliseconds.}$$

$$t = \text{time required to execute the instructions intervening between READ or WRITE and LOCATE DRUM ADDRESS.}$$

$$r_3 = \text{Time required for the location specified by LOCATE DRUM ADDRESS to appear under the reading heads (following the appearance of the timing mark if the READ or WRITE selects the physical drum not previously selected). The average value of } r_3 \text{ is 12.5. The maximum value is 25 milliseconds.}$$

The above formula gives an average access time of about 13 milliseconds for reading and 27.5 for writing when the last selected physical drum is selected and about 25.5 milliseconds for reading and 40 milliseconds writing when the other physical drum is selected.

It is important to note that this arrangement permits optimum programming for access to data on the same physical drum.

The new 701A instructions have a variety of uses. Their purpose is, of course, to simplify the job of programming certain common types of routines and to speed up the execution of such routines. Brief descriptions of these new instructions not described above follow.

Operations with 6-bit operation codes (including sign):

**END-AROUND-CARRY ADD** (full words and half-words). Full Words: The specified 36-bit number (including sign bit) is added positively into positions p, 1-35 of the accumulator, the number in the accumulator is regarded also as positive. Any carry out of position p is introduced into position 35. The sign and q position of the accumulator and the overflow indicator are not changed by this operation. Half-words: same as full word with left half-word as specified, right half word zero.

**STORE 8 RIGHT MOST POSITIONS OF ADDRESS** (half words only). Bits 10 - 17 of the accumulator are stored in the corresponding positions of the specified half-word leaving the remaining bits of the half word unchanged.

**HALT AND PROCEED** (no memory reference). The calculator stops; the next instruction is executed when the start button is depressed.

**HALT AND TRANSFER TO FIRST MEMORY FRAME.**

**HALT AND TRANSFER TO SECOND MEMORY FRAME.**

**TRANSFER NO OVERFLOW.**

**TRANSFER ON NOT ZERO**

**TRANSFER ON NOT PLUS**

**LOGICAL OR TO MEMORY** (full words only). A zero is stored in every position of the specified 36-bit word wherever the original word and the accumulator both contain zeroes and 1's are stored in every other position.

**LOGICAL AND TO ACCUMULATOR** (full words only). Analogous to present extract order except result appears in the accumulator and accumulator sign.

**LOGICAL OR TO ACCUMULATOR.**

**TRANSFER ON MQ LESS THAN ACCUMULATOR.** Transfer of control to the specified address takes place if the contents of the MQ is algebraically properly less than the contents of the accumulator.

**TEST MEMORY GREATER THAN OR EQUAL TO ACCUMULATOR AND SKIP.** Compare the contents of the specified full-word address with the contents of the accumulator and skip if the former is algebraically greater than or equal to the latter.

Instruction which has a 10-bit operation part:

**QUOTIENT LEFT RING SHIFT.** Ring shift the contents of the MQ

including the sign bit as a 36-bit register to the left by the amount specified by the 8-bit address part.

SENSE-type instructions. The whole half-word is used to designate the particular operation:

COMPLEMENT ACCUMULATOR MAGNITUDE. Inverts every bit of the accumulator. Sign bit is unchanged.

SET ACCUMULATOR POSITIVE. Make sign bit plus.

SET ACCUMULATOR NEGATIVE.

CHANGE ACCUMULATOR SIGN.

CLEAR ACCUMULATOR MAGNITUDE. Leaves sign unchanged.

The following SENSE-type test instructions cause the next instruction to be skipped if an indicated sign is negative or an indicated bit is a 1 or if a given indicator is on:

TEST LEAST SIGNIFICANT BIT OF ACCUMULATOR.

TEST MQ SIGN.

TEST P BIT.

TEST TA PE CHECK.

TEST MQ OVERFLOW INDICATOR.

TEST INDEXING INDICATOR.

TEST NORMALIZING INDICATOR.

TEST HALF WORD SELECT INDICATOR.

