Project Title: Autothrottle Test Procedure Compiler

Project No: A-2339

Project Director: Mr. J. E. Doss

Sponsor: Marconi Avionics, Inc.; Atlanta, Ga. 30338

Agreement Period: From 3/6/79 Until 8/5/79


Amount: $85,298 (Only Phase I, $9,409/One mth., auth. initially).

Reports Required: Monthly Progress Reports; Phase I Documentation; Phase II Documentation.

Sponsor Contact Person(s):

<table>
<thead>
<tr>
<th>Technical Matters</th>
<th>Contractual Matters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mr. Brian Wood</td>
<td>Mr. George E. Artman</td>
</tr>
<tr>
<td>Autothrottle Program Manager</td>
<td>(thru OCA) Procurement Manager</td>
</tr>
</tbody>
</table>

Marconi Avionics, Inc.
4500 N. Shallowford Road
Atlanta, Ga. 30338
Phone: (404)394-7800

Defense Priority Rating: N/A

Assigned to: Radar Instrumentation Laboratory (School/Laboratory)

COPIES TO:

Project Director
Division Chief (EES)
School/Laboratory Director
Dean/Director-EES
Accounting Office
Procurement Office
Security Coordinator (OCA)
Reports Coordinator (OCA)

Library, Technical Reports Section
EES Information Office
EES Reports & Procedures
Project File (OCA)
Project Code (GTRI)

CA-3 (2/76)
Project Title: Autothrottle Test Procedure Compiler

Project No: A-2339

Project Director: Mr. J.E. Doss

Sponsor: Marconi Avionics, Inc. Atlanta, Ga. 30338

Effective Termination Date: 7/31/80

Clearance of Accounting Charges: 7/31/80

Grant/Contract Closeout Actions Remaining:

- Final Invoice
- Final Fiscal Report
- Final Report of Inventions
- Govt. Property Inventory & Related Certificate
- Classified Material Certificate
- Other

Note: Final Report for Phase II submitted 9/20/79; Phase III Final Report not required

Assigned to: CSTL/SRD (School/Laboratory)

COPIES TO:

Project Director
Division Chief (EES)
School/Laboratory Director
Dean/Director—EES
Accounting Office
Procurement Office
Security Coordinator (OCA)

Library, Technical Reports Section
EES Information Office
Project File (OCA)
Project Code (GTRI)
Other

CA-4 (1/79)
AUTO THROTTLE TEST PROCEDURE
COMPILED

Monthly Status Report
6 March through 31 March 1979

EES/GIT Project A-2339

Prepared by

Computer Science and Technology Division
Radar and Instrumentation Laboratory
Engineering Experiment Station
Georgia Institute of Technology
Atlanta, Georgia 30332

J. E. Doss
E. S. Acree
C. A. Batterman
S. N. Cole

for

MARCONI AVIONICS, INC.
4500 N. Shallowford Road
Atlanta, Georgia 30338

under

Standard Industrial Agreement 3-6-79
OBJECTIVE

The design, specification, development, testing, installation, and documentation of the software specified under this contract.

WORK SUMMARY

The funds expended during the month of March are categorized as follows:

<table>
<thead>
<tr>
<th>Description</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Services</td>
<td>$3,081.53</td>
</tr>
<tr>
<td>Retirement Charges</td>
<td>$302.92</td>
</tr>
<tr>
<td>Materials and Supplies</td>
<td>$0.72</td>
</tr>
<tr>
<td>Overhead at 76%</td>
<td>$2,341.96</td>
</tr>
<tr>
<td><strong>Monthly Total</strong></td>
<td><strong>$5,727.13</strong></td>
</tr>
</tbody>
</table>

On March 6, work was begun on Phase I of the project. Phase I is the analysis of the problem and consists of the four tasks listed below. Also listed is the percentage of completion for each of the tasks.

<table>
<thead>
<tr>
<th>Task</th>
<th>Percent completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-code definition</td>
<td>100 %</td>
</tr>
<tr>
<td>Syntax rules</td>
<td>95 %</td>
</tr>
<tr>
<td>Estimate of ATP compilable statements</td>
<td>90 %</td>
</tr>
<tr>
<td>Hardware definition data structure</td>
<td>100 %</td>
</tr>
</tbody>
</table>

As can be seen, Phase I is nearly completed. A rough draft of the report for Phase I has been prepared and is in the process of being reviewed and refined. This report is scheduled for delivery to Marconi before April 6.

Upon approval of the Phase I results, Phase II will begin. Phase II consists of the actual implementation of the compiler. It is anticipated that the activities in April will consist of the completion and delivery of the Phase I report, initiating the design of the lexical analyzer, parser, and semantic routines, and the implementation of the lexical analyzer.
May 15, 1979

Mr. Brian Wood  
Marconi Avionics, Inc.  
4500 N. Shallowford Road  
Atlanta, Georgia 30338

Dear Brian:

Attached is the Monthly Status Report indicating progress on Project A-2339 for the month of April, and anticipated progress for the month of May.

If you have any questions or comments, please do not hesitate to call me.

Sincerely,

J. E. Doss  
Project Manager

JED/esc  
Attachment
AUTOTHROTTLE TEST PROCEDURE
COMPILER

Monthly Status Report
1 April through 30 April 1979

EES/GIT Project A-2339

Prepared by

Computer Science and Technology Division
Radar and Instrumentation Laboratory
Engineering Experiment Station
Georgia Institute of Technology
Atlanta, Georgia 30332

J. E. Doss
E. S. Acree
C. A. Batterman
S. N. Cole

for

MARCONI AVIONICS, INC.
4500 N. Shallowford Road
Atlanta, Georgia 30338

under

Standard Industrial Agreement 3-6-79
OBJECTIVE

The design, specification, development, testing, installation, and documentation of the software specified under this contract.

WORK SUMMARY

The funds expended during the month of April are categorized as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Services</td>
<td>$5,025.48</td>
</tr>
<tr>
<td>Retirement Charges</td>
<td>424.24</td>
</tr>
<tr>
<td>Materials and Supplies</td>
<td>77.16</td>
</tr>
<tr>
<td>Overhead at 76%</td>
<td>3,819.36</td>
</tr>
<tr>
<td><strong>Monthly Total</strong></td>
<td><strong>9,346.24</strong></td>
</tr>
</tbody>
</table>

These expenditures bring the total for the project to $15,073.37 as of the end of April.

During April, Phase I was completed and a report on the results was delivered to Marconi. After Marconi's approval of the results, Phase II was begun on April 12. Phase II is the implementation and documentation of the compiler and consists of the four tasks listed below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Percent completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Definition Translator</td>
<td>40%</td>
</tr>
<tr>
<td>Lexical Analyzer</td>
<td>90%</td>
</tr>
<tr>
<td>Parser</td>
<td>40%</td>
</tr>
<tr>
<td>Semantic Routines</td>
<td>40%</td>
</tr>
</tbody>
</table>

In May, it is anticipated that the Lexical Analyzer will be completed and that the progress of the other three tasks will continue at a similar rate.
Mr. Brian Wood  
Marconi Avionics, Inc.  
4500 N. Shalowford Road  
Atlanta, Georgia 30338

Dear Brian:

Attached is the Monthly Status Report indicating progress on Project A-2339 for the month of May, and anticipated progress for the month of June.

If you have any questions or comments, please do not hesitate to call me.

Sincerely,

J. E. Doss  
Project Director

JED/esc  
Attachment
AUTOTHROTTLE TEST PROCEDURE
COMPILER

Monthly Status Report
1 May through 31 May 1979

EES/GIT Project A-2339

Prepared by

Computer Science and Technology Division
Radar and Instrumentation Laboratory
Engineering Experiment Station
Georgia Institute of Technology
Atlanta, Georgia 30332

J. E. Doss

for

MARCONI AVIONICS, INC.
4500 N. Shallowford Road
Atlanta, Georgia 30338

under

Standard Industrial Agreement 3-6-79
OBJECTIVE

The design, specification, development, testing, installation, and documentation of the software specified under this contract.

WORK SUMMARY

The funds expended during the month of May are categorized as follows:

<table>
<thead>
<tr>
<th>Service</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Services</td>
<td>$4,991.81</td>
</tr>
<tr>
<td>Retirement Charges</td>
<td>415.63</td>
</tr>
<tr>
<td>Materials and Supplies</td>
<td>00.00</td>
</tr>
<tr>
<td>Overhead at 76%</td>
<td>3,793.78</td>
</tr>
<tr>
<td><strong>Monthly Total</strong></td>
<td><strong>9,201.22</strong></td>
</tr>
</tbody>
</table>

These expenditures bring the total for the project to $24,274.59 as of the end of May.

During May, work continued on Phase II of the project. Phase II is the implementation and documentation of the compiler and consists of the four tasks listed below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Percent completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Definition Translator</td>
<td>60%</td>
</tr>
<tr>
<td>Lexical Analyzer</td>
<td>100%</td>
</tr>
<tr>
<td>Parser</td>
<td>60%</td>
</tr>
<tr>
<td>Semantic Routines</td>
<td>60%</td>
</tr>
</tbody>
</table>

As can be seen, the Lexical Analyzer was completed in May. In June, it is anticipated that the progress of the other three tasks will continue at a similar rate as previously.
July 26, 1979

Mr. Brian Wood
Marconi Avionics, Inc.
4500 N. Shallowford Road
Atlanta, Georgia 30338

Dear Brian:

Attached is the Monthly Status Report indicating progress on Project A-2339 for the month of June and anticipated progress for the month of July.

If you have any questions or comments, please do not hesitate to call me.

Sincerely,

/John E. Doss
Project Director

JED/sd
Attachment
AUTOTHROTTLE TEST PROCEDURE
COMPILER

Monthly Status Report
1 June through 30 June 1979

EES/GIT Project A-2339

Prepared by

Computer Science and Technology Laboratory
Software Research Division
Engineering Experiment Station
Georgia Institute of Technology
Atlanta, Georgia 30332

J. E. Doss

for

MARCONI AVIONICS, INC.
4500 N. Shallowford Road
Atlanta, Georgia 30338

under

Standard Industrial Agreement 3-6-79
OBJECTIVE

The design, specification, development, testing, installation, and documentation of the software specified under this contract.

WORK SUMMARY

The funds expended during the month of May are categorized as follows:

<table>
<thead>
<tr>
<th>Category</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Services</td>
<td>$4,482.28</td>
</tr>
<tr>
<td>Retirement Charges</td>
<td>$349.42</td>
</tr>
<tr>
<td>Materials and Supplies</td>
<td>$416.06</td>
</tr>
<tr>
<td>Overhead at 76%</td>
<td>$3,406.53</td>
</tr>
<tr>
<td><strong>Monthly Total</strong></td>
<td><strong>$8,654.29</strong></td>
</tr>
</tbody>
</table>

These expenditures bring the total for the project to $32,928.88 as of the end of June.

During June, work continued on Phase II of the project. Phase II is the implementation and documentation of the compiler and consists of the four tasks listed below:

<table>
<thead>
<tr>
<th>Task</th>
<th>Percent completed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware Definition Translator</td>
<td>80%</td>
</tr>
<tr>
<td>Lexical Analyzer</td>
<td>100%</td>
</tr>
<tr>
<td>Parser</td>
<td>85%</td>
</tr>
<tr>
<td>Semantic Routines</td>
<td>90%</td>
</tr>
</tbody>
</table>

During July, it is anticipated that the four tasks listed above will be completed. The four tasks will then be integrated into a single program and tested with the Marconi-supplied Hardware Definition file and sections 5.9 and 5.19 of the ATP. It is also anticipated that the Assembler section of the semantic routines will be delivered to Marconi in early July to facilitate testing of Marconi's interpreter.
SOFTWARE DOCUMENTATION

AUTOThROTTLE TEST PROCEDURE COMPILER

By

J E. Doss
E.S. Acree
C.A. Batterman
J.B. Thompson

Software Research Division
Computer Science and Technology Laboratory
Engineering Experiment Station
Georgia Institute of Technology
August 20, 1979

Prepared for

Marconi Avionics, Inc.
4500 N. Shallowford Road
Atlanta, Georgia 30338
Under Contract 40125
DOCUMENTATION SUMMARY AND FLOWCHART INDEX

I. MAIN COMPILER ROUTINES
   1. MAINATP
   2. QUEST

II. LEXICAL ANALYZER
   1. CLAUSE
   2. DIGITS
   3. ERRMES
   4. GETDIG
   5. GETLIN
   6. GETTOK
   7. LEX
   8. LEXINT
   9. LEX2
  10. PSHTOK
  11. SEARCH
  12. SMATCH
  13. SPLIT
  14. STRING
  15. STRIP
  16. UNITS

III. COMPILER UTILITY ROUTINES
   1. CONVRT
   2. INSERT
   3. ODFUMP
   4. OUTCH
   5. OUTINT
   6. OUTLAB
   7. OUTNUM
   8. OUTSTR
   9. OUTTAB
  10. SEMERR
  11. SETSTR
  12. SYNERR
IV. PARSING AND CODE GENERATING ROUTINES

1. ACTION
2. CHANGE
3. DADJ
4. DISCON
5. DMEAS
6. DMONV
7. DSET
8. GCHECK
9. GCON
10. INIT
11. PARSE
12. STARTS
13. TRANS
14. UPRINT

V. HARDWARE DEFINITION AND CONVERSION ROUTINES

1. BLDTBLS
2. DETAB
3. GETTL
4. HCONV
5. LMEAS
6. LOCDAC
7. LOCDS
8. LOCLD
9. LOCPIN
10. PLOOK
11. POPEN
12. SELGN
13. SELIM
14. SRCHCT

VI. ASSEMBLER ROUTINES

1. ADIREC
2. ASEND
3. CHKHEX
4. CHKLB
5. DEVADR
6. DRIVER
7. ERMSG
8. FINDLB
9. FIRST
10. FLUSH
11. GCHAR
12. GENREG
13. GETNEX
14. GETOK
15. HEX
16. LABDEC
17. LITCH
18. MULGEN
19. OUT
20. OUTADC
21. OUTCON
22. OUTDEV
23. OUTGN
24. OUTHEX
25. OUTHI
26. OUTLAB
27. OUTLIT
28. OUTLOW
29. OUTREG
30. OUTRL
31. OUTSW
32. OUTTOL
33. OUTVAL
34. OUTWD
35. PARAM
36. PASSM
37. PCODE
38. PGEN
39. RCONV
40. STORE
41. VALTOL
I. MAIN COMPILER ROUTINES

1. MAINATP
2. QUEST
PROGRAM NAME - DRIVER ROUTINE FOR THE ATP COMPILER

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : MAIN DRIVER ROUTINE

CALLED BY : NONE

CALLS : PARSE QUEST

DATA GENERAL SYSTEM ROUTINES:
OPEN
FCLOS
OPEN
CLOSE
OVOPN
DFILW

FILES USED:
UNIT# NAME
2 TOKENFILE
3 SOURCE INPUT
4 HARDWARE COMMON BLOCKS AND CONNECTION FILE
5 PIN ID FILE
10 USER TERMINAL OUTPUT
11 USER TERMINAL INPUT
12 SOURCE LISTING
13 P-CODE FILE (OUTPUT)

COMMON STORAGE:
/TSPEC/ TSN(5)
/LICENSE/ LISTIN, SOURCE(20)
/LICENSE/ KOUTPT, KSRCE, KOBJ
/OUTPUT/ OPTR, OUTBUF(40)
/ERRKNT/ NUMERR
/VARTAB/ VARI(50), NUMVAR
/LABST/ LABEL
/ACETC/ DCDACNO, ACDACNO, DCLOW, ACLOW, DCHIGH, ACHIGH, GAIN, GAINRNG, COND, CONDFAC, ANGTOL, DCVLTL, ACVLTH, DCVLTH, ACVLTH, DCVEFT, ACVEFT, ACPHASE
/CTBLS/ NUMPINS, PINTAB, PRTAB
/PINS/ ADADDR(3), CNPTR, NUMENT, NUMSM, ATTRIB(2,10), POS(10), SCALE, REFER, SEMFLG

INPUT/OUTPUT OPERATIONS:
OUTPUT TO USER ON TERMINAL (10) AND ON LISTING (12)
INPUT FROM USER ON TERMINAL (11)

PURPOSE:

THIS ROUTINE CONTROLS THE MAIN CALLING SEQUENCE OF THE INITIALIZATION ROUTINES AND THE COMPILER AS WELL AS ALLOWING THE USER TO SELECT THE VALUES OF CERTAIN PARAMETERS.
MAIN COMPILER DRIVER ROUTINE

START

CLEAR ERROR COUNT

OPEN OVERLAY FILE AND LOAD IN OVERLAY

QUEST

SET USER PARAMETERS; INITIATE TABLE AND VARIABLES

INITIALIZE THE OUTPUT BUFFER AND ITS POINTER

PARSE

COMPILE SOURCE FILE

REWRITING MORE FILE AND OUTPUT THE VARIANT FILE

DISPLAY NUMBER OF ERRORS FOUND

STOP
Program Name - User Questioning Routine (Quest)

Programmer - Elaine Strong Acree
Engineering Experiment Station
Georgia Institute of Technology

Computer System - Data General Eclipse

Module - Main Driver Routine

Called By - Main Driver Routine

Calls - Lexint

Data General System Routines:
- Fopen
- Fclose
- Open
- Close
- Dfilw

Files Used:

<table>
<thead>
<tr>
<th>UNIT#</th>
<th>NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Token File</td>
</tr>
<tr>
<td>3</td>
<td>Source Input</td>
</tr>
<tr>
<td>4</td>
<td>Hardware Common Blocks and Connection File</td>
</tr>
<tr>
<td>5</td>
<td>Pin Id File</td>
</tr>
<tr>
<td>10</td>
<td>User Terminal Output</td>
</tr>
<tr>
<td>11</td>
<td>User Terminal Input</td>
</tr>
<tr>
<td>13</td>
<td>P-Code File (Output)</td>
</tr>
</tbody>
</table>

Common Storage:

- /SLIST/  LISTIN, SOURCE(20)
- /LISTS/  KOUTPT, KSRCE, KOBJ
- /Tspec/  TSNO(5)
- /Output/ OPTR, OUTFUT(60)
- /Errknt/ NUMERR
- /Vartab/ VARI(50), NUMVAR
- /Labst/  Label
- /Line/   LINBUF(132), LINSIZ, LINBPT, LINNUM
- /Tokens/ TOKBUF(35), TOKPT, TOKSIZ
- /Tables/ TOKTAB(100,8), NUMTOK, TOKWID, TOKCOD(100,3)
- /Errs/   ERROR, LINOLD
- /Qouted/ QUOTE, APOSTR, BLANK
- /Sects/  NBEGIN, NEND
- /Headin/ IDATE(3), ITIME(3), PAGENO, NUMOUT
- /Conhex/ HEXLIST(16)
- /Codes/  TCODES, IVALUE
- /Number/ RNUMB, INUMB, TRNUMB, TINUMB
- /String/ NUMCHR, SBUFER(40)
- /Pushed/ PUSH, LINPSH
- /Paren/  LEFT, RIGHT

/DACETC/ DCDACNO, ACDACNO, DLCLOW, ACLLOW, DCHIGH, ACHIGH, GAIN, GAINRNG, COND, CONDFAC, ANGTOL, DCVLTL, ACVLTL, DCVLT, ACVLT, DCDEF, ACDEF, ACPHASE
INPUT/OUTPUT OPERATIONS:
OUTPUT TO USER ON TERMINAL (10) AND ON LISTING (12)
INPUT FROM USER ON TERMINAL (11)

PURPOSE:

THIS ROUTINE CONTROLS THE MAIN CALLING SEQUENCE
OF THE INITIALIZATION ROUTINES FOR THE COMPILER
AS WELL AS ALLOWING THE USER TO SELECT THE VALUES
OF CERTAIN PARAMETERS.

**********************************************************************************************************************************************
START

GET SOURCE
FILE NAME
FROM USER

FIND OUT
IF USER
WANTS A
LISTING

LEXINT

INITIALIZE THE
LEXICAL ANALYZER'S
TABLES AND
VARIABLES

GET FIRST
LINE OF
SOURCE FILE

LOOK FOR THE
EQUAL SIGN
IN THE FIRST
LINE OF THE SOURCE

A page 2
A

EQUAL SIGN FOUND?

YES

FIND THE FIRST NON-BLANK CHARACTER AFTER EQUALS

NON-BLANK CHAR FOUND?

YES

PUT THE TEST SPEC INTO THE TEMPORARY BUFFER

NO

DISPLAY MESSAGE INDICATING NO TEST SPECIFICATION NUMBER

NO

FIND OUT IF USER WISHES TO HALT

HALT?

YES

CLOSE FILES

NO

REWIND SOURCE FILE

NO

STOP

INSERT THE PACKED # INTO THE PERMANENT ARRAY

DISPLAY TEST SPEC NUMBER

page 3
QUEST
II. LEXICAL ANALYZER

1. CLAUSE
2. DIGITS
3. ERRMES
4. GETDIG
5. GETLIN
6. GETTOK
7. LEX
8. LEXINT
9. LEX2
10. PSHTOK
11. SEARCH
12. SMATCH
13. SPLIT
14. STRING
15. STRIP
16. UNITS
PROGRAM NAME - BEGIN/END CLAUSE HANDLING ROUTINE

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY : LEX

PARAMETERS :
INDEX - INTEGER TOKEN CODE FROM THE TOKEN TABLE
EOF - LOGICAL END OF FILE

CALLS : GETTOK

FILES USED: NONE

COMMON STORAGE :
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/STRINGY/ NUMWRD, SBUFER(40)
/SECTS/ NBEGIN, NEND
/VARTAB/ VARI(50), NUMVAR

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
LUSE HANDLES THE BEGIN AND END TOKENS; AND IT PUTS THE REVISION NUMBER (IF ANY), THE SECTION/SUBSECTION NUMBER, AND THE SECTION NAME (FOR A SECTION ONLY) INTO THE STRING BUFFER (SBUFER).
A

CHAR = SLASH ?

NO

SET VARIANT NUMBER TO DEFAULT VALUE

YES

REMOVE THE VARIANT NUMBER FROM THE TABLE
TRANSFER REVENUE INTO STRING BUFFER

CHECK VARIATION TABLE FOR THE PRESENCE OF THIS VARIATION. IF IT IS NOT THERE, INSERT IT

TRANSFER SUBSCRIPTION NUMBER FROM NUSER INTO THE STRING BUFFER

RETURN
PROGRAM NAME - DIGIT CHECKING AND CONVERSION (DIGITS)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :
SMATCH
GETDIG

CALLS : NONE

PARAMETERS :
CHAR - INTEGER HOLDING ASCII CHARACTER TO BE CONVERTED TO NUMERIC FORM
DIGI - INTEGER HOLDING THE CONVERTED DIGIT
ISITD - LOGICAL FLAG INDICATING WHETHER OR NOT THE VALUE IN CHAR WAS AN ASCII DIGIT

FILES USED: NONE

COMMON STORAGE : NONE

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
TS TAKES A CHARACTER, DETERMINES IF IT IS A NUMERIC CHARACTER AND IF SO IT CONVERTS THE ASCII CHARACTER INTO A CORRESPONDING INTEGER.
DIGITS

START

IS CHAR AN ASCII DIGIT?

NO

YES

SET LOGICAL FLAG INDICATING CHAR IS A DIGIT
CONVERT ASCII CODE TO AN INTEGER DIGIT

RETURN

SET LOGICAL FLAG INDICATING CHAR IS NOT A DIGIT
PROGRAM NAME — ERROR MESSAGE PRINTING ROUTINE (ERRMES)

PROGRAMMER — ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :
LEX
GETDIG

CALLS : NONE

FILES USED:
UNIT# NAME
12 SOURCE LISTING

COMMON STORAGE :
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/HEADIN/ IDATE(3), ITIME(3), PAGENO, NUMOUT

INPUT/OUTPUT OPERATIONS : OUTPUT ERROR MESSAGES TO SOURCE LISTING

PURPOSE :
MES PRINTS ERROR MESSAGES CORRESPONDING TO THE "ERROR" CODE PASSED TO THE SUBROUTINE THROUGH COMMON.
START

IS ERROR NUMBER IN RANGE?

YES
WRITE APPROPRIATE ERROR MESSAGE BASED ON VALUE OF ERROR

NO
OUTPUT "ERROR NUMBER IS OUT OF RANGE"

RETURN

RETURN
PROGRAM NAME - GET DIGITS (GETDIG)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :
SPLIT

CALLS :
DIGITS
ERRMES

PARAMETERS :
RSIGN - REAL VALUE CONTAINING THE SIGN OF
THE VALUE BEING CONVERTED (+1.0 OR -1.0)
TOKBUF - INTEGER ARRAY CONTAINING THE TOKEN
TOKPT - INTEGER POINTER TO CURRENT CHARACTER
IN THE TOKEN BUFFER
TOKSIZ - SIZE OF THE TOKEN IN TOKBUF
INUMB - INTEGER PART OF THE NUMBER BEING CONVERTED
RNUMB - REAL VALUE CONTAINING THE INTEGER AND
FRACTIONAL PART OF THE NUMBER BEING
CONVERTED

FILES USED: NONE

COMMON STORAGE :
/ERRS/ ERROR, LINOLD

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
VERT ASCII DIGIT STRING TO BINARY FORM
GETDIG

C

CONVERT THE FRACTIONAL PART OF THE ASCII STRING INTO DINT.

DO MANY DIGITS?

NO

B

SET INTEGER NUMBER TO LEFT PORTION (ENTIRE) OF NUM.C1.

SET REAL NUMBER EQUAL TO LEFT AND RIGHT (ENTIRE AND FRACTIONAL) PORTION OF NUM.C1.

RETURN

REMOVE EXCESS DITS FROM BUFFER.

FAILING

PRINT USAGE MESSAGE.
PROGRAM NAME - GET LINE OF SOURCE INPUT

PROGRAMMER - ELAINE STRONG ACREE  
ENGINEERING EXPERIMENT STATION  
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM - DATA GENERAL ECLIPSE  
MODULE - LEXICAL ANALYZER

CALLED BY -  
CLAUSE  
STRING  
STRIP  
GETTOK

CALLS - NONE

PARAMETERS - EOF - LOGICAL FLAG INDICATING IF END-OF-FILE HAS BEEN REACHED

FILES USED:  
UNIT# NAME  
3 SOURCE INPUT

COMMON STORAGE -  
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM  
/ERROR/ ERROR, LINOLD  
/HEADIN/ IDATE(3), ITIME(3), PAGENO, NUMOUT  
/SLIST/ LISTIN  
/TSPEC/ TSN0(5)

INPUT/OUTPUT OPERATIONS - INPUT LINE OF SOURCE FILE

PURPOSE - GETLIN READS ONE LINE OF SOURCE OUTPUT AND IF REQUESTED, PRODUCES ONE LINE OF SOURCE LISTING.

******************************************************************************
GETLIN

START

CLEAR LINE BUFFER WITH BLANKS
SET LINE BUFFER POINTER TO 0

READ A LINE FROM THE SOURCE FILE INTO THE LINE BUFFER

END OF FILE?
YES - SET EOF = TRUE
NO

INPUT ERROR?
YES - SET ERROR
NO

SOURCE LISTING REQUESTED?
YES - OUTPUT LINE OF SOURCE TO LISTING FILE
NO

RETURN

RETURN

RETURN
PROGRAM NAME - GET TOKEN FROM SOURCE INPUT LINE BUFFER

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :
LEX
SMATCH
CLAUSE
UNITS

CALLS : GETLIN

PARAMETERS : EOF - LOGICAL FLAG INDICATING WHETHER OR NOT END-OF-FILE HAS BEEN REACHED

FILES USED: NONE

COMMON STORAGE :
/LINE/ LINBUF(132),LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/QOUTED/ QOUTE, APOSTR, BLANK
/PARENS/ LEFT, RIGHT

PURPOSE:
CTOK REMOVES ONE TOKEN AT A TIME FROM THE LINE BUFFER.
A TOKEN IS THEN PLACED ONE CHARACTER TO A WORD IN TOKBUF.
CTOK CALLS STRIP TO REMOVE COMMENTS FROM THE LINE WHEN A LEFT PARENTHESIS IS ENCOUNTERED.
SEPARATORS INCLUDE BLANKS, COMMAS, AND LEFT PARENTHESIS.

*************************************************************************
A

INCREMENT TOKEN BUFFER POINTER SIZE INSERT CHAR

CHAR = TO QUOTE MARK AND IT IS THE FIRST CHAR?

YES
RETURN

NO

INCREMENT LINE BUFFER POINTER OUT NEXT CHAR FROM LINE CHAR.

IS THERE A LEFT PARENTHESI, COMMA OR A BLANK?

YES
RETURN

CHECK FOR END OF TOKEN
PROGRAM NAME - LEXICAL ANALYZER (LEX) FOR THE ATP COMPILER

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

MODULE: LEXICAL ANALYZER
CALLED BY: PARSER

CALLS:
SMATCH
CLAUSE
GETTOK
SEARCH
ERRMES

PARAMETERS: EOF - LOGICAL FLAG INDICATING IF END OF FILE HAS BEEN REACHED

FILES USED: NONE

COMMON STORAGE:
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/STRNGY/ NUMWRD, SBUFER(40)
/NUMBER/ RNUMB, INUMB, TRNUMB, TINUMB
/CODES/ TCODES, TVALUE
/TABLES/ TKTAB(100,8), NUMTOK, TOKWID, TOKCOD(100,3)
/SECTS/ NBEGIN, NEND
/PUSHED/ PUSH

INPUT/OUTPUT OPERATIONS: NONE

PURPOSE:
LEXICAL ANALYZER SUBROUTINE CALLED BY THE PARSER. EX
RETURNS THE LOGICAL VALUE EOF INDICATING END OF FILE. HE
LEXICAL ANALYZER CONTROLS THE CALLING SEQUENCE TO OTHER
MODULES OF THE COMPILER WHICH REMOVE TOKENS FROM THE INPUT
SOURCE FILE AND RETURN NUMERIC CODES TO INDICATE THE TOKEN FOUND.
3 MATCH

USE SPECIAL MATCH TO IDENTIFY TOKEN

TOKEN FOUND? NO

ERROR? YES

SET TOKEN CODES AND VALUE TO INDICATE PIN-10

WRITE ERROR MESSAGE

RETURN
PROGRAM NAME - LEXICAL ANALYZER INITIALIZATION ROUTINE (LEXINT)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY : MAIN DRIVER OF THE ATP COMPILER

CALLS : NONE

PARAMETERS : NONE

FILES USED:
UNIT# NAME
2 TOKENFILE

COMMON STORAGE :
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/STRNGY/ NUMWRD, SBUFER(40)
/NUMBER/ RNUMB, INUMB, TRNUMB, TINUMB
/CODES/ TCODES, TVAULCE
/TABLES/ TOKTAB(100,8), NUMTOK, TOKWID, TOKCOD(100,3)
/SECTS/ NBEGIN, NEND
/PUSHED/ PUSH, LINPUSH
/QOUTED/ QOUTE, APOSTR, BLANK
/PARENS/ LEFT, RIGHT
/HEADIN/ IDATE(3), ITIME(3), PAGENO, NUMOUT
/VARTAB/ VARI(50), NUMVAR
/CONHEX/ HEXLST(16)
/ERRKNT/ NUMERR

INPUT/OUTPUT OPERATIONS : INPUT OF TOKEN TABLES

PURPOSE:
LEXINT INITIALIZES TABLES, POINTERS AND VARIABLES FOR THE LEXICAL ANALYZER. LEXINT SHOULD BE CALLED ONCE DURING THE INITIALIZATION PHASE OF THE COMPILER.

**************************************************************************************************
START

INITIALIZE
VAR FIELDS

INPUT TOKEN
TABLE
FROM
FILES

DATE

GET DATE
FROM THE
SYSTEM

TIME

GET TIME
FROM THE
SYSTEM

RETURN
PROGRAM NAME - LEXICAL ANALYZER VERSION TWO (LEX2)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY : PARSER

CALLS : GETTOK SEARCH GETLIN

PARAMETERS: EOF - LOGICAL FLAG INDICATING END OF FILE

FILES USED:

UNIT# NAME
13 P-CODE FILE (OUTPUT)

COMMON STORAGE :
/LINE/ LINBUF(132),LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/CODES/ TCODES, TVALUE
/TABLES/ TOKTAB(100,8), NUMTOK, TOKWID, TOKCOD(100,3)

INPUT/OUTPUT OPERATIONS : INSERTS INLINE P-CODE INTO OUTPUT FILE

PURPOSE :
LEX2 IS THE VERSION OF THE LEXICAL ANALYZER CALLED BY THE PARSER
to handle in line P-CODE. LEX2 reads a line of code and looks
for a $STOP-P token. If the $STOP-P is found the logical flag
'DONE' is set to true. If the $STOP-P token is not found on the
current line the first 120 characters of the input line are output
to the generated P-CODE output file. LEX2 then continues reading and
processing lines of P-CODE until $STOP-P or EOF is encountered.
PROGRAM NAME — PUSH TOKEN (PSHTOK)

PROGRAMMER — ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER / PARSER

CALLED BY : PARSE

CALLS : NONE

PARAMETERS : NONE

FILES USED : NONE

COMMON STORAGE :
/ERRS/ ERROR, LINOLD
/PUSHED/ PUSH, LINPSH

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE: THIS SUBROUTINE SETS THE VALUE OF PUSHED TO TRUE SO THAT,
LEX WILL RETURN ITS CURRENT VALUES AND WILL NOT ATTEMPT TO
READ A NEW TOKEN
START

SAVE THE "PUSHED" LINE NUMBER IN LINGOLD

SET PUSH = TRUE

RETURN
PROGRAM NAME — BINARY SEARCH (SEARCH)

PROGRAMMER — ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :

LEX UNITS

CALLS : NONE

PARAMETERS :

I — INTEGER ROW ARRAY DIMENSION OF TABLE
J — INTEGER COLUMN ARRAY DIMENSION OF TABLE
K — INTEGER ARRAY DIMENSION OF TOKEN
TABSIZ — INTEGER ACTUAL NUMBER OF ITEMS IN TABLE
TOKWID — INTEGER ACTUAL WIDTH OF TOKEN ARRAY
TABLE — INTEGER ARRAY CONTAINING ITEMS TO BE
SEARCHED FOR A MATCH TO TOKEN
TOKEN — INTEGER ARRAY CONTAINING ITEM TO BE
SEARCHED FOR IN THE TABLE
INDEX — INTEGER ROW SUBSCRIPT OF TABLE ROW
THAT MATCHES THE TOKEN
FOUND — LOGICAL FLAG INDICATING IF THE TOKEN
WAS FOUND IN THE TABLE

FILES USED: NONE

COMMON STORAGE : NONE

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :

ARCH PERFORMS A BINARY SEARCH USING THE CONTENTS OF TOKEN
AND THE CONTENTS OF TABLE TRYING TO FIND A MATCH.

THE AND TOKEN ARE DYNAMICALLY DIMENSIONED TABLE(I,J) AND TOKEN(K)
WITH THE SEARCH BEING PERFORMED ON THE ROW ENTRIES OF TABLE, TRYING
TO MATCH THE COLUMNS OF THE SELECTED TABLE ROW WITH THE CONTENTS
OF TOKEN.

ACTUAL ROW SIZE OF THE TABLE SEARCHED (ACTUAL NUMBER OF ENTRIES)
IS GIVEN BY TABSIZ. THE ACTUAL COLUMN WIDTH OF THE TABLE
SEARCHED IS GIVEN BY TOKWID.

A MATCH IS MADE THEN THE LOGICAL FLAG FOUND IS SET AND THE ROW
INDEX OF THE TABLE ENTRY IS RETURNED.

**********************************************************************
SEARCH

START

INITIALIZE VARIABLES

REPEAT BINARY SEARCH UNTIL ITEM IS FOUND OR THE SEARCH TABLE HAS BEEN EXHAUSTED

ITEM FOUND IN TABLE?

NO

RETURN

YES

SET "INDEX" TO VALUE OF "CURRENT" POINTER
PROGRAM NAME - SPECIAL MATCH (SMATCH)

PROGRAMMER - ELAINE STRONG ACRE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :
LEX

CALLS :
STRING
SPLIT
UNITS
DIGITS

PARAMETERS : FOUND - LOGICAL FLAG INDICATING WHETHER OR NOT THE TOKEN HAS BEEN MATCHED

FILES USED: NONE

COMMON STORAGE :
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/STRNGY/ NUMCHR, SBUFER(40)
/CODES/ TCODES, TVALUE
/QOUTED/ QOUTE, APOSTR, BLANK
/NUMBER/ RNUMB, INUMB, TRNUMB, TINUMB

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
SMATCH HANDLES THE IDENTIFICATION OF TOKENS NOT FOUND IN THE TOKEN TABLE SUCH AS PIN-ID'S, CHARACTER STRINGS, AND NUMBERS. A TOKEN WHICH HAS NOT BEEN FOUND IN THE TOKEN TABLE, AND DOES NOT BEGIN WITH A QUOTE MARK, APOSTROPE, OR NUMERIC SYMBOL IS CONSIDERED A PIN-ID.

**************************************************************************
START

INITIALIZE VARIABLES
REMOVE FIRST CHAR FROM TOKEN BUFFER

CHAR IS QUOTE OR CHAR IS punctuation?
NO

CHAR IS DIGIT?
YES

SET TOKEN CODE AND VALUE TO INDICATE A PIN-10
PACK PIN-10 INTO THE FIRST FIVE WORDS OF THE STRING BUFFER
RETURN

SET TOKEN CODE AND VALUE TO INDICATE A PIN-10
SET TANCH CODES AND VALUE
SET FLAGS
RETURN

A page 2

STRING
PUT CHARACTER STRING IN STRING BUFFER
PROGRAM NAME - SPLIT NUMBER OUT OF TOKEN (SPLIT)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY :

SMATCH

CALLS : GETDIG

PARAMETERS :

FLAG - LOGICAL FLAG INDICATING WHETHER OR NOT
THE TOKEN CONTAINS THE TOLERANCE SYMBOLS
+- (PLUS MINUS)

RNUMB - REAL VALUE CONTAINING THE NUMERIC FORM
OF THE ASCII STRING NUMBER

INUMB - INTEGER VALUE CONTAINING THE NUMERIC
OF THE ASCII STRING NUMBER (INTEGER
PORTION ONLY)

FILES USED: NONE

COMMON STORAGE :
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
LIT TAKES THE NUMBER IN THE TOKEN BUFFER AND "SPLITS" IT OUT OF
THE TOKEN BUFFER.
LIT DETERMINES THE SIGN OF THE NUMBER AND THEN CALLS GETDIG TO
GET THE DIGIT STRING.
PROGRAM NAME - CHARACTER STRING HANDLING ROUTINE (STRING)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY : SMATCH

CALLS : GETLIN

PARAMETERS : EOF - LOGICAL FLAG INDICATING WHETHER OR NOT AN END OF FILE AS BEEN REACHED IN THE SOURCE FILE

FILES USED:
NONE

COMMON STORAGE :
/LINE/ LINBUF(132),LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/STRNGY/ NUMCHR, SBUFER(40)
/CODES/ TCODES, TVALUE
/QOUTED/ QOUTE, APOSTR, BLANK

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE:
RING TAKES INPUT CHARACTER STRINGS, ENCLOSED IN QUOTES OR APOSTROPHES, REMOVES THEM FROM THE LINE BUFFER AND PACKS THEM TWO CHARACTERS TO A WORD INTO THE STRING BUFFER ( SBUFER ).
PROGRAM NAME - STRIP COMMENTS FROM SOURCE (STRIP)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

MODULE - LEXICAL ANALYZER

CALLED BY - GETTOK

CALLS - GETLIN

PARAMETERS - EOF - LOGICAL FLAG INDICATING WHETHER OR NOT AN END OF FILE HAS BEEN REACHED

FILES USED: NONE

COMMON STORAGE:
/LINE/ LINBUF(132),LINSIZ, LINBPT, LINNUM
/ERRS/ ERROR, LINOLD
/PARENS/ LEFT, RIGHT

INPUT/OUTPUT OPERATIONS: NONE

PURPOSE:
STRIP REMOVES THE COMMENTS AS THEY ARE FOUND IN THE SOURCE FILE BY GETTOK.
START

INITIALIZE VARIABLES

REPEAT UNTIL ALL RIGHT AND LEFT PARENTHESES MATCH

COMMENT TOO LONG?

SET ERROR FLAG

RETURN

LINE Buffer EMPTY?

GETLINE

FILL LINE BUFFER

EOF?

RETURN

INCREMENT LINK BUFFER POINTER

GET NEXT CHARACTER

CHAR = PARENTHESIS

INCREMENT APPROPRIATE PARENTHESES COUNTER
INCREMENT CHARACTER COUNT

RETURN
PROGRAM NAME - UNIT FINDING ROUTINE (UNITS)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : LEXICAL ANALYZER

CALLED BY : SMATCH

CALLS : SEARCH
GETTOK

PARAMETERS : FOUND - LOGICAL FLAG INDICATING WHETHER OR NOT THE UNITS HAVE BEEN FOUND

FILES USED: NONE

COMMON STORAGE :
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/ ERROR, LINOLD
/CODES/ TCODES, TVALUE
/TABLES/ TOKTAB(100,8), NUMTOK, TOKWID, TOKCOD(100,3)

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE:
ITS DETERMINES IF A NUMBER HAS UNITS (VOLTS, SECONDS ETC.) FOLLOWING IT AND SETS THE TOKEN CODE ACCORDING TO THE UNITS FOUND (IF ANY)

*************************************************************************

*************************************************************************
A

\(\text{IS \ TOKEN \ A \ UNIT \ TOKEN?} \begin{array}{c} \text{YES} \\ \text{NO} \end{array}\)

- Adjacent line buffer pointer to start of token
- Set token code and value to indicate a unitless number

RETURN

RETURN
III. COMPILER UTILITY ROUTINES

1. CONVRT
2. INSERT
3. OUDUMP
4. OUTCH
5. OUTINT
6. OUTLAB
7. OUTNUM
8. OUTSTR
9. OUTTAB
10. SEMERR
11. SETSTR
12. SYNERR
PROGRAM NAME - CONVERT (CONVRT)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY : OUTNUM
OUTINT

CALLS : NONE

PARAMETERS :

LEFT - INTEGER NUMBER TO BE CONVERTED
NUMBER - INTEGER HOLDING THE CURRENT POWER OF TEN
CHAR - INTEGER VALUE HOLDING THE ASCII CHARACTER
      WHICH HAS BEEN CONVERTED FROM LEFT.

FILES USED : NONE

COMMON STORAGE :
/CONHEX/ CHNUM(16)

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :

CONVRT TAKES THE INTEGER VALUE LEFT AND THE INTEGER VALUE NUMBER
AND RETURNS THE INTEGER CHAR WHICH CONTAINS A CHARACTER INDICATING
THE COUNT OF HOW MANY TIMES NUMBER CAN BE SUBTRACTED FROM LEFT.

CONVRT IS USED TO CONVERT LEFT FROM AN INTEGER VALUE INTO AN ASCII
CHARACTER STRING BY DETERMINING THE VALUE AT EACH DECIMAL POINT.
NUMBER WILL TAKE ON VALUES THAT ARE MULTIPLES OF 10.

******************************************************************************
START

INITIALIZE VARIABLES (COUNT = 0)

WHILE THE INTEGRAL PORTION OF THE NUMBER LEFT POWER OF 10

INCREMENT COUNT SO THAT IT WILL POINT TO THE CORRECT ARRAY ELEMENT

INCREMENT COUNT IF CHAR FOUND IN ARRAY MATCHES

RETURN
PROGRAM NAME - INSERT (INSERT)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY : OUTNUM
            OUTINT

CALLS : ODUMP

PARAMETERS :

CHAR1 - INTEGER CONTAINING UPPER BYTE OF
ASCII WORD TO BE INSERTED IN THE
OUTPUT BUFFER

CHAR2 - INTEGER CONTAINING LOWER BYTE OF
THE WORD TO BE INSERTED

FILES USED: NONE

COMMON STORAGE :
/OUTPUT/ OPTR, OUTBUF(60)

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :

INSERT TAKES CHAR1 AND CHAR2 AND PACKS THEM INTO ONE WORD OF
OUTBUF WITH CHAR1 IN THE HIGH ORDER BYTE AND CHAR2 IN THE
LOW ORDER BYTE.

**************************************************************************
PROGRAM NAME — DUMP OUTPUT BUFFER (ODUMP)

PROGRAMMER — ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY :
ACTION
PARSE
SETSTR
OUTSTR
INSERT
OUTCH
OUTTAB

CALLS : NONE

FILES USED:
UNIT# NAME
13 P-CODE FILE (OUTPUT)

COMMON STORAGE :
/OUTPUT/ OPTR, OUTBUF(60)

INPUT/OUTPUT OPERATIONS : OUTPUT OUTBUF TO P-CODE FILE

PURPOSE :
DUMP THE OUTPUT BUFFER TO THE GENERATED P-CODE OUTPUT FILE.
START

OUTPUT LINE EMPTY?

NO

WRITE OUTPUT CLEAR

RESET OUTPUT BUFFER CONTENT TO BLANK

CLEAR OUTPUT BUFFER WITH BLANKS

RETURN
PROGRAM NAME - OUTPUT CHARACTER (OUTCH)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY : OUTNUM
ACTION
PARSE

CALLS : NONE

PARAMETERS : CHAR - INTEGER CONTAINING CHARACTER TO BE INSERTED INTO THE OUTPUT BUFFER

FILES USED : NONE

COMMON STORAGE : /OUTPUT/ OPTR, OUTBUF(60)

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
PLACE IN THE OUTPUT BUFFER THE WORD CHAR CONTAINING TWO CHARACTERS.
OUTCH

START

IS LINE BUFFER FULL?

YES

OUTPUT LINE BUFFER

NO

INCREMENT OUTPUT BUFFER POINTER

INSERT CHAR INTO OUTPUT BUFFER

RETURN
**OUTINT** takes an integer "VALUE" and converts it into an ASCII string using the utility routines INSERT and CONVRT.
OUTINT

START

INITIALIZE VARIABLES

SET THE INTERNAL PORTION OF THE VALUE

SET THE SIGN OF THE VALUE

CONVERT THE NUMBER TO AN ASCII STRING

IS THIS NUMBER A LABEL?

NO

INSERT CONTENTS OF ASCII STRING INTO THE OUTPUT BUFFER

RETURN

INSERT COLON IMMEDIATELY FOLLOWING THE NUMBER
**PROGRAM NAME — OUTPUT A LABEL (OUTLAB, NOTE: THIS PROGRAM IS IN A FILE CALLED OUTLB.FR TO AVOID CONFUSION WITH AN ASSEMBLER ROUTINE OF THE SAME NAME.)**

**PROGRAMMER — ELAINE STRONG ACREE**
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

**COMPUTER SYSTEM : DATA GENERAL ECLIPSE**

**MODULE : SEMANTIC**

**CALLED BY : ACTION**

**CALLS : OUTINT**

**PARAMETERS : IVALUE — INTEGER LABEL VALUE TO BE OUTPUT**

**FILES USED : NONE**

**COMMON STORAGE :**

/LABTST/ LABEL

**INPUT/OUTPUT OPERATIONS :**

**NONE**

**PURPOSE :**

THIS SUBROUTINE SETS THE LABEL FLAG TO TRUE TO INDICATE THAT TO OUTINT THAT A LABEL IS TO BE OUTPUT. OUTINT WILL THEN INSERT A COLON IN TO THE OUTPUT BUFFER FOLLOWING THE LABEL.
OUTLAB - LEXICAL ANALYZER VERSION

START

SET LABEL FLAG TO TRUE TO INDICATE A LABEL IS TO BE OUTPUT

PRINT

PASS VALUE OF LABEL TO BE OUTPUT TO OUTPUT

RESET LABEL FLAG TO FALSE

RETURN
**PROGRAM NAME** - OUTPUT REAL NUMBER IN ASCII FORMAT (OUTNUM)

**PROGRAMMER** - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

**COMPUTER SYSTEM** : DATA GENERAL ECLIPSE

**MODULE** : SEMANTIC

**CALLED BY** :
  - ACTION
  - CHANGE

**CALLS** :
  - CONVRT
  - INSERT

**PARAMETERS** :
  - VALUE - REAL NUMBER CONTAINING THE VALUE TO CONVVERTED TO AN ASCII STRING

**FILES USED** : NONE

**COMMON STORAGE** :
  - /CONHEX/ CHNUM(16)

**INPUT/OUTPUT OPERATIONS** : NONE

**PURPOSE** :

OUTNUM TAKES A REAL "VALUE" AND CONVERTS IT INTO AN ASCII STRING USING THE UTILITY ROUTINES INSERT AND CONVRT.
PROGRAM NAME - OUTPUT STRING (OUTSTR)

PROGRAMMER - ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY : SETSTR
ACTION
CHANGE
GCON

CALLS : ODUMP

PARAMETERS :
STRING - INTEGER ARRAY CONTAINING THE STRING BUFFER TO BE INSERTED INTO THE OUTPUT BUFFER
NC4AR - INTEGER NUMBER OF CHARACTERS IN THE STRING BUFFER

FILES USED: NONE

COMMON STORAGE :
/OUTPUT/ OPTR, OUTBUF(40)

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
INSERT CONTENTS OF "STRING" INTO OUTPUT BUFFER.
START

RETURN

INPUT/OUTPUT
NUMBER OF WORDS TO BE OUTPUT FROM THE NUMBER OF CHARACTERS

II = 1

RETURN

13

IS OUTPUT BUFFER FULL?

WRITE OUTPUT BUFFER

INCREMENT OUTPUT BUFFER POINTER

INSERT ONE WORD OF STRING INTO ONE WORD OF OUTPUT BUFFER
PROGRAM NAME — OUTPUT A TAB (OUTTAB)

PROGRAMMER — ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY : SETSTR
ACTION
CHANGE

CALLS : ODUMP

PARAMETERS : NONE

FILES USED: NONE

COMMON STORAGE :
/OUTPUT/ OPTR, OUTBUF(60)

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :

OUTPUT A TAB OF UP TO TEN BLANK CHARACTERS TO THE OUTPUT BUFFER
OUTTAG

START

IS OUTPUT BUFFER FULL?

CALCULATE THE NUMBER OF SPACES TO TAB OVER

INSERT TABS INTO OUTPUT BUFFER

RETURN

OUTPUT BUFFER TO B.CIOSF FILE

YES
PROGRAM NAME - SUBROUTINE SEMERR (NERR)

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - CALLED WHENEVER SEMANTIC ERRORS OCCUR

CALLING SEQUENCE -
NERR - THE NUMBER OF THE SEMANTIC ERROR

CALLED BY - PARSE

CALLS - NOBODY

COMMON USED - HEADIN, ERRS

DESCRIPTION - SEMERR PRINTS AN ERROR MESSAGE INDICATING THE REASON A PARTICULAR PRODUCTION CANNOT BE COMPLETED. MOST OF THE ERRORS OCCUR BECAUSE OF INCORRECT OR MISSING HARDWARE DEFINITION DATA.
SEMERROR

PRINT SEMANTIC ERROR CODE

EXIT
PROGRAM NAME — SET STRING (SETSTR)

PROGRAMMER — ELAINE STRONG ACREE
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM : DATA GENERAL ECLIPSE

MODULE : SEMANTIC

CALLED BY : ACTION
GCON

CALLS : ODUMP
OUTTAB
OUTSTR

PARAMETERS :
STRING — INTEGER ARRAY USED AS A STRING BUFFER
CONTAINING CHARACTERS TO BE INSERTED IN
THE OUTPUT BUFFER
NCHAR — INTEGER NUMBER OF CHARACTERS IN THE
STRING BUFFER

FILES USED: NONE

COMMON STORAGE : NONE

INPUT/OUTPUT OPERATIONS : NONE

PURPOSE :
TAKE THE PACKED CHARACTER STRING IN ARRAY "STRING" DUMP THE
CONTENTS OF THE OUTPUT BUFFER, TAB OVER, AND OUTPUT THE STRING.