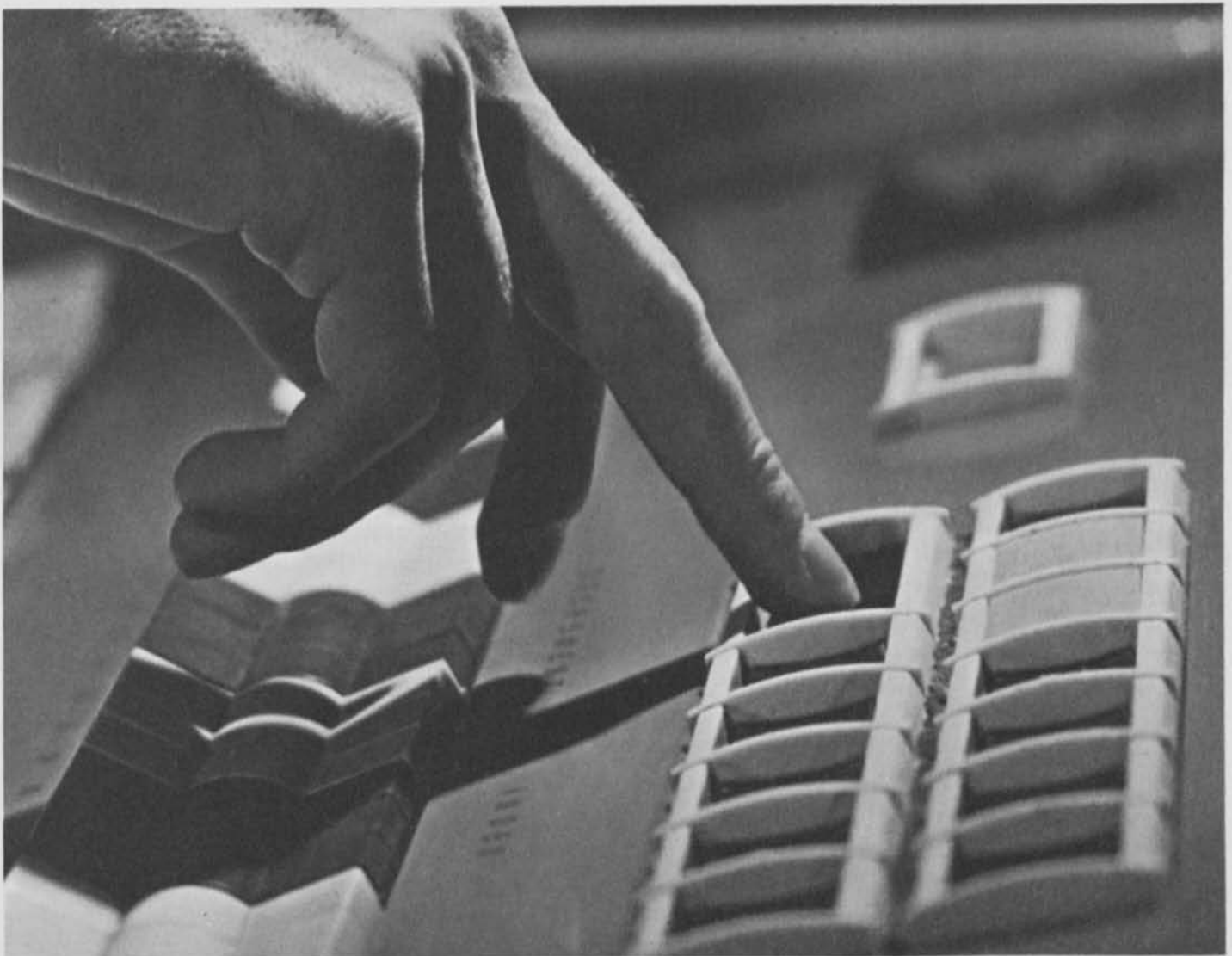
*mid 1955 production + delivery*

## Computer programming:

---

### How much does it cost to press the start button?

A computer is just a machine. No matter how fast, how powerful or how big it is, it can do only what you tell it to, the way you tell it to. So your program of instructions determines how well your computer operates. Programming systems, which come with every IBM computer, make it easier to instruct the machine to operate efficiently. They save programming time. They save operating time. They save you money.

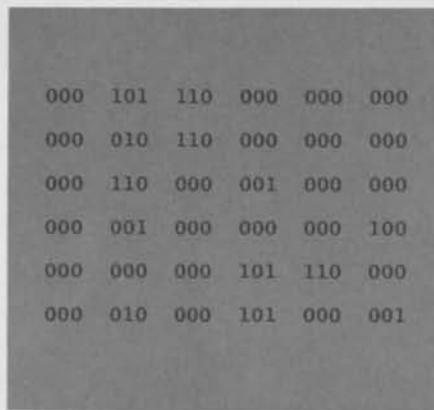
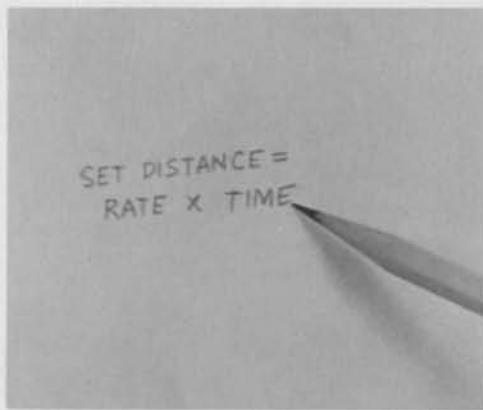


IBM

Computer programming starts with English-like statements that describe a problem.

An IBM computer automatically translates the problem to numerical computer instructions.

These instructions, on magnetic tape or punched cards, are used to operate the computer.



## A digital computer is a paradox

It can solve equations that would give a mathematician trouble. It can predict the path a missile will take. It can help you decide when to announce a new product.

But a man has to guide the computer through the decision making process—step by simple step.

Even multiplying one times two is a major problem: you have to tell the computer where to find the one and two, how to identify them, how to multiply them, where to find the answer and what to do with it.

Even a simple payroll job may take thousands of instructions. And these instructions must be written in a special numerical code—the only language the computer understands. Once the instructions are written they can be stored inside the computer and the system will perform the job automatically, as often as may be necessary.

But writing the instructions takes time. Each simple step has to be converted to a symbol the computer understands. And each computer type has a symbolic language all its own.

In the old days—five or ten years ago—a computer programmer laboriously had to write out every instruction in this computer code. He had a code book—a computer dictionary—to guide him. But it was a tedious, time-consuming job. And there was a chance for error on every instruction.

Writing this computer code cost money. An average instruction costs \$5 to \$10, counting the original

writing, checking, error correction and testing. And any sizable job took thousands of instructions.

That's why IBM has been working steadily for the past decade to simplify programming—to cut down the number of instructions your programmers must write, to make the symbols they use simpler and easier to understand, to standardize programming of common problems and eliminate the need to reprogram every function every time it's needed.

## Symbolic instructions— easier to remember

Our first efforts were to develop instructions that could be written with letters of the alphabet instead of numbers. For example, we would use the letter "P" instead of the numeral "4" to tell the machine to punch a card. The letter "P" is easier to remember, it is more meaningful to a human being, and it is less likely to cause a programming error.

In these symbolic codes, one program instruction represents one numerical instruction. A packaged program, supplied by IBM, guides the computer in making the translation.

Symbolic codes—much improved and updated—are still used in many applications.