*****************************************************************************
Program Name: Syntax Error Message Routine (SYNERR)

Programmer: Elaine Strong Acree
Engineering Experiment Station
Georgia Institute of Technology

Computer System: Data General Eclipse

Module: Parser
Called by: Parse
Calls: None
Parameters: None

Files Used:

UNIT#  NAME
12     SOURCE LISTING

Common Storage:
/LINE/ LINBUF(132), LINSIZ, LINBPT, LINNUM
/TOKENS/ TOKBUF(35), TOKPT, TOKSIZ
/ERRS/   ERROR, LINOLD
/HEADIN/ IDATE(3), ITIME(3), PAGENO, NUMOUT
/ERRKNT/ NUMERR

Input/Output Operations: Output Syntax Error Message

Purpose:
print out "Syntax Error on line number# occurred while parsing token"
START

INCREMENT ERROR COUNT AND LINE COUNT

OUTPUT SYNTAX ERROR MESSAGE TO

RETURN
IV. PARSING AND CODE GENERATING Routines

1. ACTION
2. CHANGE
3. DADJ
4. DISCON
5. DMEAS
6. DMONV
7. DSET
8. GCHECK
9. GCON
10. INIT
11. PARSE
12. STARTS
13. TRANS
14. UPRINT
PROGRAM NAME - SUBROUTINE ACTION (NUMB)

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - TO COMPLETE ACTIONS ASSOCIATED WITH STATE CHANGES
AND SEMANTIC ANALYSIS

CALLING SEQUENCE -
NUMB - NUMBER OF ACTION TO COMPLETE

CALLED BY - PARSE

CALLS - SETSTR, OUTSTR, OUTCH, OUTNUM, OUTINT, OUTTAB
DADJ, SEMERR, DSET, PLOOK, LOCPIN, LOCDS, LEX2
DISCON, DMEAS, ODUMP, DMONV, UPRINT, CHANGE, TRANS,
OUTLAB

COMMON USED - DPARSE, NUMBER, TOKENS, CODES, STRNGY, SECTS, PINS

DESCRIPTION - ACTION IS CALLED BY THE PARSER TO COMPLETE
ACTIONS REQUIRED DURING STATE CHANGES
AND SEMANTIC ANALYSIS. ACTION PROVIDES A
FACILITY FOR SAVING INTERMEDIATE DATA WHILE
A PRODUCTION IS BEING PARSED. THESE SEMANTIC
DATA ARE STORED IN COMMON /DPARSE/ AT
APPROPRIATE STATES IN THE PARSE OF PAR-
TICULAR PRODUCTIONS. WHEN A FULL PRODUCTION
IS RECOGNIZED PARSE CALLS ACTION TO COMPLETE
SEMANTIC ANALYSIS AND P-CODE GENERATION.
PRODUCTION BEGIN
OUTPUT THE START SECTION AND SUBSECTION P-CODE

NBEGIN = 1

NO

OUTPUT SUBSECTION
EXIT

PRODUCTION ADJ6
ADJUST SIGNAL AT <PIN1> BY <STEP> PER <TIME> SEC IN A <SIGN> DIRECTION UNTIL THE VOLTAGE AT <PIN2> CHANGES TO:
MEASURE \( V + \) \( V \)

DADJ
EXIT

PRODUCTION ADJ7
ADJUST SIGNAL AT <PIN1> UNTIL THE VOLTAGE AT <PIN2> IS MEASURE \( V + \) \( V \)

DADJ
EXIT

PRODUCTION ADJ8
ADJUST SYNCHRO TRANSMITTER BY <STEP> PER <TIME> SEC UNTIL THE VOLTAGE AT <PIN1> IS:
MEASURE \( V + \) \( V \)

DADJ
EXIT

PRODUCTION SET16
SET THE SYNCHRO TRANSMITTER TO - DEGREES + - DEGREES [CLOCKWISE/ COUNTERCLOCKWISE]

GENERATE SETDV P-CODE
EXIT
PRODUCTION TRANS [A]
MONITOR [THE]
TRANSITION AT <PIN> FROM
-V+ -V TO
-V- +V AT
TIME __+__

PRODUCTION
MON [THE]
VOLTAGE AT
<PIN> OF -V
+ -V FOR __ S

TRANS
EXIT

DMONV
EXIT
PROGRAM NAME - SUBROUTINE CHANGE

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - TO GENERATE CHANGE P-CODE

CALLING SEQUENCE - NO PARAMETERS

CALLED BY - ACTION

CALLS - OUTCH, OUTTAB, PLOOK, SELGN, OUTINT, OUTNUM, OUTSTR

COMMON USED - PINS, DPARSE

DESCRIPTION - USED TO IMPLEMENT THE CHANGE P-CODE.
    THIS PRODUCTION LOCATES THE DESIRED TEST PIN AND THEN SETS UP TO CHECK THE VOLTAGE AT THAT PIN. A REGISTER REFERENCE IS OUTPUT TO INDICATE THE REFERENCE VALUE FOR THE CHANGE. APPROPRIATE SCALE FACTORS, TOLERANCES, AND CONNECTIONS ARE SUPPLIED IN ACCORDANCE WITH THE PARAMETERS OF THE CHANGE P-CODE.
CHANGE

LOOK UP PIN 10

GET READY TO MONITOR VOLTAGE

GET A TOLERANCE

SELECT AN A/D GAIN

GENERATE A CHANGE P-CODE

EXIT
PROGRAM NAME - SUBROUTINE DADJ (PIN1, PIN2, STEP, DIREC,  
PHASE, VAL, TOL, CTYPE, TIME)

PROGRAMMER - C.A. BATTERMAN  
ENGINEERING EXPERIMENT STATION  
GEORGIA TECH  
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - GENERATES THE 'ADJDEV' P-CODE

CALLING SEQUENCE -
PIN1 - USED TO FIND DAC TO ADJUST  
PIN2 - RESULTING ADC VALUE MEASURED HERE  
STEP - SIZE OF ADJUSTMENT STEP  
DIREC - POS (+1) OR NEG (-1) STEP DIRECTION  
PHASE - ANGLE FOR AC DAC AND D/S (SEE BELOW)  
VAL - DESIRED VOLTAGE VALUE AT ADC  
TOL - TOLERANCE OF SAME  
CTYPE - TYPE OF DEVICE TO ADJUST  
CTYPE = 'DS'  
CTYPE = 'AC' DAC  
CTYPE = 'DC' DAC  
TIME - TIME BETWEEN ADJUSTMENT STEPS

CALLED BY - ACTION

CALLS - SETSTR, OUTTAB, OUTSTR, LOCDAC, OUTCH, OUTINT, SELLIM  
GETTL, PLOOK, SELGN, OUTNUMC

COMMON USED - PINS

DESCRIPTION - DADJ IS A VERY GENERAL PURPOSE ROUTINE  
FOR GENERATING THE 'ADJDEV' P-CODE. IT  
WAS ORIGINALLY WRITTEN WHEN THERE WAS A LARGE  
NUMBER OF ATP STATEMENTS WHICH GENERATED  
THE 'ADJDEV' P-CODES. SINCE THEN ABOUT  
ten STATEMENTS WERE DELETED AND THE GENERALITY  
IS NOW PROBABLY NOT NEEDED. (THIS IS AN  
EXPLANATION FOR THE LARGE NUMBER OF CALLING  
PARAMETERS TO THIS SUBROUTINE) IN ANY CASE  
DADJ IS USED WHENEVER IT IS NECESSARY TO  
ADJUST A DEVICE USING DISCREET STEPS  
UNTIL A CERTAIN VOLTAGE IS MEASURED AT A  
TEST PIN. THE ROUTINE FIRST LOCATES THE  
DEVICE WHICH IS TO BE ADJUSTED, FINDS A DEVICE  
LIMIT VALUE, AND GETS A TOLERANCE IF NEEDED.  
SECONDLY, THE PIN USED TO MEASURE THE RESULTING  
VOLTAGE CHANGES IS FOUND, AN A/D GAIN IS SELECTED  
AND THE P-CODE PARAMETERS ARE FINALLY OUTPUT.
SELECT A TOLERANCE

SELECT THE A/D GAIN

GENERATE THE DADJ P-CODE

EXIT
PROGRAM NAME - SUBROUTINE DISCON

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - REMOVES CONNECTIONS FROM TEST PINS

CALLING SEQUENCE - NO PARAMETERS

CALLED BY - ACTION

CALLS - PLOOK, POPEN, GCON

COMMON USED - DPARSE

DESCRIPTION - 'DISCON' LOOKS UP THE PIN ID FOUND IN IDCHK(1,1). THIS IS THE FIRST PIN FOUND IN ALL OF THE REMOVE AND DISCONNECT ATP STATEMENTS. AN OPEN CIRCUIT ENTRY IS FOUND IN THE H/W DEFINITION TABLE AND APPROPRIATE CONNECTIONS ARE MADE TO ATTAIN THE OPEN CIRCUIT STATE.
PROGRAM NAME - DMEAS

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - GENERATE 'MEAS' P-CODE (FOR ADC)

CALLING SEQUENCE -
PIN - PIN ID FOR MEASURE
VAL - DESIRED VOLTAGE
TOL - TOLERANCE OF SAME

CALLED BY - ACTION

CALLS - PLOOK, LMEAS, GCON, SETSTR, OUTAB, GETTL, SELGN
OUTCH, OUTSTR, OUTINT, OUTNUM

COMMON USED - PINS

DESCRIPTION - USED TO IMPLEMENT THE 'MEAS' P-CODE.
THE PRODUCTION LOCATES THE DESIRED
TEST PIN, FINDS A MONITOR ID IN THE
H/W DEFINITION TABLE, SELECTS APPROPRIATE GAINS AND TOLERANCE AND THEN
GENERATES THE 'MEAS' P-CODE. USED TO
CHECK CURRENT ADC STATUS VALUES.
DMEAS

LOOK UP PIN ID

FIND H/W MEASURE ENTRY (LMEAS)

MAKE CONNECTIONS (GCON)

GET A TOLERANCE

SELECT THE A/O GAIN

GENERATE THE MEAS P-CODE

EXIT
PROGRAM NAME - SUBROUTINE DMONV (PIN, VAL, TOL, TIME)

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - GENERATE A MONITOR P-CODE

CALLING SEQUENCE -
PIN - PIN ID TO MONITOR
VAL - DESIRED VOLTAGE
TOL - TOLERANCE OF ABOVE
TIME - AMOUNT OF TIME TO MONITOR

CALLED BY - ACTION

CALLS - SETSTR, OUTTAB, PLOOK, GETTL, SELGN, OUTCH, OUTSTR, OUTNUM, OUTINT

COMMON USED - PINS

DESCRIPTION - USED TO IMPLEMENT THE MONITOR P-CODE.
THE PRODUCTION LOCATES THE DESIRED TEST PIN, MAKES ANY REQUIRED CONNECTIONS,
SELECTS THE REQUIRED GAIN AND TOLERANCE, AND THEN GENERATES THE 'MONV' P-CODE.
'MONV' IS USED WHENEVER A VOLTAGE MUST BE INSURED TO REMAIN WITHIN A DESIRED RANGE FOR A CERTAIN PERIOD OF TIME.
DMONV

LOOK UP PIN ID

GET READY TO MONITOR A VOLTAGE

GET A TOLERANCE

SELECT THE A/D GAIN

GENERATE THE MONV P-CODE

EXIT
PROGRAM NAME - SUBROUTINE DSET (PIN, VAL, TOL, CTYPE, PHASE, SWITCH)

PROGRAMMER - C.A. BATTERMAN

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - SET A DEVICE TO A SELECTED VALUE

CALLING SEQUENCE -
  PIN - PID ID TO SET UP
  VAL - DESIRED VALUE
  TOL - TOLERANCE OF ABOVE
  CTYPE - TYPE OF REQUIRED DEVICE
    CTYPE = 'AC'
    CTYPE = 'DC'
  PHASE - ANGLE FOR AC DAC (0, 9, 180, 189)
  SWITCH - .TRUE. IF CONNECTIONS ARE TO BE MADE

CALLED BY - ACTION

CALLS - PLOOK, LOCLD, GCON, GCHECK, SETSTR, OUTTAB, OUTCH, GETTL, OUTNUM

COMMON USED - PINS

DESCRIPTION - 'DSET' IS USED TO SET A/C AND D/C DACS
  TO REQUIRED VOLTAGES AND CONNECT THEM
  TO THE APPROPRIATE PIN. FOR DC DACS
  THE ROUTINE FIRST LOOKS FOR A VALID
  LOGIC DISCREET. IF A LOGIC CANNOT BE
  FOUND A DC DAC IS SELECTED. THE SETDEV
  P-CODE IS GENERATED WITH APPROPRIATE
  PARAMETERS. IF NEEDED NECESSARY SWITCH
  STATES ARE UPDATED. A 'CHECK' INSTRUCTION
  IS GENERATED TO INSURE CORRECT FUNCTIONING
  OF THE HARDWARE.
DSET

LOOK UP PIN ID

CAN A LOGIC DISCRETE BE USED?

YES

MAKE CONNECTIONS

NO

LOCATE A DAC

GET A TOLERANCE

GENERATE A SETDEV P-CODE

MAKE CONNECTIONS IF NECESSARY

CHECK APPLIED VOLTAGE (GCHECK)

EXIT

CHECK APPLIED VOLTAGE (GCHECK)

EXIT
PROGRAM NAME - SUBROUTINE GCHECK (VAL)

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GEORGIA 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - CALLED AFTER DSET TO GENERATE A CHECK

CALLING SEQUENCE -
VAL - DESIRED VALUE TO CHECK

CALLED BY - DSET

CALLS - SETSTR, OUTSTR, SELGN, OUTCH, OUTINT

COMMON USED - PINS

DESCRIPTION - GCHECK IS CALLED BY DSET TO GENERATE A 'CHECK' P-CODE TO INSURE THAT A DEVICE HAS BEEN SUCCESSFULLY SET TO THE DESIRED VOLTAGE. A/D INFO MUST BE ALREADY SET UP IN PINCOM. THE ROUTINE SELECTS AN APPROPRIATE GAIN AND DUMPS THE 'CHECK' P-CODE.
GCHECK

SELECT THE
GAIN FOR
A/D READING

GENERATE
A CHECK
P-CODE

EXIT
PROGRAM NAME - SUBROUTINE GCON

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - GENERATE P-CODE TO COMPLETE A PIN CONNECTION

CALLING SEQUENCE - NO CALLING PARAMETERS

CALLED BY -

CALLS - SETSTR, OUTSTR

COMMON USED - PINS

DESCRIPTION - 'OPEN' AND 'CLOSE' P-CODES ARE GENERATED TO COMPLETE THE CONNECTION DESCRIBED IN THE COMMON/PINS/. ANY NUMBER OF CONNECTIONS MAY BE MADE, AS DEFINED IN THE HARDWARE DEFINITION TABLES. COMMON /PINS/ MUST BE SET UP THROUGH CALLS TO THE H/W DEFINITION UTILITY ROUTINES !!
GCON

I = 1

SWITCH
POS(I) S/B OPEN?

YES

GENERATE
OPEN
P-CODE

NO

GENERATE
CLOSE
P-CODE

I = I + 1

YES

EXIT

I = NUMSW?

YES
PROGRAM NAME - INIT

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - TO INITIALIZE THE PARSE TABLES

CALLING SEQUENCE - NOT APPLICABLE

CALLED BY - NOBODY

CALLS - NOBODY

COMMON USED - DPARSE
PROGRAM NAME - SUBROUTINE PARSE

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

USAGE - TO PARSE THE ATP SYNTAX

CALLING SEQUENCE - NO PARAMETERS

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

CALLED BY - MAINATP

CALLS - STARTS, SYNERR, PSHTOK, ACTION, SEMERR, ODUMP

COMMON USED - DPARSE, NUMBER, TOKENS, CODES, STRNGY, SECTS, PINS

DESCRIPTION - 'PARSE' IS A FINITE STATE MACHINE USED TO PARSE THE ATP TEST LANGUAGE. THE ATP SYNTAX IS VERIFIED BY MAKING STATE TRANSITIONS USING THE PARSE TABLES. THE ATP PARSE TABLES SPECIFY THE ENTIRE SYNTAX OF THE ATP LANGUAGE. THE PARSER FIRST FINDS A VALID START STATE BY CALLING 'STARTS'. A TRANSITION TO THE NEXT STATE CAN BE MADE PROVIDING THE NEXT TOKEN FROM LEX HAS A ENTRY IN THE PARSE TABLES. THE SUBROUTINE ACTION IS CALLED TO SAVE ANY INTERMEDIATE DATA SUCH AS PID ID'S VOLTAGES, ETC. THAT ARE REQUIRED BY THE SEMANTIC ROUTINES. WHEN THE PARSER RECOGNIZES A VALID PRODUCTION ACTION IS CALLED TO COMPLETE THE SEMANTIC ANALYSIS AND CODE GENERATION. PRODUCTIONS MAY UTILIZE LOOKAHEAD IF REQUIRED.
IDCHK- IS PIN ID BUFFER WITH CURRENT ENTRY AT IDPTR
VALCHK- IS VALUE BUFFER WITH CURRENT ENTRY AT VALPTR
CSTATE- IS THE CURRENT PARSE STATE
NSTATE- SPECIFIES VALID TRANSITIONS TO THE NEXT
STATE FROM THE CURRENT STATE. THERE IS
A MAXIMUM OF FIVE POSSIBLE NEXT STATES
FOR EACH CURRENT STATE.

IF NSTATE = -1 THEN PARSER IS AT EOS (END OF STATEMENT)
     AND THE PRODUCTION IN ACTTBL IS COMPLETED
IF NSTATE = -2 THEN PARSER HAS FOUND LOOKAHEAD PRODUCTION
     AND SHOULD PUSH CURRENT LEX INPUT BACK

TOKTBL- ASSOCIATES THE STATES OF NSTATE WITH TOKENS.
         EACH OF THE POSSIBLE STATES IS ENTERED ONLY
         WHEN THE CORRECT TOKEN IS RECEIVED FROM LEX.

ACTTBL- ASSOCIATES THE STATES OF NSTATE WITH POSSIBLE
         ACTIONS. ACTIONS ARE HANDLED BY SUBROUTINE ACTION.

SEMFLG -0 IF NO SEMANTIC ERRORS HAVE OCCURRED
         1 INVALID PIN-1 ID REFERENCE
         2 INVALID PIN-2 ID REFERENCE
         3 INVALID PIN-3 ID REFERENCE
         4 UNABLE TO SET A DAC TO REQUESTED VALUE
         5 CANNOT MEASURE DESIRED VALUE WITH A/D
         6 UNABLE TO SET UP D/S TO REQUESTED VALUE
         7 USER MUST SUPPLY TIME TOLERANCE
         8 INVALID PARSE TABLE ENTRY DETECTED
         9 INVALID DATA IN H/W DEFINITION TABLE

THE FOLLOWING DEFINES COMMON /DPARSE/ :

REAL VALCHK
LOGICAL SEMFLG
INTEGER IDCHK, IDPTR, VALPTR, CSTATE, NSTATE
INTEGER TOKTBL, ACTTBL, NSPTR, ILABEL
COMMON /DPARSE/ IDCHK(5,10), VALCHK(10), IDPTR, VALPTR,
     NSTATE(5,240), TOKTBL(240), ACTTBL(5,240), CSTATE, NSPTR,
     SEMFLG, ILABEL
FOLLOWING ARE DEFINITIONS FOR DATA IN COMMON /PINCOM/ :

ADADDR - IS PINS ASSOCIATED A/D ADDRESS
CNPTR - IS POINTER INTO CONNECTION TABLE
NUMENT - IS NUMBER OF ENTRIES IN CONNECTION TABLE
NUMSW - IS NUMBER OF ENTRIES IN ATTRIB TABLE
ATTRIB - CONTAINS ASCII PIN IDENTIFIERS OF PIN
       CONNECTIONS NEEDED TO ACCOMPLISH THE
       CURRENTLY REQUESTED FUNCTION
POS - ASSOCIATED OPEN OR CLOSED POSITION OF SWITCHES
       SPECIFIED IN ATTRIB TABLE (1=CLOSED 0=OPEN)
SCALE - A/D SCALE FACTOR ASSOCIATED WITH THE
       CURRENTLY ACTIVE PIN CONNECTION FUNCTION
REFER - REFERENCE ID NUMBER FOR DAC OR DS FUNCTION

THE FOLLOWING DEFINES COMMON /PINCOM/ :

INTEGER ADADDR, CNPTR, NUMENT, NUMSW, ATTRIB
INTEGER POS, SCALE, REFER, SEMFLG

COMMON /PINS/ ADADDR (3), CNPTR, NUMENT, NUMSW, ATTRIB(2,10),
                 POS(10), SCALE, REFER, SEMFLG
PROGRAM NAME - SUBROUTINE STARTS (EOF)

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - TO OBTAIN A START STATE FOR THE PARSER

CALLING SEQUENCE -
EOF - TRUE IF END OF SOURCE FILE REACHED

CALLED BY - PARSE

CALLS - LEX, SYNERR

COMMON USED - DPARSE, NUMBER, TOKENS, CODES, STRNGY, SECTS

DESCRIPTION - 'STARTS' OBTAINS A START STATE TOKEN FROM THE
LEXICAL ANALYZER. A TOKEN IS CHECKED AGAINST
THE FIRST SET OF ENTRIES IN THE PARSE TABLES
FOR A VALID STARTING STATE.
STARTS

RESET PARSE FLAGS TO BEGIN A NEW PRODUCTION

GET A TOKEN FROM LEX

EOF?

YES

EXIT

NO

TOKEN A VALID START STATE?

YES

SET CSTATE

NO

PRINT SYNTAX ERROR

EXIT
PROGRAM NAME - SUBROUTINE TRANS

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

USAGE - GENERATES THE 'TRANS' P-CODE FOR VOLTAGE TRANSITIONS

CALLING SEQUENCE - NO PARAMETERS

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

CALLED BY - ACTION

CALLS - SETSTR, OUTTAB, PLOOK, GETTL, SELGN, OUTCH, OUTSTR
OUTINT, OUTNUM

COMMON USED - DPARSE, PINS

DESCRIPTION - 'TRANS' DOES THE SEMANTIC ANALYSIS AND
CODE GENERATION FOR CHECKING VOLTAGE
TRANSITIONS. FIRST, THE DESIRED TEST
PIN ID IS LOOKED UP IN THE H/W DEFINITION
TABLE AND A MONITOR CONNECTION IS MADE.
APPROPRIATE TOLERANCES ARE OBTAINED FROM
GETTL IF NEEDED, A GAIN IS SELECTED,
AND VALID TRANSITION TIME TOLERANCE IS
FOUND. THE 'TRANS' P-CODE IS GENERATED
IN ACCORDANCE WITH IT'S SPECIFICATION.
TRANS

LOOK UP PIN ID

SELECT A TOLERANCE FOR VOLTAGE 1

SELECT A TOLERANCE FOR VOLTAGE 2

GET A TIME TOLERANCE

SELECT THE A/D GAIN

GENERATE THE TRANS P-CODE

EXIT
PROGRAM NAME - SUBROUTINE UPRINT

PROGRAMMER - C.A. BATTERMAN
ENGINEERING EXPERIMENT STATION
GEORGIA TECH
ATLANTA, GA. 30332

COMPUTER SYSTEM - DATA GENERAL ECLIPSE

USAGE - GENERATES P-CODE TO PRINT LITERAL AND P MACHINE REGISTERS

CALL SEQUENCE - NO PARAMETERS

CALLED BY - ACTION

CALLS - LEX, SETSTR, OUTTAB, OUTSTR, OUTINT, PSHTOK

COMMON USED - NUMBER, TOKENS, CODES, STRNGY, SECTS, LCONST

DESCRIPTION - UPRINT BUILDS P-CODE TO OUTPUT STRINGS AND P MACHINE REGISTERS TO THE OPERATORS CONSOLE. THE 'OUTSTR' P-CODE IS USED FOR STRING OUTPUT. 'OUTDEC' IS GENERATED TO PRINT REGISTER VALUES. THE ROUTINE ALLOWS FREE FORMAT OF ATP SOURCE LINES SO THAT ALMOST ANY DESIRED SEQUENCE MAY BE OUTPUT TO THE USER CONSOLE. THE ';' CHARACTER IS USED TO GENERATE A CRLF SEQUENCE.
UPRINT

GET A TOKEN FROM LEX

IS TOKEN A STRING?

YES

GENERATE OUTSTR P-CODE

NO

IS TOKEN A REGISTER REFERENCE?

YES

GENERATE LOAD REGISTER P-CODE

NO

IS TOKEN A SEMICOLON?

YES

EXIT

NO

PUSH TOKEN BACK TO INPUT

EXIT
V. HARDWARE DEFINITION AND CONVERSION ROUTINES

1. RLDTBL S
2. DETAB
3. GETTL
4. HCONV
5. LMEAS
6. LOCDAC
7. LOCDS
8. LOCLD
9. LOCPIN
10. PLOOK
11. POPEN
12. SELGN
13. SELIM
14. SRCHCT
SUBROUTINE BLDTBLS

SYSTEM- DATA GENERAL ECLIPSE

PROGRAMMER- J. B. THOMPSON, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

PURPOSE-
THIS ROUTINE BUILDS THREE FILES USED BY THE ATP COMPILER.
ONE FILE (-.D3) IS USED TO CONTAIN INFORMATION FOR THE
COMMON BLOCKS CTBLS AND DACETC. THE OTHER TWO FILES ARE
RANDOM ACCESS FILES THAT ARE KEYED INTO BY THE COMPILER
USING THE COMMON BLOCK INFORMATION SUPPLIED BY THE -.D3
FILE. THE TWO FILES ARE THE CONNECTION FILE (-.D1) AND
THE PIN FILE (-.D2). THE COMMON BLOCK INFO SPECIFIES THE
PIN IDS AND A POINTER WHICH POINTS INTO THE CONNECTION FILE.

THE CONNECTION FILE (-.D1) HAS THE FOLLOWING FORMAT FOR
EACH PIN:
1ST RECORD - 1 WORD - ALWAYS 177777K
- 3 WORDS- ASCII A/D ADDRESS
- 1 WORD - NUMBER OF ENTRIES
  FOR THIS PIN

2ND - NUMENT RECORDS - 1 WORD - CONNECTION TYPE
- 2 WORDS- ASCII SWITCH ID
- 1 BYTE - COND. FACTOR INDEX
- 1 BYTE - SWITCH STATUS
- 1 WORD - CONNECTION VALUE

THE PIN ID FILE (-.D2) HAS THE FOLLOWING FORMAT:
1 - N RECORDS - 5 WORDS- ASCII PIN ID

THE FIRST WORD OF BOTH THE CONNECTION AND PIN ID FILE
SPECIFIES THE NUMBER OF 10 BYTE RECORDS IN THE FILES.

THE CONNECTION TYPE FIELD IN THE CONNECTION FILE SPECIFIES
THE TYPES OF CONNECTIONS THAT CAN BE MADE TO THE PIN. IT
HAS THE FOLLOWING VALUES:

=0 - CONTINUATION FOR PREVIOUS RECORDS (USED
  TO SPECIFY MORE SWITCH CLOSURES)
=1 - CONNECTION TO ANOTHER PIN. NOTE THAT
  THE ASSOCIATED CONNECTION VALUE IS AN INDEX
  INTO THE PIN FILE WHERE THE 2ND PIN'S ID
  IS STORED.
=2 - LOGIC DISCRETE (INCLUDING OPEN CIRCUITS)
=3 - DIGITAL/SYNCHRO
=4 - AC DAC
=5 - DC DAC
=6 - MONITOR CONNECTION
=7 - MANUAL CONNECTION

NOTE THAT THE PIN FILE IS INDEXED BY THE CONNECTION VALUE OF
CONNECTION TYPE 1.

CALLED TO CONVERT THE HARDWARE DEFINITION FILE
COMMON STORAGE:

1) CTBLs
   NMPINS - THE NUMBER OF PINS IN THE HARDWARE DEFINITION FILE
   PINTAB(500,5) - ASCII PIN IDENTIFIERS IN ASCENDING ORDER
   PTRTAB(500) - POINTERS INTO THE CONNECTION FILE FOR EACH PIN ID

2) DACETC
   DCLow(20)
   AND SO ON

EXTERNAL REFERENCES
SUBR FSEEK - D.G. SUPPLIED

*****************************************************************************
BUILD DAC TABLES

START

NODC = 0

I = 1

I ≠ I + 1?

RETURN

READ HARDWARE DEFINITION SOURCE FILE

JTYPE ≠ DC?

YES

NO

SEQUENTIAL ORDER?

YES

NO

ASSIGN DCYCLEL LOW(I), DCYCLEH HIGH(I), FROM READ

NODC = NODC + 1

NO

3

JTYPE = AC?

YES

NO

"OUT OF ORDER" ERROR

3

"OUT OF ORDER" ERROR

ASSIGN ACYCLEL(I = NODC), ACYCLEH(I = NODC), ACYCLEH(I = NODC)

ERROR
BUILD
GAIN TABLES

START

I = 1
I = I + 1

I > NORE
YES

READ
HARDWARE
DEFINITION
SOURCE FILE

GAIN # "in sequential order"
NO
YES

ASSIGN
GAIN(I)
GAINBAE(I)

RANGES IN
DECREASING ORDER
NO
YES

ERROR

RETURN
BUILD CONDITIONING FACTOR TABLES

START

\[ I = 1 \]

\[ I > \text{NOREC} \]

\[ I = I + 1 \]

READ HARDWARE DEFINITION SOURCE FILE

ASSIGN COND(I)

CONDFAC(I)

SEQUENTIAL ORDER

ERROR

RETURN

GET ANGLE TOLERANCE

START

READ HARDWARE DEFINITION SOURCE FILE

ASSIGN ANGLE TOLERANCE

RETURN
1. Build Pin Tables

2. Start

3. Read Hardware Definition Source File

4. EOF?
   - Yes: Write out info for previous pin, return
   - No: Next pin?

5. Next pin?
   - No: First pin?
     - No: Write out info for previous pin
     - Yes: Set pin's # of connections to zero

6. Insert pin #0 in proper position in table

7. Connection present?
   - No: Error message
   - Yes: Too many connections?
     - Yes: Encode and insert connection in table, increment # of connections
     - No: Write out info for previous pin

8. Error message

Flowchart diagram showing the process of building pin tables and handling connections.
NAME- SUBROUTINE DETAB (IERROR)

SYSTEM- DATA GENERAL ECLIPSE

PROGRAMMER- J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

PURPOSE- TO REMOVE TABS FROM THE HARDWARE DEFINITION FILE. NOTE
THAT THIS FILE WILL ALWAYS END WITH A '.DF' EXTENSION.
THE OUTPUT FILE WILL HAVE A '.DT' EXTENSION.