## Common languages— fewer instructions

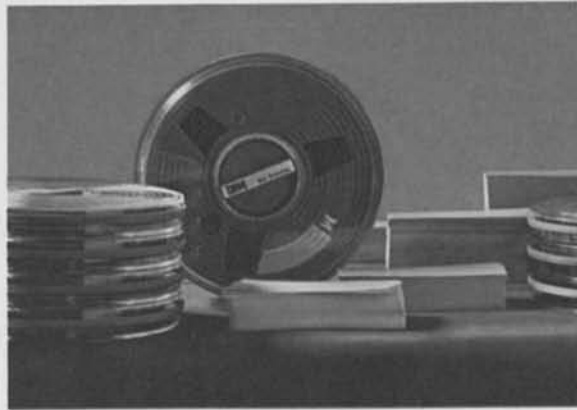
Common languages—such as COBOL and FORTRAN—take advantage of common problems, let your pro-

*IBM programming systems help you reduce computer programming and operating costs.*

*They also help you program input and output devices most efficiently.*

*A library of IBM Application Programs helps you solve specialized problems.*

# IBM



grammers give the computer one standardized instruction to represent a series of steps the computer must go through to perform a function.

These common languages—source languages, we call them—are written with symbols as nearly as possible like everyday English words.

For example, the programmer would write:

`SET DISTANCE = RATE * TIME.`

A program, supplied by IBM, would translate that to machine language that would look like this:

000	101	110	000	000	000
000	010	110	000	000	000
000	110	000	001	000	000
000	001	000	000	000	100
000	000	000	101	110	000
000	010	000	101	000	001

Thus, four English words and two arithmetic signs translate, in the computer, to the necessary machine instructions. Once the translation is made, you use the machine-language program to run your computer.

This saves time and money and reduces error.

It also enables the programmer to concentrate on the problem to be solved instead of the machine and its idiosyncrasies.

Obviously, it still costs money—computer time—to translate from source language to machine language. But it's a lot less expensive than doing it by hand. And

IBM is continually improving its translator programs so that they require less computer time.

## **Fast way to organize input and output**

So far we've been talking about programs that handle the processing of data more efficiently. Yet that is only one part of the data processing activity—the fast part.

It's just as important to speed up the handling of information as it goes into the computer and as it comes out. IBM has help for you here, too.

Programs, called Input/Output Control Systems, simplify programming of printers, card readers, tape files, disk drives and other input and output devices. They help your programmers generate efficient operating programs that include built-in error checks and signals to the operator when the end of a reel of magnetic tape or the end of a program is coming up. They help you make maximum use of your IBM computer's calculating power.

Sort/Merge Programs help you, too. They reduce the number of times you have to handle cards or spin tape to arrange an input or output file in proper sequence. They eliminate the need for complete programming of these routine functions.

Other IBM programs help you test a program before using it...convert your files from card to tape, tape to printer, tape to disk...change from one job to another or let you stack jobs on the computer so that it moves from job to job without help from the operator.



---

## What to look for in a data processing system

The point is a computer is just a machine. And, whether you rent or buy, it's the machine you pay for—its speed and information storage capacity. But it's the way you use the machine that pays off for you. And that's where IBM programs come to your aid.

They cut your programming costs. They give you generalized approaches and procedures for each machine; they quickly make translations from common languages like COBOL and FORTRAN to machine language—often require only one pass through the computer instead of two; they let you tailor computer functions to your problems and make it possible to put new jobs on your computer at minimum cost.

They also make computer processing, itself, more efficient. They give you programs that take less storage space in the computer (space that costs you money)... programs that take less time to execute... programs that permit better utilization of mechanical input and output machines.

## Application Programs for specific jobs

In addition, IBM offers you a variety of Application Programs—programs that are aimed at the solution of specific business problems. Like Demand Deposit Accounting for banks, AUTOSPOT for numerical control of machine tools, Inventory Management Simulation for a variety of industries. We also operate an information exchange service that permits customers, who wish, to exchange programming and problem solving information.

IBM programs offer you a more economical utilization of your IBM computer. All are available on punched cards or magnetic tape, ready to be used on your computer.

We also offer you the counsel and advice of programming experts when you need help. Their job is to help you solve your business problems more effectively with IBM Data Processing.

**IBM**<sup>®</sup>  
DATA PROCESSING