USAGE- RUN TO PRE-PROCESS A HARDWARE DEFINITION FILE

CALLING SEQUENCE-
IERROR - OUTPUT PARAMETER - SET TO TRUE IF AN ERROR OCCURS

COMMON STORAGE- NONE

EXTERNAL REFERENCES- NONE

FILES USED-
-.DF - INPUT - HARDWARE DEFINITION FILE WITH TABS
-.DT - OUTPUT - FILE WITH TABS REPLACED BY BLANKS

******************************************************************************
DETAB

START

READ ONE LINE OF INPUT FILE

END OF FILE?

NO

I/O ERROR?

YES

DISPLAY ERROR MESSAGE

REPLACE TAB CHARACTERS WITH BLANKS

WRITE THE DETABBED LINE TO OUTPUT FILE

YES

RETURN
PROGRAM NAME: GETTL

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: DMONV, DSET, DMEAS, TRANS, DADJ

CALLS: NONE

FILES USED: NONE

COMMON STORAGE: HWITBL (ANGTOL,ACVLTL(I),ACVLTH(I),ACDEFT(I),
DCVLTL(I),DCVLTH(I),DCDEFT(I))
STRS (ASCII STRINGS)

FUNCTION: THIS FUNCTION RETURNS THE DEFAULT TOLERANCES FOR THE
TYPE INDICATED BY ITYPE AND VAL. ITYPE CAN HAVE THE
VALUE "DS", "DC", OR "AC" INDICATING A D/S, DC DAC, OR
AC DAC RESPECTIVELY. IF A DAC IS INDICATED THEN VAL
ALSO HAS A VALUE WHICH IS USED TO DETERMINE THE RANGE
FOR THE DAC; THUS, THE CORRECT DEFAULT TOLERANCE
FOR THAT PARTICULAR VOLTAGE RANGE IS RETURNED. IF
ITYPE IS EQUAL TO "DS", THEN THE DEFAULT ANGLE
TOLERANCE IS RETURNED.
NAME- PROGRAM HCONV
SYSTEM- DATA GENERAL ECLIPSE
PROGRAMMER- J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

PURPOSE-
THIS PROGRAM CONVERTS THE SOURCE HARDWARE DEFINITION FILE
(.DF EXTENSION) INTO THREE FILES WHICH ARE USED BY THE
ATP COMPILER DIRECTLY. THE CONVERSION PROCESS CONSISTS
OF FIRST CREATING A TEMPORARY FILE OF THE SOURCE WITH
ALL TAB CHARACTERS REMOVED. THIS FILE IS THEN CONVERTED
TO THREE FILES IN A FORMAT SUITABLE FOR THE ATP COMPILER.

USAGE-
RUN TO CONVERT A HARDWARE DEFINITION FILE

COMMON STORAGE-
NONE

EXTERNAL REFERENCES-
SUBR DETAB
SUBR BLDIBLS

FILES USED-
-.DF - INPUT - HARDWARE DEFINITION FILE WITH TABS
-.DT - IN/OUT - FILE WITH TABS REPLACED BY BLANKS
-.D1 - OUTPUT - RANDOM ACCESS FILE CONTAINING THE
PIN CONNECTION INFORMATION
-.D2 - OUTPUT - RANDOM ACCESS FILE CONTAINING THE
THE PIN IDS OF PINS WHICH ARE CONNECTED
TO OTHER PINS
-.D3 - OUTPUT - SEQUENTIAL FILE WITH COMMON BLOCK
INFORMATION

*************************************************************
START

REQUEST FILENAME FROM USER

CHANGE FILE NAME EXTENSION TO NEW EXTENSION

DETAB

REMOVE THE CHARACTERS FROM SOURCE FILE

CREATE THE THREE NEW FILENAMES AND SET UP THE FILES

BLOTBYLS

BUILD THE THREE HARDWARE FILES

CLOSE ALL FILES

RETURN
PROGRAM NAME: LMEAS

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: DMEAS

CALLS: SRCHCT (TO SEARCH CONNECTION TABLES)

FILES USED: NONE

COMMON STORAGE: NONE

FUNCTION: THIS ROUTINE IS CALLED TO SEARCH FOR THE MONITOR
ENTRY FOR THE CURRENT PIN. IF A MONITOR ENTRY EXISTS
THEN ERRFLG IS ASSIGNED THE VALUE FALSE; TRUE INDICATES
NO MONITOR ENTRY. THE ACTUAL SEARCHING OF THE CONNECTION
TABLES IS DONE BY SRCHCT; LMEAS CALLS SRCHCT WITH

(6,0,0,MFND)

WHERE: 6 IDENTIFIES THE TYPE OF CONNECTION BEING
SEARCHED FOR (IN THIS CASE A MONITOR)
THE NEXT TWO ARGUMENTS ARE DUMMIES FOR THIS
TYPE - THE FIRST ARGUMENT IS ALL THAT
IS NEEDED TO SPECIFY A MONITOR ENTRY
MFND IS A LOGICAL VARIABLE WHICH SRCHCT SETS
(TRUE INDICATES A MONITOR ENTRY IS FOUND)

SRCHCT IS RESPONSIBLE FOR SETTING THE SCALE AND SWITCH
INFORMATION (IF ANY) IN COMMON.
SUBROUTINE LMEAS (ERRFLG)

START

CALL SRCHCT (6, 0, 0, MFND)

ERRFLG = ~ MFND

RETURN
PROGRAM NAME: LOCDAC

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: DADJ

CALLS: SRCHCT (TO SEARCH CONNECTION TABLES)

FILES USED: NONE

COMMON STORAGE: NONE

******************************************************************************************
SUBROUTINE LOCDAC
(IYPE, VOLT, INPHASE, ERRFLG)

START

IF IYPE = 'AC'

ITYPE = 1

IF IPHASE > 180

RCPHAS = IPHASE - 180

CALL SRCHCT
(IYPE, RCPHAS, VOLT, LOGFND)

ERRFLG = NOT LOGFND

RETURN
PROGRAM NAME: LOCDS

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: ACTION

CALLS: SRCHCT (TO SEARCH CONNECTION TABLE)

FILES USED: NONE

COMMON STORAGE: NONE

FUNCTION: THIS ROUTINE IS CALLED TO CHECK THE VALIDITY OF A CONNECTION BETWEEN THE CURRENT PIN AND THE D/S WHOSE NUMBER IS PASSED IN DSPIN. ERRFLG IS SET TO FALSE IF THE CONNECTION IS VALID. THE SEARCHING OF THE CONNECTION TABLES IS DONE BY SRCHCT WHICH LOCDS CALLS WITH THE SEQUENCE

(3,DSPIN,0,DSFND)

WHERE: 3 INDICATES A TYPE "DS" IS TO BE SEARCHED FOR

DSPIN IS THE NUMBER OF THE DESIRED D/S

0 IS NOT USED IN THIS INSTANCE AND ACTS AS A DUMMY ARGUMENT

DSFND IS A LOGICAL VARIABLE RETURNED BY SRCHCT (TRUE = FOUND)

SRCHCT SETS SCALE AND SWITCH INFORMATION IF THE CONNECTION IS VALID.

**************************************************************************
SUBROUTINE LOCDS (DSPIN, ERRFLG)

START

CALL JECNCT ('3', DSPIN, 0, DSFND)

ERRFLG = ~DSFND

RETURN
PROGRAM NAME: LOCLD
PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE
CALLED BY: DSET
CALLS: SRCHCT (TO SEARCH CONNECTION TABLES)
FILES USED: NONE
COMMON STORAGE: NONE

FUNCTION: THIS ROUTINE IS CALLED TO VERIFY THE VALIDITY OF
A CONNECTION BETWEEN THE CURRENT PIN AND A LOGIC
DISCRETE OF VALUE VOLT. ERRFLG IS RETURNED WITH A
FALSE VALUE IF THE CONNECTION IS VALID; TRUE INDICATES
AN ERROR. A CALL TO SRCHCT DOES THE ACTUAL SEARCHING
OF THE CONNECTION TABLES; THE CALLING SEQUENCE BEING

(2,IVOLT,0,LDFND)

WHERE: 2 INDICATES THE TYPE IS "LOGIC DISCRETE"
IVOLT IS THE VALUE OF THE LOGIC DISCRETE
LDFND IS THE LOGICAL VARIABLE SRCHCT PASSES
BACK (TRUE=FOUND)

SRCHCT SETS THE SCALE FACTOR AND SWITCH INFORMATION
IN COMMON WHEN A VALID CONNECTION IS FOUND.
Subroutine LOCAL (VOLT, ERRFLG)

START

IVOLT = VOLT

CALL SRC.HCT (2, IVOLT, 0, 0, 0)

ERRFLG = ~ LDEN

RETURN
PROGRAM NAME: LOCPIN

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: ACTION

CALLS: SRCHCT (TO SEARCH CONNECTION TABLE)

FILES USED: NONE

COMMON STORAGE: NONE

FUNCTION: THIS ROUTINE CHECKS TO SEE IF THE PIN INDICATED IN
THE INTEGER ARRAY PINID CAN BE CONNECTED TO THE CURRENT
PIN. ERRFLG IS SET TO FALSE IF IT IS A VALID CONNECTION.
LOCPIN CHECKS ON THE VALIDITY OF THE CONNECTION THROUGH
A CALL TO SRCHCT WHICH IN TURN DOES THE ACTUAL SEARCHING
OF THE CONNECTION TABLES. SRCHCT IS CALLED THROUGH THE
SEQUENCE

(1,PINID,0,PINFND)

WHERE: 1 INDICATES THE TYPE OF CONNECTION TO BE
SEARCHED FOR - A PIN
PINID IS THE PIN'S IDENTIFIER
0 IS A DUMMY ARGUMENT IN THIS CASE (NOT USED
WHEN SEARCHING FOR A PIN
PINFND IS A LOGICAL VARIABLE WHICH IS RETURNED
BY SRCHCT (TRUE=VALID CONNECTION)

IF THE CONNECTION IS VALID, SRCHCT DOES THE ACTUAL SETTING
OF SWITCH INFORMATION IN COMMON.

****************************************************
SUBROUTINE LOCPIN (PIN10, ERRFLG)

START

CALL SECHCT (1, PIN10, 0, PINEND)

ERRFLG = ~ PINEND

RETURN
PROGRAM NAME: PLOOK

PROGRAMMER: J.E. DOSS / J.B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: ACTION, CHANGE, DMONV, DSET, DMEAS, TRANS, DADJ, DISCON

CALLS: SEARCH (BINARY SEARCH)

FILES USED: PIN CONNECTION FILE

COMMON STORAGE: HW1TBL (DAC AND GAIN TABLES)
HW2TBL (PIN/CONNECTION INFO)
PININFO (SWITCH INFO FOR CURRENT PIN)
STRS (ASCII STRINGS)
LOGICAL FUNCTION PLOOK(PINID)

START

CALL SEARCH (---, Found)

PLOOK = Found

Found = TRUE?

READ CONNECTION FILE

FIRST WORD = "77777"

FIRST WORD = "77777"

NO

RETURN

YES

CNPTR = PRTAB(INDEX) + 1

NUMENT = NOENTAG

RETURN

VALIDITY CHECK: FIRST WORD MUST BE ALL ONES

SET POINT AND NUMBER OF ENTRIES
PROGRAM NAME: POPEN

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: DISCON

CALLS: SRCHCT (TO SEARCH CONNECTION TABLES)

FILES USED: NONE

COMMON STORAGE: NONE

FUNCTION: THIS ROUTINE IS CALLED TO CHECK FOR AN OPENC ENTRY
IN THE CONNECTION TABLES FOR THE CURRENT PIN. ERRFLG
IS RETURNED WITH A FALSE VALUE IF THE OPENC ENTRY IS
FOUND. THE ACTUAL SEARCHING IS DONE BY A CALL TO SRCHCT;
THE CALLING SEQUENCE IS THE SAME AS FOR A LOGIC DISCRETE
OF VALUE "99", I.E. (2,99,0,OPNFND), DUE TO THE MANNER
IN WHICH OPENC IS ENCODED IN THE CONNECTION TABLES.
SRCHCT SETS ALL APPROPRIATE SWITCH INFORMATION IN COMMON.
SUBROUTINE OPEN (ERRFLG)

START

CALL SRC.HCT (2,99,0,OPNFG)

ERRFLG = ~ (OPNFND)

RETURN
PROGRAM NAME: SELGN

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: CHANGE, DMONV, DMEAS, TRANS, DADJ, GCHECK

CALLS: NONE

FILES USED: NONE

COMMON STORAGE: HW1TBL (GAIN(I), GAINRNG(I), CONDFAC(I))

FUNCTION: THIS ROUTINE IS CALLED TO PICK THE APPROPRIATE
GAIN FOR THE GIVEN VOLTAGE AND CONDITIONING FACTOR.
VAL AND ISCALE ARE PASSED FROM THE CALLING PROGRAM;
VAL IS THE REAL VARIABLE REPRESENTING THE VOLTAGE;
ISCALE IS AN INTEGER INDEX INTO THE CONDITIONING
FACTOR TABLES. THE CONDITIONING FACTOR IS APPLIED
TO THE VOLTAGE AND A TEN PERCENT MARGIN IS ADDED TO
THE RESULT. (IF THE VOLTAGE EXCEEDS THE MAXIMUM
RANGE , 'TRUE' IS RETURNED IN ERRFLG; IF THE VOLTAGE
PLUS TEN PERCENT EXCEEDS THE MAXIMUM RANGE , THEN
THE HIGHEST GAIN IS ASSIGNED.) THE GAIN RANGE TABLES
ARE SEARCHED FOR THE RANGE IN WHICH THE MODIFIED VALUE
LIES. THE CORRESPONDING GAIN IS RETURNED IN IGAIN AND
ERRFLG IS SET TO FALSE (INDICATING THAT THE GAIN WAS SET).
SUBROUTINE SELGN (VAL, ISCAI, GAIN, ERRFLG)

START

VAL = ABS(VAL)

IF CONDFRC (ISCAI+1) EQ 0 THEN
    ERRFLG = TRUE
    RETURN

TRANGE = VAL / (CONDFRC (ISCAI+1))

PRANGE = TRANGE + (TRANGE > 1)

IGAIN = 0
ERRFLG = FALSE

IF TRANGE > GAIN*ENGI THEN
    ERRFLG = TRUE
    RETURN

IF PRANGE < GAIN*ENGU THEN
    IGAIN = GAIN
    RETURN

SEARCH FOR CORRECT GAIN; ASSIGN TO IGAIN

RETURN
PROGRAM NAME: SELLIM

PROGRAMMER: JUNE B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: DADJ

CALLS: NONE

FILES USED: NONE

COMMON STORAGE: STRS (ASCII STRINGS)
HWITBL (ACLOW(I),ACHIGH(I),DCLOW(I),DCHIGH(I))

FUNCTION: THIS FUNCTION RETURNS AN UPPER OR LOWER LIMIT FOR THE
ENTITY DETERMINED BY ITYPE AND REFER. ITYPE CAN HAVE
THE VALUE "DS","DC", OR "AC". D'REC HAS EITHER A VALUE
OF +1 OR -1 INDICATING AN UPPER OR LOWER LIMIT IS
DESIRED, RESPECTIVELY. IF ITYPE IS EQUAL TO "DS", A
VALUE OF 360 OR 0 IS RETURNED FOR AN UPPER OR LOWER LIMIT.
IF ITYPE INDICATES A DAC THEN REFER, THE DAC NUMBER WHICH
IS PASED TO THE FUNCTION, IS USED IN CONJUNCTION WITH
ITYPE TO INDEX INTO THE APPROPRIATE DAC TABLE BEFORE
ASSIGNING THE REQUESTED LIMIT.
REAL FUNCTION
SELLIM
(ITYPE, REFER, DIRECT)

START

INDEX = REFER + 1

IF TYPE = 'DS'
    IF DIRECT = 1
        SELLIM = 360
    ELSE
        SELLIM = 0
        RETURN
    END IF
ELSE IF TYPE = 'AC'
    IF DIRECT = 1
        SELLIM = ACHIGH(INDX)
    ELSE
        SELLIM = ACLOW(INDX)
        RETURN
    END IF
ELSE
    SELLIM = DCHIGH(INDX)
    RETURN
ENDIF
PROGRAM NAME: SRCHCT

PROGRAMMER: J.E. DOSS / J.B. THOMPSON
ENGINEERING EXPERIMENT STATION
GEORGIA INSTITUTE OF TECHNOLOGY

COMPUTER SYSTEM: DATA GENERAL ECLIPSE

CALLED BY: LMEAS
LOCMD
POPEN
LOCDS
LOCPIN

CALLS: NONE

FILES USED: PIN AND CONNECTION FILE

COMMON STORAGE: HW1TBL (DAC AND GAIN TABLES)
HW2TBL (PIN AND CONNECTION TABLES)
PININFO (SWITCH INFO ON CURRENT PIN)
STRS (ASCII STRINGS)

***********************************************************
SUBROUTINE SREC(T, TYPE, IDENT, RVAL, LOGEND)

START

I = LIMIT

MATCH = FALSE

NEXREC = CNPTR

I = CNPTR

I = I + 1

LOGEND = FALSE

RETURN

CHECK TO SEE IF TYPES MATCH

TYPE = I

MATCH = FALSE

CHECK IF CORRECT CONNECTION

MATCH = TRUE

SET SWITCH INFO

LOGEND = TRUE

RETURN
CHECK IF CORRECT CONNECTION

START

+ TYPE = PIN

NO

MATCH = TRUE

IDENT = CVAL

NO

MATCH = FALSE

YES

RETURN TO SRCHCT

IDENT = CVAL

NO

MATCH = TRUE

RETURN TO SRCHCT

IDENT

NO

LD OR OPENS

YES

IDENT = CVAL

NO

YES

1
1

IF TYPE = D/S
    NO
    THEN IDENT(I) = CVAL
    YES
    MATCH = TRUE
    REFER = 0
    RETURN TO SRCHCT

IF TYPE = AC, DAC
    NO
    THEN 2
    YES
    CORRECT DAC, I/H and PHASE
    NO
    THEN MATCH = TRUE
    REFER = CVAL
    RETURN TO SRCHCT
2

- **TYPE: DC DAC**
  - YES
  - **CORRECT DAC # (RANGE)**
  - YES
  - **MATCH = TRUE**
  - **REFER = CVL**
  - **RETURN TO SEARCH**

- **TYPE: MONITOR**
  - YES
  - **MATCH = TRUE**
  - **RETURN TO SEARCH**

- **NO**
SET SWITCH INFO

START

SET SCALE

NUMSW = 0

NON-BANK SWITCH ENTER

YES

NUMSW = NUMSW + 1

PUT SWITCHES IN ATTACK

READ NEXT RECORD

TYPE = 0 (CONTINUATION)

YES

RETURN TO SEEK
VI. ASSEMBLER ROUTINES

1. ADIREC
2. ASEND
3. CHKHEX
4. CHKLB
5. DE VADR
6. DRIVER
7. ERMSG
8. FINDLB
9. FIRST
10. FLUSH
11. GCHAR
12. GENREG
13. GETNEX
14. GETOK
15. HEX
16. LABDEC
17. LITCH
18. MULGEN
19. OUT
20. OUTADC
21. OUTCON
22. OUTDEV
23. OUTGN
24. OUTHEX
25. OUTHI
26. OUTLAB
27. OUTLIT
28. OUTLOW
29. OUTREG
30. OUTRL
31. OUTSW
32. OUTTOL
33. OUTVAL
34. OUTWD
35. PARAM
36. PASSM
37. PCODE
38. PGEN
39. RCONV
40. STORE
41. VALTOL
*SUBROUTINE ADIREC (IOPCOD)*

**TERM**

DATA GENERAL ECLIPSE

**GRAMMER**

J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY

**POSE**

This routine handles the section (.SEC) and subsection (.SUB) assembler directives. These directives specify the section or subsection name.

Note that the assembler directives are handled similar to a regular p-code operator (i.e. they exist in the table of code operators, they have an associated op-code, and they are a P-type (=7)).

**AGE**

Called by PGEN for assembler directives.

**LING SEQUENCE**

IOPCOD - input buffer - the associated op code of the directive.

= 000 for a section directive

= 001 for a subsection directive

**MON STORAGE**

1) GLOBL

NEOF - flag indicating end of the source file

MERROR - global error flag

2) LEXCL

LTOKTY - token type

LPCKBF(35) - token string in packed format

3) OUTBK

NAME(35) - the name of the current section or subsection in packed format

NTYPE - specifies whether name contains a section name (=0), a subsection name (=1), or is uninitialized (=−1)

**RNL REFERENCES**

SUBR ERMSG

SUBR GETNEX

***********************************************************
POSE-CALLED TO PRODUCE END OF ASSEMBLY STATISTICS.

GLOBAL REFERENCES-

MON STORAGE-

1) GLOBL
   MERNUM - NUMBER OF ERRORS DETECTED DURING ASSEMBLY

2) LISTS
   KSRCE - SET TO TRUE IF A SOURCE LISTING IS BEING PRODUCED
SEND

'4xx MEANING
ASSUMED ERRORS
DETECTED'

ERRORS?

'NO ASSUMED ERRORS
DETECTED'

RETURN
**NAME**
CHKLB

**STEM**
DATA GENERAL ECLIPSE

**PROGRAMMER**
J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

**PURPOSE**
THIS ROUTINE IS CALLED WHENEVER A NEW SUBSECTION HAS BEEN
DETECTED. IT CHECKS IF ANY LABEL REFERENCES IN THE PREVIOUS
SUBSECTION ARE STILL UNRESOLVED (I.E. NO LABEL DECLARATION
FOR THE REFERENCE WAS FOUND). AFTER THIS IT REINITIALIZES
THE LABEL TABLES FOR THE NEW SUBSECTION.

**AGE**
CALLED BY PASSM WHEN A SETTST P-CODE OPERATOR IS DETECTED.

**CALLING SEQUENCE**
NONE

**COMMON STORAGE**
1) GLOBL

MERNUM - NUMBER OF ASSEMBLY ERRORS DETECTED

2) LISTS

KSRCE - SET TO TRUE IF A SOURCE LISTING IS BEING
PRODUCED

3) LABL

NUMLAB - THE NUMBER OF LABELS CURRENTLY IN THE
TABLE
LABID (10,5) - THE CHARACTERS OF THE LABEL IDENTIFIER
IN PACKED FORMAT
LABLOC (10) - LOCATION OF THE LABEL DECLARATION IN THE
OUTPUT BUFFER; SET TO -1 IF UNRESOLVED

**EXTERNAL REFERENCES**
NONE

******************************************************************************
CHKLB

$I \leftarrow 1$

$I \leq \text{number of labels}$
\[ \begin{align*} &\text{yes} \\
&\text{no} \
\end{align*} \]

\[ \begin{align*} &\text{LABELS} \leftarrow \phi \\
&\text{RETURN} \\
&I \leftarrow I + 1 \\
&'\text{ERROR - UNRESOLVED LABEL REFERENCE.'} \]

$\text{LABELS} \leftarrow \phi$
THIS ROUTINE CHECKS IF A TOKEN IS A VALID DEVICE
IF IT IS, THE BINARY REPRESENTATION FOR THE DEVICE
IS CALCULATED.

NOTE THAT THE CONVERSE CONDITION OF A DEVICE IS INDICATED BY THE INSERTION OF A 'C' BEFORE THE DEVICE NUMBER.

CALLED BY GETNEX TO IDENTIFY A DEVICE.

LING SEQUENCE-
JDEV - OUTPUT PARAMETER - SET TO TRUE IF A VALID
DEVICE IS IDENTIFIED.

MON STORAGE-
1) LEXCL
   LTOKTY - TOKEN TYPE
   LTOKVA - TOKEN VALUE IN INTEGER FORMAT
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
   INCHAR - NUMBER OF CHARACTERS IN TOKEN
   LPCBF(35) - TOKEN STRING IN PACKED FORMAT

3) LITCH
   KLITC  - LITERALLY 'C'
   KLITDS - LITERALLY 'DS'
   KLITDC - LITERALLY 'DC'
   KLITAC - LITERALLY 'AC'

RNL REFERENCE:
FUNC RCONV

RKS-
THE BIT ASSIGNMENTS FOR THE DEVICE ADDRESS ARE AS FOLLOWS:

BIT 15  - SPECIFIES DIGITAL/SYNCHRO
BIT 14  - SPECIFIES AN DC DAC
BIT 13  - SPECIFIES AN AC DAC
BIT 8   - INDICATES THE CONVERSE CONDITION OF A DEVICE, I.E. COUNTERCLOCKWISE FOR THE D/S AND OUT OF PHASE FOR THE AC DAC.
BITS 7-0  - NUMERIC VALUE FOR THE DEVICE ADDRESS
THIS ROUTINE SETS UP THE OPTIONS ASSOCIATED WITH THE P-CODE ASSEMBLER. DEPENDING ON THE OPERATOR'S RESPONSES, THREE FLAGS ARE SET. THE FLAGS WITH THE LOGICAL UNIT NUMBERS THAT THEY AFFECT ARE AS FOLLOWS:

KOUTPT - LUN 14 - PRODUCES EXECUTABLE OUTPUT FILE
KSRCE - LUN 12 - PRODUCES P-CODE LISTING
KOBJ - LUN 12 - DISPLAYS OBJECT CODE ON LISTING FILE

NOTE THAT THE INPUT P-CODE FILE IS ALWAYS NAMED 'PCODE.ATP'.

USES USED-
PCODE.ATP - LUN 13 - INPUT FILE CONTAINING SOURCE P-CODE
AN OUTPUT FILE - LUN 14 - SPECIFIED BY THE OPERATOR
LISTING - LUN 12 - TO THE LINE PRINTER
EOJ SUMMARY - LUN 11 - TO THE OPERATOR
DRIVER

OBJECT FILE PRODUCED? Yes

GET OBJECT FILE NAME

OBJECT ON LISTOR? Yes

KOBJ ← ,TRUE,

LISTING PRODUCED? No

KOUPTR ← ,TRUE,

KRREF ← ,TRUE,

ASSEMBLE THE P-CODE FILE

PASSM

STOP
**ERRORMSG (MSGNUM)**

**DATA GENERAL ECLIPSE**

**J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY**

**THIS ROUTINE PRINTS ERROR MESSAGES FOR THE ASSEMBLER. THE ONLY EXCEPTION IS CHKLB WHICH PRINTS ERROR MESSAGES FOR UNRESOLVED LABEL REFERENCES.**

**CALLED TO PRINT ERROR MESSAGES.**

**SEQUENCE**

**MSGNUM** - INPUT PARAMETER - SPECIFIES WHICH ERROR MESSAGE TO PRINT.

**STORAGE**

1) **GLOBL**
   - **MERNUM** - THE NUMBER OF ERRORS DETECTED.
2) **LISTS**
   - **KSRCE** - SET TO TRUE WHEN A SOURCE LISTING IS BEING PRODUCED
3) **MESSG**
   - **RHEAD (3)** - ERROR MESSAGE HEADER
   - **RMSG (7,30)** - ERROR MESSAGES

**REFERENCES**

**NONE**
E- FINDLB (INDX,IFIND,IERR)

TERM- DATA GENERAL ECLIPSE

GRAMMER- J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

POSE-
THIS ROUTINE SEARCHES THE LABEL TABLES FOR THE LABEL
WHOSE ID IS STORED IN LPCKBF. IF IT IS FOUND, IT
RETURNS ITS LOCATION IN THE TABLE. IF NOT FOUND, IT
ADDS IT TO THE END OF THE TABLE.

GE-
CALLED TO SEARCH THE LABEL TABLE FOR A SPECIFIED ID.

LING SEQUENCE-
INDX - OUTPUT PARAMETER - THE INDEX OF THE LABEL IN THE
TABLE.
IFIND - OUTPUT PARAMETER - SET TO TRUE IF THE LABEL IS
FOUND TO EXIST IN THE TABLE ALREADY.
IERR - OUTPUT PARAMETER - SET TO TRUE WHEN THE TABLE
ARRAY SIZE IS EXCEEDED WHEN ADDING A NEW LABEL.

MON STORAGE-
1) LEXCL
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT
2) LABL
   NUMLAB - THE NUMBER OF LABELS CURRENTLY IN THE
   TABLE
   LABID (10,5) - THE CHARACTERS OF THE LABEL IDENTIFIER
   IN PACKED FORMAT
   LABLOC (10) - THE LOCATION OF THE LABEL IN THE OUTPUT
   BUFFER; SET TO -1 IF NOT RESOLVED
   LREFNO (10) - THE NUMBER OF UNRESOLVED REFERENCES TO
   A LABEL

ERNAL REFERENCES-
SUBR ERMSG
This routine is used to get the first symbol of a line of P-CODE. This symbol should be either a P-CODE operator or a label declaration.

Called by PASr1 to get the first symbol of a P-CODE statement.

This routine is used to get the first symbol of a line of P-CODE. This symbol should be either a P-CODE operator or a label declaration.

Called by PASr1 to get the first symbol of a P-CODE statement.

None

MON STORAGE-
1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LEXCL
   LTOKTY - TOKEN TYPE
   LTOK VA - TOKEN VALUE IN INTEGER FORMAT
   LPTYPE - P-CODE OPERATOR TYPE
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
   INCHAR - NUMBER OF CHARACTERS IN TOKEN
3) LITCH
   KCOLON - LITERALLY ': '
   KBLANK - LITERALLY ' '

EXTERNAL REFERENCES-
   SUBR GETOK
   FUNC PCODE
   SUBR STORE
**FLUSH**

**DATA GENERAL ECLIPSE**

**E. S. ACREE, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY**

**POSE**

THIS ROUTINE IS CALLED AT THE BEGINNING OF A NEW SECTION OR SUBSECTION. FIRST, THE OUTPUT BUFFER CONTAINING THE OBJECT CODE FOR THE PREVIOUS SUBSECTION IS DUMPED TO THE OUTPUT FILE. THIS IS DONE USING THE FOLLOWING FORMAT:

**OBJECT CODE RECORD = :NNAAAAARRBBBB . . . BBBBCC**

**COL 1** - COLON (:) INDICATING THIS IS AN OBJECT CODE LINE

**COL 2-3** - NN; NUMBER OF DATA BYTES ON THIS LINE

**COL 4-7** - AAAA; ADDRESS OF THE FIRST DATA BYTE (=0000 AT THE BEGINNING OF A SUBSECTION)

**COL 8-9** - RR; RECORD TYPE (=00)

00 INDICATES AN ABSOLUTE RECORD TYPE

**COL 10-79 MAX** - BB; EACH TWO CHARACTERS IS A DATA BYTE

**COL 11 MIN-80 MAX** - CC; CHECKSUM, THE EIGHT BIT TRUNCATED SUM OF ALL ASCII-ENCODED-BYTES ON THIS LINE.

NOTE THAT WITH THIS FORMAT, EACH SUBSECTION BEGINS WITH THE ADDRESS OF THE FIRST DATA BYTE EQUAL TO ZERO.


EVERYTHING ON THE OBJECT CODE LINE IS IN HEX.

A BEGINNING OF SECTION LINE (BOSL) PRECEDES A SERIES OF SUBSECTIONS. A BEGINNING OF SUBSECTION LINE PRECEDES THE OBJECT CODE OF A SUBSECTION.

**BEGINNING OF SECTION LINE RECORD:**

**COL 1** - PERIOD (.) INDICATING THAT THIS IS A BOSL LINE

**COL 2-71 MAX** - SECTION NUMBER AND NAME

**BEGINNING OF SUBSECTION LINE RECORD:**

**COL 1** - COMMA (,) INDICATING THAT THIS IS A BOSSL

**COL 2-3** - VARIANT IDENTIFIER

**COL 4-71 MAX** - SUBSECTION NUMBER

NOTE THAT THE VARIANT NUMBER IS ALREADY INCLUDED IN THE PROPER PLACE WITHIN THE NAME ARRAY.

- CALLED BY PASSM ON DETECTION OF A .SEC OR .SUB ASSR DIRECTIVE.
MON STORAGE-

1) LEXCL
   LTKTY - TOKEN TYPE
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
   INCHAR - NUMBER OF CHARACTERS IN TOKEN
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT
   ISKIP - SET TO TRUE WHEN A LOOKAHEAD HAS BEEN
           DONE ON THE NEXT TOKEN

2) OUTBK
   LOCOUT(600) - OUTPUT BUFFER CONTAINING A SUB-
                  SECTION’S OBJECT CODE
   LOCCNT - POINTER INTO OUTPUT BUFFER INDICATING
            THE END OF OBJECT CODE
   NAME(35) - THE NAME OF THE CURRENT SECTION OR
               SUBSECTION IN PACKED FORMAT
   NTYPE - SPECIFIES WHETHER NAME(35) CONTAINS A SECTION
           OR SUBSECTION NAME.
           NTYPE = -1 NO NAME; NOT INITIALIZED
           NTYPE = 0 SECTION NAME
           NTYPE = 1 SUBSECTION NAME

3) LISTS
   KOUT - SET TO TRUE WHEN AN EXECUTABLE
           OUTPUT FILE IS BEING PRODUCED

4) LITCH
   KBLANK - LITERALLY ' '
   KCOMMA - LITERALLY ',', '
   KPERID - LITERALLY '.', '
   KCOLON - LITERALLY ': '

EXTERNAL REFERENCES-
   SUBR GETNEX
   SUBR HEX

***********************************************************
**E-**

GCHAR (CHAR)

**ITEM-**

DATA GENERAL ECLIPSE

**GRAMMER-**

J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

**POSE-**

THIS ROUTINE RETURNS THE NEXT CHARACTER FROM THE SOURCE
FILE, READING IN NEW LINES AS NEEDED. IT ALSO PRODUCES
A SOURCE LISTING IF SO SPECIFIED.

**GE-**

CALLED BY GETOK TO GET THE NEXT CHARACTER.

**LING SEQUENCE-**

CHAR - OUTPUT PARAMETER - THE NEXT CHARACTER IN ASCII
      PADDED WITH A BLANK.

**MON STORAGE-**

1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LITCH
   KBLANK - LITERALLY '
3) INITV
   LPTR - POINTER INTO THE LINE OF SOURCE CODE.
   LINE(80) - THE LINE OF SOURCE CODE
4) LISTS
   KSRCE - SET TO TRUE IF A SOURCE LISTING IS DESIRED.

**ERNAL REFERENCES-**

SUBR ERMESG
GCHAK

LPTR > 120?

READ/WRIT
NEXT LINE

LPTR ← 1

CHAR ← LINE(LPTR)

LPTR ← LPTR + 1

RETURN
THIS ROUTINE PROCESSES P-CODE OPERATORS WHICH HAVE A
REGISTER AS A PARAMETER. THE OPCODE IS MODIFIED WITH
THE VALUE OF THE REGISTER REFERENCE. THE RESULTING
VALUE IS WRITTEN TO THE OUTPUT BUFFER.

CALLED BY PGEN FOR P-CODE OPERATORS OF TYPE 4.
GENREG

GET THE NEXT TOKEN

ERROR - NOT A REGISTER PARAMETER

ERROR - INVALID REGISTER REFERENCE

TOKEN TYPE = 2

TOKEN VALUE > 15

MODIFY OP-CODE WITH TOKEN VALUE

OUTLAY

OUTPUT 8 BIT OP-CODE TO BUFFER...

RETURN
THIS ROUTINE IDENTIFIES THE P-CODE PARAMETERS FOLLOWING A P-CODE OPERATOR. WHEN A P-CODE OPERATOR IS IDENTIFIED, THIS ROUTINE IS REPEATEDLY CALLED TO GET THE OPERATOR'S PARAMETERS.

CALLED TO GET THE NEXT P-CODE PARAMETER.

MON STORAGE-
1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LEXCL
   LTOKTY - TOKEN TYPE
   LTOKVA - TOKEN VALUE IN INTEGER FORMAT
   RTOKVA - TOKEN VALUE IN REAL FORMAT
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
   INCHAR - NUMBER OF CHARACTERS IN TOKEN
   ISKIP - SET TO TRUE WHEN A LOOKAHEAD HAS BEEN DONE ON THE NEXT TOKEN
3) LITCH
   KLITS - LITERALLY 'S'
   KLITG - LITERALLY 'G'
   KLITC - LITERALLY 'C'
   KPERC - LITERALLY '%'
   KLITR - LITERALLY 'R'
   KZERO - LITERALLY '0'
   KININE - LITERALLY '9'
   KPLUS - LITERALLY '+'
   KMINUS - LITERALLY '−'
4) OLD
   LOLDTY - PREVIOUS TOKEN TYPE
   LOLDVA - PREVIOUS TOKEN VALUE
   ROLDVA - PREVIOUS TOKEN VALUE IN REAL FORMAT
   LOLDP - PREVIOUS P-CODE OPERATOR'S TYPE
   MOLDER - PREVIOUS ERROR CONDITION

RNAL REFERENCES-
  SUBR GETOK
  SUBR DEVADR
  SUBR PARAM
  SUBR VALTOL
  SUBR ERMSG
  FUNC ISHFT - DATA GENERAL SUPPLIED
2

CHHEX
CHECK THAT FOLLOWING CHAR IS HEX

1

FIRST CHAR = '0' %

PARAM
CHECK THAT FOLLOWING NUMBER < 15

FIRST CHAR = 'R'

VALTOL
CHECK FOR VALID VALUE OR TOLERANCE

FIRST CHAR = 'L' - 'U'

ERROR - INVALID PARAMETER

SAVE THIS TOKEN

RETURN
THIS ROUTINE ASSEMBLES A TOKEN FROM THE STREAM OF INPUT CHARACTERS. IT ALSO IDENTIFIES TOKENS THAT ARE LITERALS.

THE TOKENS ARE STORED IN TWO FORMATS: UNPACKED AND PACKED. "INBUF" CONTAINS THE TOKEN STRING IN UNPACKED FORMAT WITH EACH ELEMENT CONTAINING A CHARACTER LEFT-JUSTIFIED AND A BLANK. "LPCKBF" CONTAINS THE TOKEN IN PACKED FORMAT WITH TWO CHARACTERS OF THE TOKEN PER ARRAY ELEMENT AND PADDED WITH BLANKS. THE PACKED FORMAT IS USED FOR TABLE SEARCHES.

CALLED TO GET THE NEXT TOKEN FROM THE SOURCE FILE.

LITFLG - OUTPUT PARAMETER - SET TO TRUE WHEN A LITERAL IS DETECTED.

MON STORAGE-
1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LECL
   INCHAR - NUMBER OF CHARACTERS IN TOKEN
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT
3) LITCH
   KLPARN - LEFT PARENTHESIS CHARACTER
   KRPARN - RIGHT PARENTHESIS CHARACTER
   KCOMMA - COMMA CHARACTER
   KSQUOT - SINGLE QUOTE CHARACTER
   KDQUOT - DOUBLE QUOTE CHARACTER
   KBLANK - BLANK CHARACTER

RNL REFERENCES-
SUBR GCHAR
SUBR STORE
SUBR ERMSG
**SUBROUTINE HEX (IWORD, NOCHAR, STRING)**

**DATA GENERAL ECLIPSE**

**J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY**

**POSE**

**THIS ROUTINE ENCODES A 16 BIT WORD INTO ITS ASSOCIATED DIGITS.**

**CALLED BY FLUSH AND LABDEC TO ENCODE INTEGER VARIABLES.**

**LING SEQUENCE**

- **IWORD** - INPUT PARAMETER - THE INTEGER TO ENCODE
- **NOCHAR** - INPUT PARAMETER - THE NUMBER OF HEX DIGITS DESIRED (STARTS WITH THE LEAST SIGNIFICANT)
- **STRING (2)** - OUTPUT PARAMETER - INTEGER ARRAY CONTAINING THE PACKED ASCII-ENCODED HEX DIGITS

**MON STORAGE**

**NONE**

**ERNAL REFERENCES**

- **FUNC ISHFT** - DATA GENERAL SUPPLIED

*******************************************************************************
THIS ROUTINE IS CALLED WHEN A LABEL DECLARATION HAS BEEN DETECTED. INITIALLY THE LABEL TABLE IS SEARCHED FOR THIS LABEL. IF THE LABEL IS NOT FOUND, IT IS ADDED TO THE END OF THE TABLE AUTOMATICALLY BY FINDLB. IF IT IS FOUND, THE LABEL LOCATION FIELD IS CHECKED TO SEE IF IT HAS ALREADY BEEN RESOLVED. IF RESOLVED, THERE ARE DUPLICATE LABELS. IF NOT RESOLVED, THE LABEL LOCATION FIELD IS UPDATED WITH THE CURRENT LOCATION AND ANY UNRESOLVED REFERENCES TO THIS LABEL ARE RESOLVED.

NOTE THAT SINCE THE OUTPUT BUFFER IS FLUSHED FOR EACH NEW SUBSECTION, REFERENCES TO LABELS ARE RESTRICTED TO THE SECTION IN WHICH THE LABEL IS DECLARED (AS REFERENCES IN PREVIOUS SUBSECTIONS COULD NOT BE RESOLVED). STANDARD PROCEDURE REFERENCES ARE THE ONLY EXCEPTION TO THIS.

CALLED BY PASSM WHEN A LABEL DECLARATION IS FOUND.

COMMON STORAGE-

1) GLOBL
   MERROR - GLOBAL ERROR FLAG

2) LEXCL
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT

3) OUTBK
   LOCCNT - POINTER INTO OUTPUT BUFFER INDICATING THE END OF THE OBJECT CODE
   LOCOUT(600) - OUTPUT BUFFER CONTAINING THE OBJECT CODE

4) LABL
   LABLOC (10) - LOCATION OF THE LABEL DECLARATION IN THE OUTPUT BUFFER; SET TO -1 IF UNRESOLVED
   LREFNO (10) - THE NUMBER OF UNRESOLVED REFERENCES TO A LABEL
   LABREF (10,10) - THE LOCATIONS IN THE OUTPUT BUFFER OF UNRESOLVED REFERENCES TO A LABEL

FINAL REFERENCES-

SUBR FINDLB
SUBR ERMSG
SUBR HEX
LABDEC

CHECK IF LABEL IN TABLE, ADD IT IF NOT.

ERROR? YES

NOT FOUND IN TABLE? YES

LABEL ALREADY RESOLVED? NO

UPDATE LOCATION INFORMATION IN TABLE.

UPDATE PREVIOUS REFERENCES TO THIS LABEL IN THE OBJECTS.

RETURN
TO INTIALIZE THE VARIABLES IN THE COMMON BLOCK LITCH.

COMMON STORAGE-
  1) LITCH

KLARN - LITERALLY '( '
KRPARN - LITERALLY ') '
KCOMMA - LITERALLY ', '
KSQLT - LITERALLY '" '
KDQUOT - LITERALLY '" '
KBLANK - LITERALLY ' '
KLITS - LITERALLY 'S '
KLITG - LITERALLY 'G '
KLITC - LITERALLY 'C '
KPERC - LITERALLY '%'
KLITR - LITERALLY 'R '
KZERO - LITERALLY '0 '
KNINE - LITERALLY '9 '
KPLUS - LITERALLY '+'
KMINUS - LITERALLY '-'
KPERID - LITERALLY '.'
KLITDS - LITERALLY 'DS'
KLITDC - LITERALLY 'DC'
KLITAC - LITERALLY 'AC'
KLITSP - LITERALLY 'SP'
KCOLON - LITERALLY ':'
KLITA - LITERALLY 'A '
KLITF - LITERALLY 'F'
**E-** MULGEN (IOPCOD)

**TEM-** DATA GENERAL ECLIPSE

**GRAMMER-** J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY

**POSE-**

THIS ROUTINE HANDLES P-CODE OPERATORS WHICH HAVE
MULTIPLE PARAMETERS. THE PARAMETERS ARE CHECKED FOR
THE CORRECT ORDER AND TYPE.

**GE-**

CALLED BY PGEN FOR P-CODE OPERATORS OF TYPE 6.

**LING SEQUENCE-**

IOPCOD - INPUT PARAMETER - THE P-CODE OPERATOR'S OPCODE

**MON STORAGE-**

1) GLOBL

MERROR - GLOBAL ERROR FLAG

**ERNAL REFERENCES-**

SUBR OUTGN
SUBR OUTCON
SUBR OUTADC
SUBR OUTVAL
SUBR OUTTOL
SUBR OUTREG
SUBR OUTDEV

*****************************************************************************
NOTE: A CALL IS MADE TO EACH ROUTINE SPESIFIED. ALSO NOTE THAT 'A/D REF' REFERS TO THE CONTROL, OUTD, OUTON, AND OUTDE.
OUT (IHEX)

DATA GENERAL ECLIPSE

J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

THIS ROUTINE LOADS THE TWO ASCII-ENCODED HEX DIGITS
IN IHEX INTO THE OUTPUT BUFFER. THESE DIGITS ARE
INCLUDED IF THE APPROPRIATE FLAGS ARE SET.

CALLED BY OUTLOW AND OUTHEX TO LOAD DATA INTO THE OUTPUT
BUFFER.

IHEX - INPUT PARAMETER - CONTAINS THE TWO ASCII-ENCODED
HEX DIGITS OF OBJECT CODE.

ION STORAGE-
1) GLOBL
   MERROR - GLOBAL ERROR FLAG
2) OUTBK
   LOCOUT(600) - OUTPUT BUFFER CONTAINING A SUB-
   SECTION’S OBJECT CODE
   LOCCNT - POINTER INTO OUTPUT BUFFER INDICATING
   THE END OF OBJECT CODE
3) LISTS
   KSRCE - SET TO TRUE WHEN A LISTING FILE IS BEING
   PRODUCED
   KOBJ - SET TO TRUE WHEN THE OBJECT CODE IS BEING
   PRINTED ON THE SOURCE LISTING

RAL REFERENCES-
SUBR ERMSG

**************************************************
OUT

POINTER ≤ 600
  YES
  NO

LOCOUT(POINTER)
  ← OBJECT code
  POINT ⇒
  POINT ⇒ + 1

SOURCE ← OBJECT flag
  YES
  NO

WRITE OBJECT code

RETURN
OUTADC

DATA GENERAL ECLIPSE

J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

This routine inputs a token, verifies that it is of the correct type, and writes it out to the output buffer. Called by PGEN and MULGEN to output an A/D address.

ANG SEQUENCE-
NONE

ION STORAGE-
  1) GLOBL
     MEOF - FLAG INDICATING END OF THE SOURCE FILE
     MERROR - GLOBAL ERROR FLAG
  2) LEXCL
     LTOKTY - TOKEN TYPE
     LTOKVA - TOKEN VALUE IN INTEGER FORMAT

EXTERNAL REFERENCES-
  SUBR GETNEX
  SUBR ERMSG
  SUBR OUTWD
THIS ROUTINE INPUTS A TOKEN, VERIFIES THAT IT IS OF THE CORRECT TYPE, AND WRITES IT OUT TO THE OUTPUT BUFFER.

Called by PGEN and MULGEN to output a conditioning spec.

External References:
- SUBR GETNEX
- SUBR ERMSG
- SUBR OUTLOW

DON STORAGE:
- 1) GLOBL
  - MEOF - Flag indicating end of the source file
  - MERROR - Global error flag
- 2) LEXCL
  - LTOKTY - Token type
  - LTOKVA - Token value in integer format
OUTDEV
DATA GENERAL ECLIPSE
J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

POSE-
THIS ROUTINE INPUTS A TOKEN, VERIFIES THAT IT IS OF THE
CORRECT TYPE, AND WRITES IT OUT TO THE OUTPUT BUFFER.

CALLED BY PGEN AND MULGEN TO OUTPUT A DEVICE ADDRESS.

ENTRY SEQUENCE-
NONE

MON STORAGE-
1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LEXCL
   LTOKTY - TOKEN TYPE
   LTOKVA - TOKEN VALUE IN INTEGER FORMAT

EXTERNAL REFERENCES-
   SUBR GETNEX
   SUBR ERMSG
   SUBR OUTWD

******************************************************************************
**OUTGN**

**DATA GENERAL ECLIPSE**

**J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY**

**THIS ROUTINE INPUTS A TOKEN, VERIFIES THAT IT IS OF THE CORRECT TYPE, AND WRITES IT OUT TO THE OUTPUT BUFFER.**

**CALLED BY PGEN AND MULGEN TO OUTPUT A GAIN SPEC.**

**NONE**

**1) GLOBL**

- **NEOF** - FLAG INDICATING END OF THE SOURCE FILE
- **MERROR** - GLOBAL ERROR FLAG

**2) LEXCL**

- **LTOKTY** - TOKEN TYPE
- **LTOKVA** - TOKEN VALUE IN INTEGER FORMAT

**EXTERNAL REFERENCES**

- **SUBR GETNEX**
- **SUBR ERMSG**
- **SUBR EOUTLOW**
OUTGN

YES NO
ERROR FLAG SET?

GETNEXT

GET NEXT TOKEN

TOKEN TOPE

ERRMSG

ERROR - NOT A GAIN PARAMETER

OUTPUT 8 BIT SPECIFIER

RETURN
SUBROUTINE OUTHEX (NUMDIG)

DATA GENERAL ECLIPSE

J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY

THIS ROUTINE PROCESSES A/D AND SWITCH ADDRESSES. THE ADDRESSES ARE INPUTED TO THE ASSEMBLER IN HEX AND SO ONLY NEED BE PADDED TO THE LEFT WITH ASCII-ENCODED HEX ZEROS TO THE REQUIRED LENGTH BEFORE BEING STORED IN THE OUTPUT BUFFER. NOTE THAT A/D ADDRESSES NEED 6 HEX DIGITS (24 BITS) AND SWITCH ADDRESSES NEED 4 DIGITS (16 BITS).

CALLED BY OUTSW AND OUTADC TO OUTPUT A SWITCH OR A/D ADDR.

NUMDIG - INPUT PARAMETER - THE NUMBER OF HEX DIGITS REQUIRED (4 OR 6).

STORAGE

1) LEXCL
   LTOQVA - TOKEN VALUE (NUMBER OF HEX DIGITS)
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT

2) LITCH
   KZERO - LITERALLY '0'

EXTERNAL REFERENCES

SUBR OUT
FUNC ISHFT - DATA GENERAL SUPPLIED
OUTHEX

LOAD HEX
DIGITS INTO
BUFFER LEFT
JUSTIFIED

I ← 1

IF I ≥ #
OF DIGITS
YES

PACK TWO
HEX CHARS

OUT

I ← I + 2

LOAD RESULT
INTO OUTPUT
BUFFER

RETURN
OUTHI (IWORD)

DATA GENERAL ECLIPSE

J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

This routine encodes the high byte of IWORD into hex ASCII and stores the value into the next available space in the output buffer.

Called to output a byte of object code located in the high byte of a word.

INPUT PARAMETER - WORD CONTAINING THE BYTE OF OBJECT CODE

NONE

EXTERNAL REFERENCES -
FUNC ISHFT - DATA GENERAL SUPPLIED
SUBR OUTLOW

******************************************************************************
OUT HI

SHIFT WORD 8 BITS TO THE RIGHT

OUTPUT LOW BYTE TO BUFFER

RETURN
THIS ROUTINE GETS THE PARAMETER ASSOCIATED WITH A BRANCH INSTRUCTION (P-CODE OPERATOR TYPE 3). THIS PARAMETER IS EITHER A LABEL REFERENCE OR A STANDARD PROCEDURE NAME. FIRST THE ROUTINE DETERMINES IF THE PARAMETER IS A STANDARD PROCEDURE NAME. IF SO, THE STANDARD PROCEDURE NUMBER IS WRITTEN OUT WITH THE MOST SIGNIFICANT BIT SET. IF NOT, THE REFERENCE IS ASSUMED TO BE TO A LABEL.

THE LABEL TABLE IS THEN SEARCHED. NOTE THAT ONLY THE FIRST 10 CHARACTERS OF THE LABEL ARE SIGNIFICANT. IF THE TOKEN IS FOUND IN THE LABEL TABLE, IT IS THEN CHECKED AS TO WHETHER ITS LOCATION HAS BEEN RESOLVED (I.E. IF THE LABEL HAS ALREADY BEEN DECLARED). IF RESOLVED, THE OFFSET TO THE LABEL DECLARATION LOCATION FROM THE CURRENT LOCATION IS CALCULATED AND WRITTEN OUT. IF NOT RESOLVED, THE CURRENT LOCATION IS ADDED TO THE LABREF TABLE SO IT CAN BE RESOLVED WHENEVER THE LABEL DECLARATION IS FOUND. A ZERO IS WRITTEN OUT.

IF THE TOKEN IS NOT FOUND IN THE LABEL TABLE, IT IS ADDED TO THE TABLE WITH THE LABLOC FIELD SET TO -1 TO INDICATE THAT IT IS UNRESOLVED. THE CURRENT LOCATION POINTER IS STORED IN LABREF SO THAT IT CAN BE RESOLVED WHENEVER THE LABEL DECLARATION IS FOUND. A ZERO IS WRITTEN OUT.

CALLED BY PGEN TO PROCESS A BRANCH REFERENCE.

CALLING SEQUENCE:
NONE

ON STORAGE:
1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LEXCL
   INCHAR - NUMBER OF CHARACTERS IN TOKEN
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT
3) OUTBK
   LOCCNT - POINTER INTO OUTPUT BUFFER INDICATING THE CURRENT END OF THE OBJECT CODE
4) LITCH
   KLITSP - LITERALLY 'SP'
5) LABL
   LABLOC (10) - LOCATION OF THE LABEL DECLARATION IN THE OUTPUT BUFFER; SET TO -1 IF UNRESOLVED
   LCREFO (10) - THE NUMBER OF UNRESOLVED REFERENCES TO A LABEL
   LABREF (10,10) - THE LOCATIONS IN THE OUTPUT BUFFER OF UNRESOLVED REFERENCES TO A LABEL
6) INITV
   MAXSP - THE LARGEST STANDARD PROCEDURE NUMBER

NAL REFERENCES—
SUBR OUTWD
SUBR FINDLB
FUNC RCONV
SUBR ERMSG
SUBR OUTWD

***************************************************************************
This routine gets the next token, verifies that it is a literal, and outputs the literal string. It also saves the name of the next section/subsection.

Called by PGEN for P-code operators requiring a literal (i.e., type 1 operators).

**GLOBAL REFERENCES:**

- SUBR GETnex
- SUBR OutHI
- SUBR OutLow
- SUBR ErmSG
**OUTLOW (IWORD)**

**DATA GENERAL ECLIPSE**

**J. E. DOSS, ENGINEERING EXPERIMENT STATION,**
**GEORGIA INSTITUTE OF TECHNOLOGY**

**THIS ROUTINE ENCODES THE LOW BYTE OF IWORD INTO HEX ASCII AND STORES THE VALUE INTO THE NEXT AVAILABLE SPACE IN THE OUTPUT BUFFER.**

**CALLED TO OUTPUT A BYTE OF OBJECT CODE.**

**LING SEQUENCE—**

**IWORD — INPUT PARAMETER — WORD CONTAINING THE BYTE OF OBJECT CODE**

**MON STORAGE—**

1) **GLOBL**

**MERROR — GLOBAL ERROR FLAG**

**EXTERNAL REFERENCES—**

**FUNC ISHFT — DATA GENERAL SUPPLIED**

**SUBR OUT**

*******************************************************************************
OUT LOW

CONVERT BITS TO TWO HEX CHAR

PACK HEX CHAR

OUT

OUTPUT WORD TO BUFFER

RETURN
**OUTREG**

**DATA GENERAL ECLIPSE**

**J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY**

**POSE**

This routine inputs a token, verifies that it is of the correct type, and writes it out to the output buffer.

**GENERAL**

Called by PGEN and MULGEN to output a register reference.

**LING SEQUENCE**

NONE

**MON STORAGE**

1) GLOBL
   MEEOF - flag indicating end of the source file
   MERROR - global error flag

2) LEXCL
   LTOKTY - token type
   LTOKVA - token value in integer format

**EXTERNAL REFERENCES**

SUBR GETNEX
SUBR ERMSG
SUBR OUTLOW
OUTREG

ERROR FLAG SET? NO

GET NEXT TOKEN

TOKEN TYPE ≠ 2

ERROR - NOT A REGISTER PARAMETER

OUTPUT LOW BYTE TO BUFFER

RETURN
OUTRL (RWORD)

SYSTEM: DATA GENERAL ECLIPSE

PROGRAMMER: J. E. DOSS, ENGINEERING EXPERIMENT STATION,
             GEORGIA INSTITUTE OF TECHNOLOGY

PURPOSE:
   THIS ROUTINE ENCODES THE 4 BYTES OF RWORD INTO HEX ASCII
   AND STORES THE VALUE INTO THE NEXT 4 AVAILABLE SPACES
   IN THE OUTPUT BUFFER.

USAGE:
   CALLED TO OUTPUT A REAL VALUE OF OBJECT CODE.

INPUT SEQUENCE:
   RWORD - INPUT PARAMETER - REAL VARIABLE CONTAINING THE
            4 BYTES OF OBJECT CODE

COMMON STORAGE:
   1) OTDUM
      1 (2) - FOR EQUIVALENCING PURPOSES

EXTERNAL REFERENCES:
   SUBR OUTWD

*****************************************************************************
OUT RL

OUTWD

OUTPUT FIRST WORD TO BUFFER

OUTWD

OUTPUT SECOND WORD TO BUFFER

RETURN
E- OUTSW

TEM- DATA GENERAL ECLIPSE

GRAMMER- J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

POSE-
THIS ROUTINE INPUTS A TOKEN, VERIFIES THAT IT IS OF THE
CORRECT TYPE, AND WRITES IT OUT TO THE OUTPUT BUFFER.

GE-
CALLED BY PGEN AND MULGEN TO OUTPUT A SWITCH PARAMETER.

LING SEQUENCE-
NONE

MON STORAGE-
1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG
2) LEXCL
   LTOKTY - TOKEN TYPE
   LTOKVA - TOKEN VALUE IN INTEGER FORMAT

ERNAL REFERENCES-
SUBR GETNEX
SUBR ERMSG
SUBR OUTHEX

*******************************************************************************
OUTTOL

DATA GENERAL ECLIPSE

J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

This routine inputs a token, verifies that it is of the correct type, and writes it out to the output buffer.

Called by PGEN and MULGEN to output a tolerance parameter

External Sequence:
None

Common Storage:
1) GLOBL
   MEOF - flag indicating end of the source file
   MERROR - global error flag
2) LEXCL
   LTOKTY - token type
   RTOKVA - token value in real format

External References:
SUBR GETNEX
SUBR ERMSG
SUBR OUTRL
OUTTOL

YES
ERROR FLAG SET?
NO
GET NEXT TOKEN

YES
TOKEN TYPE ≠ 11
NO
ERROR - NOT A TOLERANCE PARAMETER

OUTPUT REAL TO OUTPUT BUFFER

RETURN
THESE ROUTINE INPUTS A TOKEN, VERIFIES THAT IT IS OF THE
CORRECT TYPE, AND WRITES IT OUT TO THE OUTPUT BUFFER.

CALLED BY PGEN AND MULGEN TO OUTPUT A VALUE PARAMETER.

NONE

1) GLOBL
   MEOF - FLAG INDICATING END OF THE SOURCE FILE
   MERROR - GLOBAL ERROR FLAG

2) LEXCL
   LTOKTY - TOKEN TYPE
   RTOKVA - TOKEN VALUE IN REAL FORMAT

SUBR GETNEX
SUBR ERMSG
SUBR OUTRL

*****************************************************************************
E- OUTWD (IWORD)

TEM- DATA GENERAL ECLIPSE

GRAMMER- J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY

POSE- THIS ROUTINE ENCODES THE VALUE OF IWORD INTO HEX ASCII AND STORES THE VALUE INTO THE NEXT TWO AVAILABLE SPACES IN THE OUTPUT BUFFER.

GEN- CALLED TO OUTPUT A WORD OF OBJECT CODE.

LING SEQUENCE- IWORD - INPUT PARAMETER - WORD CONTAINING THE OBJECT CODE.

MON STORAGE- NONE

VERNAL REFERENCES- SUBR OUTLOW
SUBR OUTHII

************************************************************************************
PARAM (ITYPE, MAXVAL)

DATA GENERAL ECLIPSE

J. E. DOSS, ENGINEERING EXPERIMENT STATION,
GEORGIA INSTITUTE OF TECHNOLOGY

This routine processes P-code parameters whose symbolic formats consist of a single letter followed by a number. The number is checked to be in the proper range.

Called by GETNEX to process single letter P-code parameters:
- Gain specifications
- Conditioning specifications
- Register identifiers

ITYPE - Input parameter - the token type of the P-code parameter.
MAXVAL - Input parameter - the maximum allowable value for the number.

MON STORAGE:
1) GLOBL
   MERROR - Global error flag
2) LEXCL
   LTOKTY - Token type
   LTOKVA - Token value in integer format
   INCHAR - Number of characters in token string

ternal references:
  SUBR ERMSG
  Func RCONV

******************************************************************************
THIS ROUTINE ASSEMBLES THE P-CODE SOURCE FILE PRODUCED FROM THE COMPILER INTO A HEX ASCII-ENCODED BINARY FORMAT. INITIALLY 'FIRST' IS CALLED TO GET THE FIRST SYMBOL OF A P-CODE STATEMENT, I.E. EITHER A P-CODE OPERATOR OR A LABEL DECLARATION. DEPENDING OF THE TYPE OF THE FIRST SYMBOL, ROUTINES ARE CALLED TO EITHER GENERATE OBJECT P-CODE OR TO UPDATE THE LABEL TABLES.

NOTE THAT AN ASSEMBLER DIRECTIVE (.SEC OR .SUB) SPECIFIES A NEW SECTION OR SUBSECTION AND INITIATES THE FOLLOWING ACTIONS:

1) IF THE LABEL TABLES CONTAIN ANY UNRESOLVED LABEL REFERENCES FROM THE PREVIOUS SUBSECTION, ERROR MESSAGES ARE GENERATED.
2) THE LABEL TABLES ARE REINITIALIZED.
3) THE BUFFER CONTAINING THE PREVIOUS SUBSECTION'S OBJECT P-CODE IS WRITTEN TO THE OUTPUT FILE.

THIS PROCEDURE PROHIBITS ANY REFERENCES TO LABELS DECLARED OUTSIDE OF THE CURRENT SUBSECTION.
THIS FUNCTION IS CALLED TO DETERMINE IF A TOKEN IS A P-CODE OPERATOR OR AN ASSEMBLER DIRECTIVE. A BINARY SEARCH IS PERFORMED ON THE TABLE OF P-CODE OPERATORS USING THE TOKEN IN COMMON.

FOR REGISTER/IMMEDIATE DATA P-CODE OPERATORS, THE FUNCTION MUST ALSO DECIDE WHETHER OR NOT THE OPERATOR REFERS TO A REGISTER OR IMMEDIATE DATA. IT DOES THIS BY LOOKING AHEAD AT THE NEXT TOKEN; I.E. CALLING GETNEX AND SETTING ISKIP TO TRUE.

NOTE THAT ASSEMBLER DIRECTIVES ARE TREATED SIMILARLY TO P-CODE OPERATORS. THE DIFFERENCE IS THAT DIRECTIVES ARE ONLY USED TO GENERATE INFORMATION LINES WITHIN THE OBJECT CODE (I.E. THE BEGINNING OF SECTION AND SUBSECTION LINES).

CALLED BY FIRST TO IDENTIFY A TOKEN AS A P-CODE OPERATOR OR ASSEMBLER DIRECTIVE.

PARAMETERS:

PCODE - FUNCTION REFERENCE - SET TO TRUE IF THE TOKEN IS A P-CODE OPERATOR.
IOPCOD - OUTPUT PARAMETER - P-CODE OPERATOR'S OPCODE.
ITYPE - OUTPUT PARAMETER - P-CODE OPERATOR'S TYPE.

MON STORAGE:

1) GLOBL
   MERROR - GLOBAL ERROR FLAG
2) LEXCL
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT
   ISKIP - SET TO TRUE WHEN A LOOKAHEAD HAS BEEN DONE ON THE NEXT TOKEN
3) CODES
   POIDS (5,45) - P-CODE OPERATOR IDENTIFIERS
   POPCD (45) - P-CODE OPERATOR OPCODES
   PTYPE (45) - P-CODE OPERATOR TYPES
   TRANS (7) - ALTERNATE OPCODES FOR P-CODES WHICH CAN REFERENCE EITHER REGISTERS OR IMMEDIATE DATA
4) LITCH
   KBLANK - LITERALLY '

RNLAL REFERENCES:

SUBR GETNEX

RKS-

THE P-CODE OPERATOR TYPES (PTYPE) HAVE THE FOLLOWING DEFINITION:

- 0 - NO PARAMETERS
- 1 - HAS A STRING LITERAL PARAMETER
- 2 - HAS A SINGLE VALUE PARAMETER
- 0 - HAS A SWITCH PARAMETER
- 6 - HAS MULTIPLE PARAMETERS
- 7 - INDICATES AN ASSEMBLER DIRECTIVE

***********************************************************************
**POSE**

This routine is called when a valid P-code operator or assembler directive is identified. It calls routines to check the operator's parameters for validity and to generate the object code.

**GE**

Called by PASSM to generate the object code for a P-code statement.

**LING SEQUENCE**

None

**MON STORAGE**

1) GLOBL

- GERROR - Global error flag

2) LEXCL

- LTOKVA - Token value in integer format
- LPTYPE - P-code operator type

**EXTERNAL REFERENCES**

- SUBR OUTLOW
- SUBR GENREG
- SUBR OUTLAB
- SUBR OUTLIT
- SUBR OUTVAL
- SUBR OUTSW
- SUBR MULGEN
- SUBR ADIREC
- SUBR ERMSG

**ARKS**

LPTYPE can have the following values:

- 0 - Operator requires no parameters
- 1 - Operator requires a string literal
- 2 - Operator requires a value
- 3 - Operator requires a label reference
- 4 - Operator requires a register reference
- 5 - Operator requires a switch reference
- 6 - Operator requires multiple parameters of varying types
- 7 - Indicates an assembler directive
THIS ROUTINE CONVERTS A NUMBER FROM THE UNPACKED ASCII FORMAT TO BINARY. INBUF IS SCANNED BEGINNING AT INITC FOR VALID NUMERIC CHARACTERS OR A '.}'. THE SCAN IS ENDED BY ANY CHARACTER OTHER THAN NUMERIC OR '.}'. LASTNR IS SET TO THE LAST VALID NUMERIC CHARACTER READ. IF NO NUMERIC CHARACTERS WERE DETECTED, NONUMR IS SET TO TRUE. NOTE THAT LASTNR WILL EQUAL INCHAR IF NO NON-NUMERIC CHARACTERS ARE FOUND.

INCASED TO CONVERT NUMBERS IN UNPACKED ASCII TO BINARY.

SEQUENCE-
INITC - INPUT PARAMETER - SPECIFIES THE LOCATION OF INITIAL NUMERIC CHARACTER IN INBUF.
LASTNR - OUTPUT PARAMETER - SPECIFIES THE LAST VALID NUMERIC CHARACTER IN INBUF (INCLUDING '.').
INTFLG - OUTPUT PARAMETER - SET TO TRUE IF THE RETURNED VALUE WAS AN INTEGER.
NONUMR - OUTPUT PARAMETER - SET TO TRUE WHEN NO NUMERIC CHARACTERS WERE DETECTED IN THE STRING.

STORAGE-
1) LEXCL
INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
INCHAR - NUMBER OF CHARACTERS IN TOKEN
2) LITCH
KZERO - LITERALLY '0'
KNINE - LITERALLY '9'
KPERID - LITERALLY '.'

REFERENCES-
FUNC ISHFT (IWORD,NUMBITS) - DATA GENERAL SUPPLIED.
RCONV

INDEX = φ
RSUM = φ
INTFLG = True,

J = INITIAL CHAR

INDEX(?) NUMERIC?

INDEX(?) A DECIMAL POINT?

INVALID CHAR

LASTNR ← J - 1

CONVERT TO REAL NUMBER

RETURN

J ← J + 1

YES

J ≤ # OF CHARS

LASTNR ← J - 1

CONVERT TO REAL NUMBER

RETURN

YES

INDEX(?) NUMERIC?

RSUM ← (RSUM * 10.0) + NUMSTR

INDEX = J

INTFLG ← False

INDEX(?) A DECIMAL POINT?

INDEX(?) NUMERIC?
**STORE (CHAR)**

**ITEM-**
DATA GENERAL ECLIPSE

**GRAMMER-**
J. E. DOSS, ENGINEERING EXPERIMENT STATION, GEORGIA INSTITUTE OF TECHNOLOGY

**POSE-**
THIS ROUTINE STORES THE CHARACTER GIVEN IN THE INCHAR ELEMENT OF INBUF. NOTE THAT CHAR CONSISTS OF THE CHARACTER LEFT JUSTIFIED AND A BLANK. THE CHARACTER IS ALSO STORED IN LPCKBF IN PACKED FORMAT. LPCKBF IS USED BY THE SEARCH ROUTINES.

**CE-**
CALLED BY GETOK TO STORE THE CHARACTERS OF A TOKEN.

**LING SEQUENCE-**
CHAR - INPUT PARAMETER - THE CHARACTER TO BE STORED LEFT JUSTIFIED PADDED WITH A BLANK

**MON STORAGE-**
1) LEXCL
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
   TNCHAR - NUMBER OF CHARACTERS IN TOKEN
   LPCKBF(35) - TOKEN STRING IN PACKED FORMAT

**ERNAL REFERENCES-**
FUNC ISHFT (IWORD, NUMBITS) - DATA GENERAL SUPPLIED

******************************************************************************
STORE

STORE THE CHAR IN THE UNPacked BUFFER

NUM, OF CHAR $ \geq 70$

STORE THE CHAR IN THE Packed BUFFER

RETURN
NAME- VALTOL

STEM- DATA GENERAL ECLIPSE

PROGRAMMER- J. E. DOSS, ENGINEERING EXPERIMENT STATION,
              GEORGIA INSTITUTE OF TECHNOLOGY

PURPOSE-
THIS ROUTINE PROCESSES VALUE AND TOLERANCE P-CODE
PARAMETERS. THE TOKEN STRING IS CHECKED FOR VALIDITY
AND CONVERTED TO A FLOATING POINT NUMBER. NOTE THAT
TOLERANCES ARE ALWAYS PRECEDED BY A '+-'.

USAGE-
CALLED BY GETNEX TO PROCESS VALUES AND TOLERANCES.

CALLING SEQUENCE-
NONE

COMMON STORAGE-
1) GLOBL
   MERROR - GLOBAL ERROR FLAG

2) LEXCL
   LTOKTY - TOKEN TYPE
   RTOKVA - TOKEN VALUE IN REAL FORMAT
   INBUF(255) - TOKEN STRING IN UNPACKED FORMAT
   INCHAR - NUMBER OF CHARACTERS IN TOKEN

3) LITCH
   KZERO - LITERALLY '0'
   KNINE - LITERALLY '9'
   KPLUS - LITERALLY '+'
   KMINUS - LITERALLY '-'

EXTERNAL REFERENCES-
   FUNC RCONV
   SUBR ERMSG
REPORT ON THE ANALYSIS PHASE  
AUTOTHROTTLE TEST PROCEDURE COMPILER  

PROJECT A-2339  

BY  

J. E. DOSS  
E. S. ACREE  
C. A. BATTERMAN  
S. N. COLE  

COMPUTER SCIENCE & TECHNOLOGY DIVISION  
ENGINEERING EXPERIMENT STATION  
GEORGIA INSTITUTE OF TECHNOLOGY  
ATLANTA, GEORGIA 30332  

Prepared for  

MARCONI AVIONICS, INC.  
4500 N. Shallowford Rd.  
Atlanta, Georgia 30338
1. Introduction

The Computer Science and Technology Division of the Radar and Instrumentation Laboratory is performing a 5-month project to develop software for Marconi Avionics. This software will compile test specification statements from an Acceptance Test Procedure (ATP) in English-language form to P-code. The software will be developed in a high-order language to run on a minicomputer, and its output files containing the resultant P-code will reside on the minicomputer's disk or tape. It is presumed that other software organizations will be concurrently developing companion software to transport the compiled P-code to the Autothrottle Semi Automatic Test Equipment being built by Marconi. The companion software would include routines to provide a data interface from the minicomputer to the TI-990 microprocessor and a P-code interpreter to provide a signal interface between the TI-990 and the Automatic Test Equipment.

The effort to develop the software consists of two phases -- analysis and implementation. This report presents the results of the analysis phase which consists of the following four tasks:

1. Define and document the pseudo code which will be the output of the compiler.
2. Define and document the syntax rules that the compiler will implement.
3. Define a computer-compatible description of the autothrottle test equipment.
4. Provide an estimate of the percentage of ATP statements which will be compilable using the defined syntax rules and P-code.
The following four sections of this report delineate the results for each task. Upon approval of the results of the analysis phase by Marconi Avionics, the implementation phase may commence.
2.0 Introduction - Definition of P-Code

The process of converting English language test statements into an automated test sequence can be greatly simplified through the use of an intermediate P-code machine. Defining a virtual computer tailored specifically to a class of applications can simplify and speed system implementation. Using a P-code interpretive approach for automatic testing of autothrottle computers is likely to result in a flexible, general purpose test system.

The primary goal of the virtual p-machine is to achieve the flexibility to handle the various test sequences required for testing Marconi autothrottle computers. Test sequences may be categorized into instructions which occur frequently and a set of infrequent special test cases. Handling both classes of test instructions efficiently may be accomplished using two levels of P-code.

High Level P-Code

High level P-code instructions were designed to handle the most frequent test cases using a minimum of P-code instruction space. Included in this category are the instructions to measure and check voltages, open and close switches, set a voltage, wait a period of time, and check a voltage transition. These categories occur frequently, hence it is desirable to have P-codes available which directly accomplish the required tasks. High level instructions could be coded using the low level P-code instruction set. However, for maximum speed and efficiency the high level instructions should be used where possible. Note that in this documentation many of the high level instruction sequences are actually specified in terms of low level instructions. This approach is used only to add clarity. High level
instructions should be hard-coded within the interpreter. The instructions can utilize many of the same utilities used to implement the low level instructions. Making low level instructions accessible should not add a significant amount of overhead since most low level functions must be provided to implement the high level codes.

**Low Level P-Code**

In order to achieve the required test system flexibility it is appropriate to specify a set of low level P-code instructions. The low level instructions are used only to code special cases which cannot be handled by the high level instructions.

Low level P-codes were chosen to provide a minimum set of codes which can be used to implement most automatic test applications. Included are the P-codes to accomplish simple looping control, comparisons, error processing, console I/O and the various arithmetic functions. Since most test applications do not require expression evaluation and subroutine linkages the P-machine does not include stack oriented instructions. These facilities could be added later if needed for future applications.
2.1 P-Machine Implementation

Marconi's automatic test P-machine will be implemented as a 32 bit virtual computer. The actual implementation of the P-machine will utilize a TI-9900 16-bit minicomputer. An interpretive program emulates the 32 bit pseudo machine.

The pseudo machine itself is implemented as a general purpose 32 bit computer. A 32 bit accumulator is used as the source and destination for the results of most instructions. Sixteen 32 bit general purpose registers can be used for temporary data storage. A 16 bit program counter is used as a pointer for fetching instructions. The status register records the results of compare and arithmetic instructions. String registers maintain test section data and console input. See Figure 3.1.

The sixteen general registers are categorized into user and system registers. Registers 0 to 9 are reserved for the user as temporary storage registers. Registers 10 through 15 are used by the system to pass data between P-codes and have the following definition:

- Register 10 - Raw A/D Data (i.e. a 12 bit word)
- Register 11 - A/D Data converted to the real 32 bit format.
- Register 12 - The correct value for the voltage at a pin.
- Register 13 - The tolerance associated with the value in Register 12.
- Register 14 - Initial value for change P-code.
- Register 15 - Temporary for adjust device P-code.
Status Register (S)
EQ LT GT Z M

Program Counter (PC)
16 bits

Accumulator (A)
32 bits

General Purpose Registers (RO - R15)
32 bits

String Input Buffer (SI)
80 bytes

Test Section Number (TS)
20 bytes

Figure 2.1 Pseudo Machine Registers
2.2 **Number Representation**

The automatic test P-code interpreter will utilize 32 bit floating point arithmetic for all calculations, thus providing all the accuracy needed by the ATP. The compiler will produce 32 bit values in the P-code which correspond to the floating point format used by the TI 9900 microprocessor.

2.3 **P-code Formats**

P-codes are specified in two categories. The majority are numbered sequentially from 01 hex to 2A hex. These may be implemented easily with a branch table. All P-codes which reference the p-machine registers have the high order bit of the opcode byte set. Register related instructions all have a decimal value greater than 127. Register reference p-codes consist of two nibbles. The high order nibble specifies the operation to be performed, while the low order nibble contains a register reference. In all cases the accumulator is used to obtain operand 1 and the register reference (0 . . F) hex is used for operand 2. The destination of the instruction is the accumulator.

<table>
<thead>
<tr>
<th>Register reference instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nibble 1</td>
</tr>
<tr>
<td>8--&gt;E</td>
</tr>
<tr>
<td>Op code</td>
</tr>
</tbody>
</table>

2.4 **Branch Instructions**

The branch instructions allow the P-machine to transfer control to either
a standard procedure or to another section of the P-code. The branch (BRA) instruction causes an unconditional transfer of program control. All other instructions in this category cause a change in the P-code execution sequence only if the appropriate bit in the status register is set (cleared). Each branch instruction includes a single byte opcode and a two byte offset. The two byte offset specifies where program control should transfer if the specified condition is true. If the high order bit of the offset is set, then the low order bits indicate which standard procedure to execute. When the high order bit is cleared then the low order bits contain a offset from the current PC.

The branch opcodes are represented:

\[
\begin{array}{c|c}
\text{8 bit opcode} & x \ y (15 \text{ bits}) \\
\hline
\end{array}
\]

If \( x = 0 \) then \( y = \) offset in bytes from PC
If \( x = 1 \) then \( y = \) standard procedure reference number

2.5 Standard Procedures

SP0 - Record successful completion of test subsection contained in the test section register (TS). Begin execution of next subsection. Note entire subsection must be memory resident.

SP1 - Unassigned.

SP2 - Normal test termination. Ready equipment for next test. The BRA SP2 and STOP opcodes have identical effect.

SP3 - Abnormal subsection termination. ADC value out of tolerance. Print
results contained in registers R10 - R13 and appropriate id's.

SP4 - Abnormal subsection termination. Unable to set DAC to proper value. Print results contained in registers R10 - R13 and appropriate id's.

SP5 - Abnormal subsection termination. Monitored voltage out of tolerance. Print elapsed time and results in registers R10 - R13 along with appropriate id's.

SP6 - Abnormal subsection termination. Voltage of a transition is out of tolerance. Print elapsed time, registers R10 - R13, and appropriate id's.

SP7 - Print normal results of a monitor instruction.

SP8 - Print normal results of an ADC measurement. Results contained in registers R10 - R13.

SP9 - Print normal results of a transition.

SP10 - Restart section.

SP11 - Terminate testing.

SP12 - Abnormal subsection termination. ADC value changed incorrectly. Print results contained in registers R10 - R14 along with appropriate id's.

SP13 - Abnormal subsection termination. Unable to set synchro to correct value. Print desired value and tolerance from registers R12 and R13.

SP14 - Abnormal subsection termination. Operator measured value is out of tolerance.
2.6 P-codes

The following p-codes will be implemented:

<table>
<thead>
<tr>
<th>OPCODE</th>
<th>P-CODE</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>01H</td>
<td>SETTST</td>
<td>SET TEST SECTION NUMBER</td>
</tr>
<tr>
<td>02H</td>
<td>OUTSTR</td>
<td>OUTPUT ASCII STRING</td>
</tr>
<tr>
<td>03H</td>
<td>GETSTR</td>
<td>INPUT ASCII STRING</td>
</tr>
<tr>
<td>04H</td>
<td>CMPSTR</td>
<td>COMPARE ASCII STRING</td>
</tr>
<tr>
<td>05H</td>
<td>OUTDEC</td>
<td>OUTPUT DECIMAL NUMBER</td>
</tr>
<tr>
<td>06H</td>
<td>GETDEC</td>
<td>INPUT DECIMAL NUMBER</td>
</tr>
<tr>
<td>07H</td>
<td>SETCLK</td>
<td>SET REAL TIME CLOCK</td>
</tr>
<tr>
<td>08H</td>
<td>GETCLK</td>
<td>READ REAL TIME CLOCK</td>
</tr>
<tr>
<td>09H</td>
<td>BRA</td>
<td>BRANCH UNCONDITIONALLY</td>
</tr>
<tr>
<td>0AH</td>
<td>BLT</td>
<td>BRANCH IF LESS THAN</td>
</tr>
<tr>
<td>0BH</td>
<td>BGT</td>
<td>BRANCH IF GREATER THAN</td>
</tr>
<tr>
<td>0CH</td>
<td>BLE</td>
<td>BRANCH IF LESS THAN OR EQUAL</td>
</tr>
<tr>
<td>0DH</td>
<td>BGE</td>
<td>BRANCH IF GREATER THAN OR EQUAL</td>
</tr>
<tr>
<td>0EH</td>
<td>BEQ</td>
<td>BRANCH IF EQUAL</td>
</tr>
<tr>
<td>0FH</td>
<td>BNE</td>
<td>BRANCH IF NOT EQUAL</td>
</tr>
<tr>
<td>10H</td>
<td>BM</td>
<td>BRANCH IF MINUS</td>
</tr>
<tr>
<td>11H</td>
<td>BP</td>
<td>BRANCH IF POSITIVE</td>
</tr>
<tr>
<td>12H</td>
<td>BZ</td>
<td>BRANCH IF ZERO</td>
</tr>
<tr>
<td>13H</td>
<td>BNZ</td>
<td>BRANCH IF NOT ZERO</td>
</tr>
<tr>
<td>14H</td>
<td>ABS</td>
<td>TAKE ABSOLUTE VALUE OF A</td>
</tr>
<tr>
<td>15H</td>
<td>NEG</td>
<td>NEGATE A</td>
</tr>
<tr>
<td>16H</td>
<td>CIA</td>
<td>CLEAR A</td>
</tr>
<tr>
<td>17H</td>
<td>STOP</td>
<td>NORMAL TERMINATION</td>
</tr>
<tr>
<td>18H</td>
<td>CMP RX</td>
<td>COMPARE A WITH REGISTER</td>
</tr>
<tr>
<td>19H</td>
<td>CMP #</td>
<td>COMPARE A WITH IMMEDIATE DATA</td>
</tr>
<tr>
<td>20H</td>
<td>STA RX</td>
<td>STORE A IN REGISTER</td>
</tr>
<tr>
<td>21H</td>
<td>LDA RX</td>
<td>LOAD A FROM REGISTER</td>
</tr>
<tr>
<td>22H</td>
<td>LDA #</td>
<td>LOAD A WITH IMMEDIATE DATA</td>
</tr>
<tr>
<td>23H</td>
<td>SUB RX</td>
<td>SUBTRACT REGISTER FROM A</td>
</tr>
<tr>
<td>24H</td>
<td>SUB #</td>
<td>SUBTRACT IMMEDIATE FROM A</td>
</tr>
<tr>
<td>25H</td>
<td>ADD RX</td>
<td>ADD REGISTER TO A</td>
</tr>
<tr>
<td>26H</td>
<td>ADD #</td>
<td>ADD IMMEDIATE TO A</td>
</tr>
<tr>
<td>27H</td>
<td>MULT RX</td>
<td>SIGNED MULTIPLY OF A BY REGISTER</td>
</tr>
<tr>
<td>28H</td>
<td>MULT #</td>
<td>SIGNED MULTIPLY OF A BY IMMEDIATE</td>
</tr>
<tr>
<td>29H</td>
<td>DIV RX</td>
<td>SIGNED DIVIDE OF A BY REGISTER</td>
</tr>
<tr>
<td>30H</td>
<td>DIV #</td>
<td>SIGNED DIVIDE OF A BY IMMEDIATE</td>
</tr>
<tr>
<td>31H</td>
<td>WAIT</td>
<td>WAIT THE SPECIFIED TIME</td>
</tr>
<tr>
<td>32H</td>
<td>OPEN</td>
<td>OPEN SPECIFIED SWITCH</td>
</tr>
<tr>
<td>33H</td>
<td>CLOSE</td>
<td>CLOSE SPECIFIED SWITCH</td>
</tr>
<tr>
<td>34H</td>
<td>INPUT</td>
<td>READ THE A/D</td>
</tr>
<tr>
<td>35H</td>
<td>CONV</td>
<td>CONVERT THE A/D DATA TO REAL</td>
</tr>
<tr>
<td>36H</td>
<td>MEAS</td>
<td>TAKE A MEASUREMENT</td>
</tr>
<tr>
<td>37H</td>
<td>SETDEV</td>
<td>SET A DAC OR D/S TO A VALUE</td>
</tr>
<tr>
<td>38H</td>
<td>CHECK</td>
<td>CHECK THE OUTPUT OF A DAC</td>
</tr>
<tr>
<td>39H</td>
<td>MONV</td>
<td>MONITOR A PIN'S VOLTAGE</td>
</tr>
<tr>
<td>3AH</td>
<td>TRANS</td>
<td>MONITOR A PIN FOR A TRANSITION</td>
</tr>
<tr>
<td>3BH</td>
<td>INCD</td>
<td>INCREMENT DAC OR D/S VALUE</td>
</tr>
<tr>
<td>3CH</td>
<td>DECD</td>
<td>DECREMENT DAC OR D/S VALUE</td>
</tr>
</tbody>
</table>

-10-  Rev. 1.0
2AH  CHANGE  CHECK FOR ADC VALUE CHANGE
2BH  ADJDEV  ADJUST THE SPECIFIED DEVICE
2CH  READEV  READ THE DEVICE'S CURRENT VALUE

NOTE: 1) 'X' in the opcode field refers to one of the sixteen general registers.
2) # refers to immediate data.

In the following sections, the P-codes are defined with the following parameters:

<table>
<thead>
<tr>
<th>P-CODE PARAMETER</th>
<th>SYMBOLIC FORMAT</th>
<th>OBJECT FORMAT</th>
<th>REMARKS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) &lt;ADC address&gt;</td>
<td>10 character pin ID</td>
<td>16 bit word</td>
<td>DIGITAL/SYNCHRO, movement in a clockwise direction.</td>
</tr>
<tr>
<td>2) &lt;device address&gt;</td>
<td>'DS0'-'DS255'</td>
<td>Bit 11 = 1, Bit 8 = 0, Bit 7-0 = VALUE</td>
<td>Bit 11 = 1, Bit 8 = 0, Bit 7-0 = VALUE</td>
</tr>
<tr>
<td></td>
<td>'DSOC'-'DS255C'</td>
<td>Bit 11 = 1, Bit 8 = 1, Bit 7-0 = VALUE</td>
<td>Bit 11 = 1, Bit 8 = 1, Bit 7-0 = VALUE</td>
</tr>
<tr>
<td></td>
<td>'DC0'-'DC255'</td>
<td>Bit 9 = 1, Bit 7-0 = VALUE</td>
<td>Bit 9 = 1, Bit 7-0 = VALUE</td>
</tr>
<tr>
<td></td>
<td>'AC0'-'AC255'</td>
<td>Bit 10 = 1, Bit 8 = 0, Bit 7-0 = VALUE</td>
<td>Bit 10 = 1, Bit 8 = 0, Bit 7-0 = VALUE</td>
</tr>
<tr>
<td></td>
<td>'ACOC'-'AC255C'</td>
<td>Bit 10 = 1, Bit 8 = 1, Bit 7-0 = VALUE</td>
<td>Bit 10 = 1, Bit 8 = 1, Bit 7-0 = VALUE</td>
</tr>
<tr>
<td>3) &lt;gain spec&gt;</td>
<td>'G1'-'G4'</td>
<td>8 bits</td>
<td>A/D gain</td>
</tr>
<tr>
<td>4) &lt;conditioning spec&gt;</td>
<td>'C1'-'C16'</td>
<td>8 bits</td>
<td>A/D scale factor</td>
</tr>
<tr>
<td>5) &lt;switch address&gt;</td>
<td>'S0'-'S4096'</td>
<td>16 bits</td>
<td>A/D scale factor</td>
</tr>
<tr>
<td>6) &lt;value&gt;</td>
<td>+99999.999</td>
<td>32 bits</td>
<td>floating point</td>
</tr>
<tr>
<td>7) &lt;tolerance&gt;</td>
<td>+99999.999</td>
<td>32 bits</td>
<td>floating point</td>
</tr>
<tr>
<td></td>
<td>The '+' is required.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8) &lt;time&gt;</td>
<td>99999.999</td>
<td>32 bits</td>
<td>seconds</td>
</tr>
<tr>
<td>9) &lt;step size&gt;</td>
<td>+99.999</td>
<td>32 bits</td>
<td>step value - volts, deg, etc.</td>
</tr>
</tbody>
</table>
2.6.1 SETTST – Set the test section register (TS) with immediate ASCII data.

Opcode: 01 hex

Results: The immediate ASCII data following the SETTST opcode is copied into the TS register until the termination character is detected. The termination character is a null (0) byte.

Status bits affected: None

2.6.2 OUTSTR – Output a string of ASCII data to the currently active I/O devices.

Opcode: 02 hex

Results: Copies the ASCII following the opcode byte to active I/O devices until a null (0) termination character is detected.

Status bits affected: None

2.6.3 GETSTR – Get a string of input from the console input device into the string input buffer (SI).

Opcode: 03 hex

Results: Retrieves characters from the operator console until a termination character is received. All characters are copied into the string input (SI) buffer. Any termination character causes the instruction to terminate with the remainder of the SI buffer blank filled.

Termination chars: carriage return line feed or any unprintable char.

Status bits affected: None

2.6.4 CMPSTR – Compare the immediate ASCII string with the string contained in the string input buffer (SI).

Opcode: 04 hex

Results: The immediate ASCII string is compared byte by byte with the contents of the string buffer. Characters are compared until the termination character is encountered within the immediate string or non-matching characters are detected.

Example: A CMPSTR 'Y' instruction would match string buffer data of 'Y', 'YES', 'YEAH', etc. A CMPSTR 'YES' instruction would match any string buffer data containing
'YES' as the first three characters.
Status bits affected: EQ

2.6.5 OUTDEC - Output the contents of the A register in ASCII to the currently active I/O devices.
Opcode: 05 hex
Results: Converts the contents of the A register to a decimal ASCII string. The string is copied to the active I/O devices.
Status bits affected: None

2.6.6 GETDEC - Retrieve and convert numeric data from the operators console.
Opcode: 06 hex
Results: A <--- (converted ASCII numeric data) Retrieves numeric characters from the operator console until a termination char is received. The ASCII data is converted to 32 bit binary and stored in the A register.
Termination chars: Carriage return.
Status bits affected: Z, M

2.6.7 SETCLK - Set the test system clock with the contents of A.
Opcode: 07 hex
Results: (CLOCK)<---A
Status bits affected: None

2.6.8 GETCLK - Set the accumulator (A) with the contents of the system clock.
Opcode: 08 hex
Results: A<---(CLOCK)
Status bits affected: Z, M
2.6.9 BRA - Branch unconditionally
   Opcode: 09 hex
   Results: PC <- PC + offset
   Status bits affected: None

2.6.10 BLT - Branch if less than
   Opcode: 0A hex
   Results: If status bit 'LT' is set, then PC <- PC + offset.
   Status bits affected: None

2.6.11 BGT - Branch if greater than
   Opcode: 0B hex
   Results: If status bit 'GT' is set, then PC <- PC + offset.
   Status bits affected: None

2.6.12 BLE - Branch if less than or equal
   Opcode: 0C hex
   Results: If status bit 'EQ' or status bit 'LT' is set, then PC <- PC + offset.
   Status bits affected: None

2.6.13 BGE - Branch if greater than or equal
   Opcode: 0D hex
   Results: If status bit 'GT' or status bit 'EQ' is set, then PC <- PC + offset.
   Status bits affected: None
2.6.14 BEQ - Branch if equal.

   Opcode: 0E hex
   Results: If status bit 'EQ' is set, then PC ← PC + offset.
   Status bits affected: None

2.6.15 BNE - Branch if not equal.

   Opcode: 0F hex
   Results: If status bit 'EQ' is cleared, then PC ← PC + offset.
   Status bits affected: None

2.6.16 BM - Branch if minus.

   Opcode: 10 hex
   Results: If status bit 'M' is set, then PC ← PC + offset.
   Status bits affected: None

2.6.17 BP - Branch if positive.

   Opcode: 11 hex
   Results: If status bit 'M' is cleared, then PC ← PC + offset.
   Status bits affected: None

2.6.18 BZ - Branch if zero.

   Opcode: 12 hex
   Results: If status bit 'Z' is set, then PC ← PC + offset.
   Status bits affected: None

2.6.19 BNZ - Branch if non-zero.

   Opcode: 13 hex
   Results: If status bit 'Z' is cleared, then PC ← PC + offset.
   Status bits affected: None
2.6.20 **ABS** - Take absolute value of register A.

**Opcode:** 14 hex

**Results:** A <-- |A|

**Status bits affected:** Z, M

2.6.21 **NEG** - Negate register A.

**Opcode:** 15 hex

**Results:** A <-- -A

**Status bits affected:** Z, M

2.6.22 **CLA** - Clear accumulator (A).

**Opcode:** 16 hex

**Results:** A <-- 0

**Status bits affected:** Z, M

2.6.23 **STOP** - Normal termination of a test.

**Opcode:** 17 hex

**Results:** PC <-- Standard Procedure 2.

**Status bits affected:** None

2.6.24 **CMP RX** - Compare A with register X.

**Opcode:** 8X hex X = 0..F

**Results:** Compare contents of A with contents of X register and set appropriate status bits.
- If A = RX then 'EQ' set.
- If A < RX then 'LT' set.
- If A > RX then 'GT' set.

**Status bits affected:** EQ, LT, GT
2.6.25 CMP # - Compare A with 32 bit immediate data.

Opcode: 18 hex

Results: Compare contents of A with immediate data and set appropriate status bits.
If A = # data then 'EQ' set.
If A < # data then 'LT' set.
If A > # data then 'GT' set.

Status bits affected: EQ, LT, GT

2.6.26 STA RX - Store A into register X.

Opcode: 9X hex x = 0..F

Results: RX <-- A

Status bits affected: None

2.6.27 LDA RX - Load A from register X

Opcode: AX hex x = 0..F

Results: A <-- RX

Status bits affected: Z, M

2.6.28 LDA # - Load A with immediate data.

Opcode: 19 hex

Results: A <-- (32 bit immediate data)

Status bits affected: Z, M

2.6.29 SUB RX - Subtract register x from A.

Opcode: BX hex X = 0..F

Results: A <-- A - RX

Status bits affected: Z, M
2.6.30 SUB # - Subtract immediate data from A
   Opcode: 1A hex
   Results: A <- A - (32 bit immediate data)
   Status bits affected: Z, M

2.6.31 ADD RX - Add register x to A.
   Opcode: CX hex X = 0..F
   Results: A <- A + RX
   Status bits affected: Z, M

2.6.32 ADD # - Add immediate data to A.
   Opcode: 1B hex
   Results: A <- A + (32 bit immediate data)

2.6.33 MULT RX - Signed multiply A by register x.
   Opcode: DX hex X = 0..F
   Results: A <- A * RX
   Status bits affected: Z, M

2.6.34 MULT # - signed multiply A by immediate data.
   Opcode: 1C hex
   Results: A <- A * (immediate data)
   Status bits affected: Z, M

2.6.35 DIV RX - Signed divide A by register X.
   Opcode: EX hex X = 0..F
   Results: A <- A/RX
   Status bits affected: Z, M
2.6.36 DIV # - Signed divide A by immediate data.

   Opcode:    1D hex
   Results:   A ←← A / (immediate data)
   Status bits affected: A, M

2.6.37 WAIT <time> - wait the specified time.

   Opcode:    1E hex
   Results:   Program execution suspends until the 32 bit immediate time
              reference has elapsed. The time is given in seconds.
   Status bits affected: None

   The WAIT instruction could be written in the low level p-codes as follows:

   CLA
   SETCLK          ; 0 clock
   LOOP:
   GETCLK
   CMP <time>
   BLT LOOP

2.6.38 OPEN <switch address> - Open specified switch

   Opcode:    1F hex
   Results:   The switch specified by the two byte immediate switch
              identifier is opened.
   Status bits affected: None

2.6.39 CLOSE <switch address> - Close specified switch

   Opcode:    20 hex
   Results:   The switch specified by the two byte immediate switch
              identifier is closed.
   Status bits affected: None
2.6.40 INPUT <ADC address>, <gain spec>

Opcode: 21 hex

Results: set the ADC to the specified gain and input a word from the
ADC address into the A register and R10. R10 ←→ A

Status bits affected: Z, M

2.6.41 CONV <gain spec>, <conditioning spec>

Opcode: 22 hex

Results: Convert the contents of a into a 'REAL' value by applying
the gain spec and then the conditioning spec. Leave the
result in A and R11. R11 ←→ A

Status bits affected: Z, M

2.6.42 MEAS - <ADC address>, <gain spec>, <conditioning spec>, <desired
value>, <tolerance>

Opcode: 23 hex

Results - Set the ADC to the appropriate gain, input the ADC word,
convert according to gain and conditioning, and check if
within tolerance. If not in tolerance branch to standard
error processing 3 with registers set:
R10 ←→ (ADC data)
R11 ←→ (converted ADC data)
R12 ←→ (desired value)
R13 ←→ (tolerance)

Status bits affected: Undefined

The MEASURE statement is equivalent to the following p-code sequence:

LDA <desired value>
STA R12
LDA <tolerance>
STA R13
INPUT <gain spec>, <ADC address>
CONV <gain spec>, <conditioning spec>
SUB R12
ABS
CMP R13
BGT SP3
OUTSTR "MEASURED: "
LDA R11
OUTDEC
OUTSTR " WITHIN TOLERANCE"

2.6.43 SETDEV <device address>, <value>, <tolerance>

Opcode: 24 hex

Results: set the specified device to the desired value after applying the implicit scale factor. If the D/S is being addressed, bit 8 of the device address specifies the direction of movement; i.e. if set, counterclockwise; if cleared, clockwise.

\[
\begin{align*}
R12 & \leftarrow \text{ (value)} \\
R13 & \leftarrow \text{ (tolerance)}
\end{align*}
\]

Status bits affected: Undefined

2.6.44 CHECK <ADC address>, <gain spec>, <conditioning spec>

Opcode: 25 hex

Result: checks that the device is set to the value in R12 within tolerance contained in R13. If not in tolerance branch to standard error processing 4 with registers set:

\[
\begin{align*}
R10 & \leftarrow \text{ (ADC data)} \\
R11 & \leftarrow \text{ (converted data)} \\
R12 & \leftarrow \text{ (desired value)} \\
R13 & \leftarrow \text{ (tolerance)}
\end{align*}
\]

Check is most commonly used after a 'SETDEV'.

Status bits affected: Undefined

2.6.45 MCNV - <ADC address>, <gain spec>, <conditioning spec>, <desired value>, <tolerance>, <time>

Opcode: 26 hex

Results: Check that the converted value read from the ADC address remains in tolerance for the time specified. If not, branch to standard error processing 5 with registers set as follows:

\[
\begin{align*}
R10 & \leftarrow \text{ (ADC data)} \\
R11 & \leftarrow \text{ (converted ADC data)} \\
R12 & \leftarrow \text{ (desired value)} \\
R13 & \leftarrow \text{ (tolerance)}
\end{align*}
\]

Status bits affected: undefined after execution
The Monitor p-code can be written with the following low level p-code sequence:

```
LDA <desired value> ; set up for possible error processing
STA R12
LDA <tolerance>
STA R13
CLA
SETCLK
LOOP: INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB R12
ABS
CMP R13
BGT SP5
GETCLK
CMP <time>
BLT LOOP
; OK
```

2.6.46 TRANS <ADC address>, <gain spec>, <conditioning spec>, <value 1>,
<tolerance 1>, <value 2>, <tolerance 2>, <time>, <time
tolerance>

**Opcode:** 27 hex

**Results:** Continuously monitor the specified ADC for value 1. The voltage should remain within tolerance 1 until at least time minus time tolerance. The voltage must change to value 2 within tolerance 2 before time plus time tolerance.

**Status bits affected:** Undefined

The Transition P-code can be written as the following sequence:

```
CLA
SETCLK
CMP <time> ; t=0 ?
BEQ REPEAT 1
REPEAT: INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB <value 1>
ABS
CMP <tolerance 1>
BGT SP6
GETCLK
SUB <time>
CMP < - time tolerance>
BLT REPEAT
```

-22- Rev. 1.0
REPEAT1: GETCLK
    SUB <time>
    CMP <+time tolerance>
    BGT SP6
    INPUT <ADC address>, <gain spec>
    CONV <gain spec>, <conditioning spec>
    SUB <value 2>
    ABS
    CMP <tolerance 2>
    BGT REPEAT1

; OK

2.6.47 INCD <device address>, <step size>

    Increment the value of the DAC or D/S at device address.

    Opcode:  28
    Status bits affected:  None

2.6.48 DECD <device address>, <step size>

    Decrement the value of the DAC or D/S at device address.

    Opcode:  29
    Status bits affected:  None

2.6.49 CHANGE <register>, <ADC address>, <gain spec>, <conditioning spec>,
        <change value>, <tolerance>

    Opcode:  2A hex
    Results:  Set the ADC to the appropriate gain, input the ADC value,
              convert the value using the gain and conditioning specs.
              The converted value shall have changed from the value
              contained in the specified register by 'change value'
              within tolerance. If not in tolerance branch to standard
              error processing 12 with registers set:

              R10 <--- (ADC data)
              R11 <--- (converted ADC data)
              R12 <--- (change value)
              R13 <--- (tolerance)
              R14 <--- (initial value)

    Status bits affected:  Undefined

-23-
The CHANGE statement is equivalent to the following p-code sequence:

```
LDA <register>
STA R14
LDA <change value>
STA R12
LDA <tolerance>
STA R13
INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB R14 ; initial value
SUB R12 ; change amount
ABS
CMP R13 ; tolerance
BGT SP12
```

```
2.6.50 ADJDEV <device address>, <ADC address>, <gain spec>, <conditioning spec>, <value>, <tolerance>, <step size>, <limit value>

Opcode: 2B hex

Results: Adjust the device starting at its current value until the voltage at the ADC address is at 'value' within tolerance. Note that <step size> can be positive or negative. If unable to set the device within tolerance branch to standard error processing with registers set:

R12 <--- (desired value)
R13 <--- (tolerance)

Status bits affected: Undefined

The adjust device p-code can be written using the following low-level p-codes:

```
READEV <device address>
STA R15 ; loop counter
LDA <value>
STA R12
LDA <tolerance>
STA R13
LOOP: INPUT < ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB R12 ; desired value
ABS
CMP R13 ; tolerance
BLE DONE
INCD <device address>, <step size> ; increment/decrement device
LDA R15
ADD <step size>
STA R15
LDA <step size>
BM NEG ; NEGATIVE step size?
LDA R15
```

-24-  Rev. 1.0
CMP <value> ; exceeded maximum value?
BLE LOOP
BRA SP13

NEG:
LDA R15
CMP <limit value> ; exceeded minimum value?
BGE LOOP
BRA SP13

DONE:

2.6.51 READEV <device adress>

Opcode: 2C hex

Results: Load the accumulator with the last value written to the specified device.

Status bits affected: Undefined.
3.0 Introduction - SYNTAX RULES FOR THE ATP

The syntax of the language processed by the ATP compiler is given here in Backus Naur Form or BNF. BNF is a commonly used notation to write grammars which specify the syntax of programming languages.

BNF notation is used as a concise and explicit method of defining the language formally. English is not suitable for formal language definition due to its vagueness and because it leads to ambiguous definitions.

The BNF as the defining language is called the syntactic meta-language. The meta-language is a language which is used to talk about another language or the object language. The object language in this case is the language to be processed by the ATP compiler. Symbols of the object language are called terminals. Symbols of the meta-language which denote strings in the object language are called nonterminals.

Nonterminals in BNF are expressed as names written in corner-brackets '<>'. The syntax of the object language is given as a set of "rewriting rules". These rules consist of a nonterminal on the left-hand side of the rewriting rule, the ::= sign indicating that left-hand side of the rule is replaced by the right-hand side, and the sequence of terminals and/or nonterminals which make up the right-hand side of the rewriting rule.

Within the right-hand side of the rule, alternative ways of rewriting a given nonterminal are separated by a vertical bar, | (read "or"). Also on the right side items written in curly brackets, {}, may be repeated zero or more times. Any item which may be considered optional within a rule has been enclosed in square brackets. For example, in the case of the rewriting rule:

<CLAUSE NUMBER> ::= DIGIT STRING | DIGIT STRING

the nonterminal is CLAUSE NUMBER (enclosed in corner-brackets) with ::= indicating that the nonterminal CLAUSE NUMBER is to be replaced by the
right-hand side of the rule. The right-hand side of the rule consists of a string of appropriate digits separated by periods. In this case the period followed by a digit string may be repeated zero or more times giving clause numbers of the form 5, 5.5, 5.4.3 etc.

In the case of the STATEMENT rewriting rule (see the BNF) the nonterminal <STATEMENT> may be replaced by any one of the types of statements found on the right-hand side of the rule. Here, <STATEMENT> may be replaced by an <ADJUST STATEMENT> or by an <INCREASE STATEMENT> or by a <DECREASE STATEMENT> and so on.

Not every form of every statement type has been kept in the object language. One of a kind statements can easily be converted to the standard format developed here, and their omission from the language will help to avoid overburdening the language and the compiler which must process it. Variations on the standard statement types have also been limited for the same reason. The majority of statements in the ATP should be compilable.
<TEST PROCEDURE> ::= <SECTION> {<SECTION>}

<SECTION> ::= BEGIN <SECTION NAME> {<CLAUSE>} END

<SECTION NAME> ::= <SECTION NUMBER> CHARACTER STRING

<SECTION NUMBER> ::= DIGIT STRING { . DIGIT STRING}

<CLAUSE> ::= BEGIN <CLAUSE NUMBER> <STATEMENT> {<STATEMENT>} END

<CLAUSE NUMBER> ::= DIGIT STRING . DIGIT STRING { . DIGIT STRING}

<REVISION> ::= / DIGIT STRING /

<STATEMENT> ::= <ADJUST STATEMENT>|<INCREASE STATEMENT>|<DECREASE STATEMENT>|<SET STATEMENT>|<CONNECT STATEMENT>|<DISCONNECT STATEMENT>|<VOLTAGE STATEMENT>|<WAIT STATEMENT>|<OPEN CIRCUIT STATEMENT>|<OPERATOR INTERACTION STATEMENT>|<SAVE STATEMENT>|<VOLTAGE CHANGE STATEMENT>|<START P-CODE STATEMENT>|<STOP P-CODE STATEMENT>|<MONITOR STATEMENT>

-28-
<ADJUST STATEMENT> ::= 
ADJUST SIGNAL AT <PIN#> TO __V [+-__V] 
ADJUST SIGNAL AT <PIN#> IN A <SIGN> DIRECTION UNTIL THE VOLTAGE AT <PIN#> CHANGES TO: MEASURE __V +- __V 
ADJUST SIGNAL AT <PIN#> UNTIL THE VOLTAGE AT <PIN#> IS MEASURE __V +- __V 
ADJUST SIGNAL BETWEEN <PIN#> AND <PIN#> TO __V _Hz [__DEGREES] 
ADJUST SYNCHRO TRANSMITTER TO __DEGREES [CLOCKWISE ; COUNTERCLOCKWISE] 
ADJUST SYNCHRO TRANSMITTER TO __V +- __V AT <PIN#> 
ADJUST SYNCHRO TRANSMITTER UNTIL THE VOLTAGE AT <PIN#> IS: MEASURE __V +- __V 
ADJUST THE VOLTAGE BETWEEN <PIN#> AND <PIN#> TO: MEASURE __V +- __V

<SIGN> ::= POSITIVE ; NEGATIVE

<INCREASE STATEMENT> ::= 
INCREASE SIGNAL AT <PIN#> UNTIL THE VOLTAGE <INCREASE OBJECT> IS: MEASURE __V +- V MAX

<INCREASE OBJECT> ::= BETWEEN <PIN#> AND <PIN#> ; AT <PIN#>

<DECREASE STATEMENT> ::= 
DECREASE SIGNAL AT <PIN#> UNTIL THE VOLTAGE AT <PIN#> IS: MEASURE __V+ __V

<SET STATEMENT> ::= 
SET THE SYNCHRO TRANSMITTER TO __DEGREES +- __DEGREES [CLOCKWISE ; COUNTERCLOCKWISE]
<CONNECT STATEMENT> ::= 
    CONNECT <PIN#> TO <CONNECT OBJECT> ;
    CONNECT <SYNCHRO OUTPUT> TO <PIN#> ;
    CONNECT __V, __HZ [, __DEGREES] TO <PIN#>

<CONNECT OBJECT> ::= <PIN#> | __V | A SIGNAL SET TO __V

<SYNCHRO OBJECT> ::= S1 ; S2 ; S3

<DISCONNECT STATEMENT> ::= 
    DISCONNECT <PIN#> FROM <DISCONNECT OBJECT> ;
    DISCONNECT SIGNAL FROM <PIN#> ;
    DISCONNECT __V __HZ [ __DEGREES] BETWEEN <PIN#> AND <PIN#>

<DISCONNECT OBJECT> ::= __V | <PIN#> | SYNCHRO TRANSMITTER

<OPEN CIRCUIT STATEMENT> ::= OPEN CIRCUIT <PIN#>

<REMOVE STATEMENT> ::= 
    REMOVE <REMOVE SUBJECT I> BETWEEN <PIN#> AND <PIN#> ;
    REMOVE <REMOVE SUBJECT II> FROM <PIN#> ;
    REMOVE <PIN#> FROM <VOLTAGE>

<REMOVE SUBJECT I> ::= SIGNAL | CONNECTION | __V __HZ [ __DEGREES]

<REMOVE SUBJECT II> ::= <VOLTAGE> | <PIN#>

<VOLTAGE> ::= SIGNAL | __V
<VOLTAGE STATEMENT> ::= 

[THE] VOLTAGE AT THE FOLLOWING POINTS SHALL BE: 
  {<PIN#> MEASURE _V → _V} ; 

[THE] VOLTAGE AT <PIN#> SHALL BE: 
  MEASURE _V → _V 
  [WITHIN A PERIOD OF: CHECK _S MAX ! 
  FOR A PERIOD OF:    CHECK _S MIN] ; 

[THE] VOLTAGE BETWEEN <PIN#> AND <PIN#> SHALL BE: 
  MEASURE _V ← _V ; 

<OPERATOR INTERACTION STATEMENT> ::= <PRINT STATEMENT> | READ-YES | 
    READ-NO | READ-NUM _V ← _V 

<PRINT STATEMENT> ::= PRINT { { R<NUMBER>} {"CHARACTER STRING"} 
  {R<NUMBER>} } ; 

<SAVE STATEMENT> ::= SAVE [IN] R<NUMBER> 

<VOLTAGE CHANGE STATEMENT> ::= 

VOLTAGE CHANGE AT <PIN#> FROM R<NUMBER> SHALL BE: MEASURE _V+→ _V ; 

<WAIT STATEMENT>::= WAIT ___ SECONDS 

<START P-CODE STATEMENT> ::= $START-P 

<STOP P-CODE STATEMENT> ::= $STOP-P 

-31-
<MONITOR STATEMENT> ::= 
MONITOR [A] TRANSITION AT <PIN#> FROM __V + __V TO __V + __V AT TIME __ + __ ;
MONITOR [THE] VOLTAGE AT <PIN#> OF __V + __V FOR __ S
4.0 HARDWARE DEFINITION FILE/TABLES

The hardware definition file specifies the hardware configuration of the
test set to the compiler. Included are the pins identifiers, their A/D
addresses, and the possible connections that can be made to each pin. Also
specified is the A/D scale factor for a connection to a pin. The A/D scale
factor specifies the hardware conditioning of a signal between the pin and the
A/D.

Marconi should supply the hardware definition source file in the format
described below. This ASCII source file will be converted into a form
suitable for compilation. The compiler compatible form was designed to allow
the efficient retrieval of information required at compile time. Three tables
link the appropriate data as described later in this document.

HARDWARE DEFINITION FILE - SOURCE FORM

Each record of the file has the following format described below. Note
unused fields should be blank filled.

1) Column 1-10 Pin identifier. -left justified- This consists of up to ten
   alphanumeric characters and hyphens. No embedded blanks are
   allowed in the ID.

2) Column 16-19 A/D address. -right justified- This specifies the A/D
   address of the pin. Range: 0000-4095.

3) Column 24-33 Type of connection -left justified- This specifies valid pin
   connections. The valid types are:
      1) 10 Character pin ID
      2) DC DACS: 'DCO' to 'DC 255'
      3) AC DACS: 'ACO' to 'AC 255'
      4) D/S Pins: 'DS01', 'DS02', 'DS03'
5) Logic Discretes: 'LD+28', 'LD+16', 'LD-16',
   'LD+13', 'LD-13', 'LD+10', 'LDO', 'OPENC'
6) Operator Controlled: 'MANUAL' (i.e. to
distinguish pins not under computer control)
7) Continuation: '*' - to specify another switch
   condition for a previous connection.

4) Column 40-44 Switch identifier - left justified- Range: 'S0' to 'S4095'
5) Column 48-53 Switch state - 'OPEN' or 'CLOSED'
6) Column 56-57 A/D conditioning factor - Range ' ' or '00' to '99'

Figure 1 illustrates the source form of the hardware definition field.
J101A-1 4095 OPENC  S34  OPEN  3
  *  S29  CLOSED
  *  S33  CLOSED
LD+28 S34  CLOSED  1
  *  S29  OPEN
  *  S33  CLOSED
J101A-2  S33  OPEN

J101A-3 000001 MANUAL
J101A-4  34  LDO  S10  OPEN  0
LD=28 S10  CLOSED  2

Figure 1 HARDWARE DEFINITION FILE - SOURCE FORM
4.1 **H/W DEFINITION TABLES - CONVERTED FORM**

The converted H/W definition for the ATP compiler consists of three tables interconnected by pointers. The three tables are the Pin-A/D table, the Connection table, and the Pin ID table.

Each entry of the Pin-A/D table consists of a 10 character pin ID, its associated A/D address, and a pointer into the connection table. Its purpose is to cross reference pin IDs with their A/D addresses and possible connections. It has the following format:

1) 10 character pin ID - five 16 bit words
2) A/D address - 16 bit word
3) Number of entries in connection table - 16 bit word
4) Connection table pointer - 16 bit word

The connection table pointer gives the location in the connection table that is associated with the specified pin ID.

The Connection table specifies the connections that can be made to a particular pin. The type of connection is given followed by a variable number of switch specifications and their states. For each connection, an A/D scale factor can also be given. This scale factor specifies the type of hardware signal conditioning that occurs between the specified pin and the A/D. Each entry has the following format:
1) Type of connection - 16 bit word with the following format

| BITS 15 14 13 12 11 10 9 8 | 
|---------------------------|---
| LOGIC DISCRETE-            | 
| DC DAC -                   | 
| AC DAC -                   | 
| DIGITAL/SYNCHRO -          | 
| MANUAL                     | 
| ANOTHER PIN -              | BITS 7-0
|                           | VALUE OF DISCRETE
|                           | (OPEN = 99)
|                           | DAC NUMBER
|                           | DAC NUMBER
|                           | D/S PIN NUMBER (1,2,3)
|                           | INDEX INTO PIN ID TABLE

Also, bits 15-8 = 0 for a continuation.

2) Switch ID and State - 16 bit word

<table>
<thead>
<tr>
<th>BIT 15</th>
<th>1 = CLOSED, 0 = OPEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>BITS 14-0</td>
<td>Switch Number</td>
</tr>
</tbody>
</table>

3) A/D scale factor - 16 bit word

The pin ID table is used to contain the 10 character IDs of pins which are connected via switches to other pins. When an entry in the connection table specifies a connection to a pin, it contains a pointer to the 10 character ID in the pin ID table. The 10 character IDs are stored in ASCII in five 16 bit words.

Figure 2 illustrates the converted form of the hardware definition table.
<table>
<thead>
<tr>
<th>PIN ID</th>
<th>A/D ADDR</th>
<th># OF ENTRIES</th>
<th>POINTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>101A-1</td>
<td>123</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

**Switch ID A/D Scale & State Factor**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>SWITCH ID A/D SCALE &amp; STATE</th>
<th>FACTOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD + 28</td>
<td>OPEN-S28</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>CLOS-S27</td>
<td></td>
</tr>
<tr>
<td>OPENC</td>
<td>OPEN-S28</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>OPEN-S27</td>
<td></td>
</tr>
<tr>
<td>PIN</td>
<td>CLOS-S28</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 2 Hardware Definition Tables - Converted Form**

**PIN-A/D Table**

**Connection Table**

**PIN ID Table**

- J101A-2
- J101A-3
5.0 ESTIMATE OF PERCENTAGE OF COMPILABLE ATP STATEMENTS

For estimating the percentage of compilable ATP statements, two sections of the ATP were selected. Section 9 was selected so as to include some waveform measurements. Section 19 was selected as it was included in the original ATP software specification as a "typical" section.

Both sections of the ATP were modified according to the syntax rules given in Section 3. It was also necessary to use mnemonic p-code to encode a few clauses of the ATP. The original and modified forms of both sections are given in the Appendix.

The results of the analysis of the sections is given in Table 5.1. It is evident from the high levels of percentage compilation (97% and 100%) that the Marconi ATP Language is more than adequate to express the Boeing 747 autothrottle tests. Sections 9 and 19 are typical in the sense that they employ the kinds of statements found in the rest of the test procedure. Therefore, there is good evidence that the high percentage of compilation will be realized in the remaining sections.
<table>
<thead>
<tr>
<th></th>
<th>SECTION 9</th>
<th>SECTION 19</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL NUMBER OF ORIGINAL ATP STATEMENTS:</td>
<td>89</td>
<td>110</td>
</tr>
<tr>
<td>TOTAL NUMBER OF MODIFIED ATP STATEMENTS:</td>
<td>103</td>
<td>119</td>
</tr>
<tr>
<td>NUMBER OF HIGH LEVEL STATEMENTS AFTER MODIFICATION:</td>
<td>100</td>
<td>119</td>
</tr>
<tr>
<td>NUMBER OF P-CODE STATEMENTS AFTER MODIFICATION:</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>PERCENTAGE OF HIGH LEVEL STATEMENTS AFTER MODIFICATION:</td>
<td>97%</td>
<td>100%</td>
</tr>
</tbody>
</table>

TABLE 5.1 ANALYSIS OF SECTIONS 9 AND 19
APPENDIX

ORIGINAL AND MODIFIED FORMS
OF SECTION 9 AND SECTION 19
<table>
<thead>
<tr>
<th>Test No.</th>
<th>Test Procedure</th>
<th>Value</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9</td>
<td>MAIN INTERLOCKS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.1</td>
<td>Connect J101B-6 (ATT VALID) to +28.000V</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td></td>
<td>Connect J101B-8 (CADC/TAS VALID) to +28.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 (AT VALID) shall be:</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td></td>
<td>If the test set type 94-052-01 is being used place the DISC switch in the O/C position</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.2</td>
<td>Remove J101B-55 (AT DISENABLE) from +28V</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.2.1</td>
<td>Connect J101B-55 to +28.000V (Over Ride)</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>5.9.2.2</td>
<td>(Move biased switch of Test Circuit 7 to +28V)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove 115V, 400Hz from J101A-7(L)</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td></td>
<td>J101A-15(L)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J101B-122 shall be:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.2.3</td>
<td>Connect 115V, 400Hz to J101A-7(L)</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td></td>
<td>J101A-15(L)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Release biased switch of Test Circuit 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.3</td>
<td>Connect J101B-6 to 0V</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>5.9.4</td>
<td>Connect J101B-6 to +28.000V</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>5.9.5</td>
<td>Connect J101B-8 to 0V</td>
<td>Measure</td>
<td>+11.500V</td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.6</td>
<td>Connect J101B-8 to +28.000V</td>
<td>Measure</td>
<td>+28.000V</td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.7</td>
<td>Connect J101B-47 to +28.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-65 to +28.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove J101B-55 from 28V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-29 to -16.000V then +16.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-55 to J101B-122</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-124 to J101B-55</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J102-22 (EPR COMPAR VALID) to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage J101B-122 shall be of the form shown in Figure 703:</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
V_1
\]

\[
V_2
\]

Where \( V_1 \) is +28.000V ±2.000V

\( V_2 \) is +9.500V ±1.000V

\( T \) is 0.6s ±0.4s

Figure 703

<table>
<thead>
<tr>
<th>5.9.8</th>
<th>Wait at least 1.0 second</th>
<th>Measure</th>
<th>+28.000V</th>
<th>±2.000V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J102-22 to +12.000V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open circuit J101B-124</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>5.9.9</td>
<td>Connect J101B-29 to -16.000V and then to +16.000V. Connect J101B-124 to J101B-55. Connect J101B-65 (MASTER DIM) to 0V. The voltage at J101B-122 shall be of the form shown in Figure 704.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V₁ is +28.000V ±2.000V. V₂ is +9.500V ±1.000V. T is 0.600s ±0.4s. Figure 704.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Wait at least 1.0 second. The voltage J101B-122 shall be: Measure +28.000V ±2.000V. Connect J101B-65 to +28.000V. Open circuit J101B-124.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.11</td>
<td>DELETED</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 704
<table>
<thead>
<tr>
<th>Test No.</th>
<th>Test Procedure</th>
<th>Value</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9.12</td>
<td>Connect J101B-29 to -16.000V and then to +16.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove J101B-55 from J101B-122</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-55 to +28V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.13</td>
<td>Connect J101B-124 to J101B-55</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-47 (TAT/EPR L VALID) to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>at time t = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be of the form shown in Figure 706:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Figure 706" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Where V&lt;sub&gt;1&lt;/sub&gt; is +28.000V ±2.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>V&lt;sub&gt;2&lt;/sub&gt; is +11.500V ±1.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>T&lt;sub&gt;2&lt;/sub&gt; is 0.750s ±0.250s</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t&lt;sub&gt;1&lt;/sub&gt;-t&lt;sub&gt;2&lt;/sub&gt; is 0.6s ±0.4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.14</td>
<td>Wait at least 1.0 second</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td></td>
<td>Connect J101B-47 to +28V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.15</td>
<td>Open circuit J101B-120 (GA INIT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.16</td>
<td>Connect J101B-120 to +28.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>5.9.17</td>
<td>Connect J101B-21 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-22 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-23 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.18</td>
<td>Connect J101B-23 to +10.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>5.9.19</td>
<td>Connect J101B-24 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.20</td>
<td>Connect J101B-22 to +10.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-23 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.21</td>
<td>Connect J101B-21 to +10.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-22 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.22</td>
<td>Connect J101B-22 to +10.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-23 to +10.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-24 to +10.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>5.9.22.1</td>
<td>Open circuit J101B-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td>5.9.22.2</td>
<td>Connect J101B-11 to +28.000V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+28.000V ±2.000V</td>
</tr>
<tr>
<td>5.9.22.3</td>
<td>Open circuit J101A-11</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-122 shall be:</td>
<td>Measure</td>
<td>+11.500V ±1.000V</td>
</tr>
<tr>
<td></td>
<td>Connect J101A-11 to +28.090V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
</tr>
<tr>
<td>5.9.23</td>
<td>Connect J101B-47 to 0V</td>
<td></td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>Connect J101B-65 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-6 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-8 to 0V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open circuit J102-22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Open circuit J101B-124</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.9.24</td>
<td>The resistance between J101B-55 and J101B-12 shall be: Measure 0.5 ohm Max.</td>
<td></td>
<td>D</td>
</tr>
</tbody>
</table>
SECTION 9
MODIFIED FORM
5.9 MAIN INTERLOCKS

5.9.1 CONNECT J101B-6 (ATT VALID) TO +28.000V
CONNECT J101B-8 (CADC/TAS VALID) TO +28.000V
THE VOLTAGE AT J101-B122 (AT VALID) SHALL BE:
MEASURE +28.000V +-2.000V

5.9.2 REMOVE J101B-55 (AT DISENGA) FROM +28V
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.000V

5.9.2.1 CONNECT J101B-55 TO +28.000V (OVER RIDE)
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V

5.9.2.2 PRINT "MOVE BIASED SWITCH OF TEST CIRCUIT 7 TO +28V";
PRINT "READY [Y/N]?";
READ-YES
PRINT "REMOVE 115V, 400HZ FROM J101A-7(L) AND J101A-15(L)";
PRINT "READY [Y/N]?";
READ-YES
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.00V

5.9.2.3 PRINT "CONNECT 115V, 400HZ TO J101A-7(L) AND J101A-15(L)";
PRINT "READY [Y/N]?";
READ-YES
PRINT "RELEASE BIASED SWITCH OF TEST CIRCUIT";
PRINT "READY [Y/N]?";
READ-YES
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V

5.9.3 CONNECT J101B-6 TO OV
5.9.4 CONNECT J101B-6 TO +28.000V
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V ± 2.000V
END

5.9.5 CONNECT J101B-8 TO 0V
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V ± 1.000V
END

5.9.6 CONNECT J101B-8 TO +28.000V
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V ± 2.000V
END

5.9.7 CONNECT J101B-47 TO +28.000V
CONNECT J101B-65 TO +28.000V
REMOVE J101B-55 FROM 28V
CONNECT J101B-29 TO -16.000V
CONNECT J101B-29 TO +16.000V
CONNECT J101B-55 TO J101B-122
CONNECT J101B-124 TO J101B-55
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V ± 2.000V
CONNECT J102-22 TO 0V
MONITOR A TRANSITION AT J101B-122 FROM 28.000V ± 2.000V TO 9.500V ± 1.000V AT TIME 0.0S → 0.200S
MONITOR A TRANSITION AT J101B-122 FROM 9.500V → 1.000V TO 28.000V ± 2.0V AT TIME 0.600S → 0.400S
END

5.9.8 MONITOR THE VOLTAGE AT J101B-122 OF +28.000V ± 2.000V FOR 1.000S
END

5.9.9 CONNECT J101B-29 TO -16.000V
CONNECT J101B-29 TO +16.000V
CONNECT J101B-124 TO J101B-55
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V ± 2.000V
CONNECT J101B-65 (MASTER DIM) TO 0V
MONITOR A TRANSITION AT J101B-122 FROM 28.000V ± 2.000V TO 9.500V
+-1.000V AT TIME 0.000S +-2.000S
MONITOR A TRANSITION AT J101B-122 FROM 9.5 +-1.000V TO +28.000V
+-2.000V AT TIME 0.600S +-0.400S

BEGIN
5.9.10 MONITOR THE VOLTAGE AT J101B-122 OF +28.000V +-2.000V FOR 1.000S
END

BEGIN
5.9.11 DELETED (A)
END

BEGIN
5.9.12 CONNECT J101B-29 TO -16.000V
CONNECT J101B-29 TO +16.000V
REMOVE J101B-55 FROM J101B-122
CONNECT J101B-55 TO +28V
END

BEGIN
5.9.13 CONNECT J101B-124 TO J101B-55
CONNECT J101B-47 (TAT/EPR L VALID) TO OV
MONITOR A TRANSITION AT J101B-122 FROM 28.000V +-2.000V TO 11.500V
+-1.000V AT TIME 0.000S +-2.000S
MONITOR A TRANSITION AT J101B-122 FROM 11.500V +-1.000V TO +28.000V
+-2.000V AT TIME 0.600S +-0.400S
END

BEGIN
5.9.14 MONITOR THE VOLTAGE AT J101B-122 OF 28.000V +-2.000V FOR 1.000S
CONNECT J101B-47 TO +28V
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V
END

BEGIN
5.9.15 OPEN CIRCUIT J101B-120 (GA INIT)
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V
END

BEGIN
5.9.16 CONNECT J101B-120 TO +28.000V
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V
END
5.9.17 CONNECT J101B-21 TO OV  
CONNECT J101B-22 TO OV  
CONNECT J101B-23 TO OV  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.000V  
END

5.9.18 CONNECT J101B-23 TO +10.000V  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V  
END

5.9.19 CONNECT J101B-24 TO OV  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.000V  
END

5.9.20 CONNECT J101B-22 TO +10.000V  
CONNECT J101B-23 TO OV  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.000V  
END

5.9.21 CONNECT J101B-21 TO +10.000V  
CONNECT J101B-22 TO OV  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.000V  
END

5.9.22 CONNECT J101B-22 TO +10.000V  
CONNECT J101B-23 TO +10.000V  
CONNECT J101B-24 TO +10.000V  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V  
END

5.9.22.1 OPEN CIRCUIT J101B-11  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500V +-1.000V (B)  
END

5.9.22.2 CONNECT J101B-11 TO +28.000V  
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +28.000V +-2.000V (B)  
END
5.9.22.3 OPEN CIRCUIT J101A-11
THE VOLTAGE AT J101B-122 SHALL BE: MEASURE +11.500 ±1.000V (D)
CONNECT J101A-11 TO +28.000V (D)
END

BEGIN
5.9.23 CONNECT J101B-47 TO OV (D)
CONNECT J101B-65 TO OV (D)
CONNECT J101B-6 TO OV (D)
CONNECT J101B-8 TO OV (D)
OPEN CIRCUIT J102-22 (D)
OPEN CIRCUIT J101B-124 (D)
END

BEGIN
5.9.24 PRINT "MEASURE THE RESISTANCE BETWEEN J101B-55 AND J101B-12."; (D)
PRINT "INPUT THIS NUMBER:"; (D)
$START-P (D)
GETDEC (D)
CMP 0.500 (D)
BGT SP12 ; EXIT IF GREATER THAN 0.5 (D)
$STOP-P (D)
END

END
SECTION 19

ORIGINAL FORM
<table>
<thead>
<tr>
<th>Test No.</th>
<th>Test Procedure</th>
<th>Value</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.19</td>
<td>AIRSPEED SELECT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 5.19.1  | Connect J101B-6 to +28.000V  
           J101B-8 to +28.000V |       |           |
|         | Voltage between J101A-79 (VCMG) and J101A-78 (IAS SEL) shall be: | Measure | 0V | ±0.005V |
| 5.19.2  | Disconnect J101A-78 (IAS SEL) from 0V  
           Connect J101A-78 to a signal set to +10.000V |       | +10.000V | ±0.010V |
|         | Voltage at J101A-79 shall be: | Measure |       |           |
|         | Adjust signal at J101A-78 (IAS SEL) to -10.000V  
           Voltage at J101A-79 shall be: | Measure | -10.000V | ±0.010V |
| 5.19.3  | Connect J101B-47 (TAT EPRL COMP VALID) to +28.000V  
           Connect J101B-65 (MASTER DIM) to +28.000V  
           Connect J101B-115 (ALT VCMD VALID) to +28.000V  
           Connect J101B-116 (ALT VCMD SEL) to +28.000V  
           Connect J101B-31 (ALT MODE SEL) to -16.000V  
           and then +16.000V |       | 0V | ±0.005V |
| 5.19.4  | Disconnect J101A-80 (ALT VCMD) from 0V  
           Connect J101A-80 to a signal set to -10.000V |       | -10.000V | ±0.025V |
| 5.19.5  | Adjust signal at J101A-80 to +10.000V  
           Voltage at J101A-79 shall be: | Measure | +10.000V | ±0.025V |
| 5.19.6  | Remove signal from J101A-80  
           Connect J101A-80 to 0V  
           Connect J101B-115 to 0V  
           Connect J101B-116 to 0V  
           Remove signal from J101A-78  
           Connect J101A-78 to 0V |       |           |
<table>
<thead>
<tr>
<th>Test No.</th>
<th>Test Procedure</th>
<th>Value</th>
<th>Tolerance</th>
<th>REV</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.19.7</td>
<td>Adjust the synchro transmitter (CX) until the voltage at J103-9 (CAS) is $-3.000\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J102-3 (FLAP POSN) to a signal set to $-6.750\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 (FVC) shall be:</td>
<td>Measure</td>
<td>$0\text{V}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J103-9 (CAS) to a signal set to $-3.000\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 (FVC) shall be:</td>
<td>Measure</td>
<td>$0\text{V}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$0\text{V}$</td>
<td>±0.040V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.19.8</td>
<td>Connect J101A-66 (ALT AV) to a signal set to $-10.000\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove link from between J101A-50 and J101A-51 (AV)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect a 5.500V, 400Hz, 0° signal between J101A-51 (C) and J101A-50 (H)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage change at J103-12 shall be:</td>
<td>Measure</td>
<td>$+8.996\text{V}$</td>
<td>±0.340V</td>
</tr>
<tr>
<td></td>
<td>Voltage at J101B-66 (IAS AIDS O/P) shall be</td>
<td>Measure</td>
<td>$+5.020\text{V}$</td>
<td>±0.250V</td>
</tr>
<tr>
<td>5.19.9</td>
<td>Connect J101B-110 to $+28.000\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-109 to $+28.000\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 shall be:</td>
<td>Measure</td>
<td>$-3.214\text{V}$</td>
<td>±0.120V</td>
</tr>
<tr>
<td></td>
<td>Adjust signal at J101A-66 to $+10.000\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 shall be:</td>
<td>Measure</td>
<td>$+3.214\text{V}$</td>
<td>±0.120V</td>
</tr>
<tr>
<td></td>
<td>Connect J101B-110 to $0\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-109 to $0\text{V}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove signal from between J101A-51 and J101A-50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 shall be:</td>
<td>Measure</td>
<td>$0\text{V}$</td>
<td>±0.040V</td>
</tr>
<tr>
<td>5.19.10</td>
<td>Connect a 2.000V, 400Hz, 180° signal between J101A-51 (C) and J101A-50 (H)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage change at J103-12 shall be:</td>
<td>Measure</td>
<td>$-3.270\text{V}$</td>
<td>±0.180V</td>
</tr>
<tr>
<td></td>
<td>Voltage at J101B-66 shall be:</td>
<td>Measure</td>
<td>$+1.700\text{V}$</td>
<td>±0.140V</td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>5.19.11</td>
<td>Adjust the synchro transmitter (CX) until the voltage at J103-9 is -1.000V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 shall be: Measure</td>
<td>-5.794V</td>
<td>±0.500V</td>
<td></td>
</tr>
<tr>
<td>5.19.12</td>
<td>Adjust the synchro transmitter (CX) until the voltage at J103-9 is +0.450V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Voltage at J103-12 shall be: Measure</td>
<td>-8.160V</td>
<td>±0.420V</td>
<td></td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>5.19.13</td>
<td>Adjust the synchro transmitter (CX) until the voltage at J103-9 is (-3.000) V. Connect a signal to J101B-27 set to (-4.000) V. The voltage at J103-12 shall be:</td>
<td>Measure (-3.270) V</td>
<td>(\pm 0.075) V</td>
<td></td>
</tr>
<tr>
<td>5.19.14</td>
<td>Adjust signal at J101B-27 to (+4.000) V. The voltage at J103-12 shall be:</td>
<td>(-3.920) V</td>
<td>(\pm 0.075) V</td>
<td></td>
</tr>
<tr>
<td>5.19.15</td>
<td>Adjust signal between J101A-51 (C) and J101B-50 (H) to (2.000) V, (400) Hz, (0^\circ). Adjust signal at J101B-27 to (-2.500) V. The voltage at J103-12 shall be:</td>
<td>Measure (+2.450) V</td>
<td>(\pm 0.050) V</td>
<td></td>
</tr>
<tr>
<td>5.19.16</td>
<td>Adjust signal between J101A-51 (C) and J101A-50 (H) to (4.000) V, (400) Hz, (180^\circ). Adjust signal at J101B-27 to (+4.000) V. Connect J103-24 to (-13.000) V. The voltage at J103-12 shall be:</td>
<td>(-3.920) V</td>
<td>(\pm 0.075) V</td>
<td></td>
</tr>
<tr>
<td>5.19.17</td>
<td>Connect J101B-124 to (+28.000) V. Adjust signal at J101B-27 to (-4.000) V. The voltage at J103-12 shall be:</td>
<td>(0) V</td>
<td>(\pm 0.050) V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-92 shall be:</td>
<td>(+28.000) V</td>
<td>(\pm 2.000) V</td>
<td></td>
</tr>
<tr>
<td>5.19.18</td>
<td>Adjust signal at J101B-27 to (+4.000) V. The voltage at J103-12 shall be:</td>
<td>(-3.920) V</td>
<td>(\pm 0.075) V</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-92 shall be:</td>
<td>(0) V</td>
<td>(\pm 0.500) V</td>
<td></td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
<td>REV</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td>-----</td>
</tr>
<tr>
<td>5.19.19</td>
<td>Open circuit J101B-124</td>
<td>0V</td>
<td>±0.050V</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Open circuit J103-24</td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Connect J101B-30 to -16.000V and then +16.000V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connect J101B-124 to +28.000V</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J103-12 shall be:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101B-92 shall be:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.19.20</td>
<td>Connect J101B-29 to -16.000V and then +16.000V</td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>The voltage at J103-12 shall be:</td>
<td>-6.560V</td>
<td>±0.320V</td>
<td>A</td>
</tr>
<tr>
<td>5.19.21</td>
<td>Adjust J101B-27 to -4.000V</td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>Adjust signal between J101A-51 (C) and J101A-50 (H) to 2.000V, 400Hz, 0º</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The voltage at J101A-85 shall be:</td>
<td>-3.250V</td>
<td>±0.130V</td>
<td>B</td>
</tr>
<tr>
<td>Test No.</td>
<td>Test Procedure</td>
<td>Value</td>
<td>Tolerance</td>
<td>REV</td>
</tr>
<tr>
<td>---------</td>
<td>----------------</td>
<td>-------</td>
<td>-----------</td>
<td>-----</td>
</tr>
<tr>
<td>5.19.22</td>
<td>Remove signal from J101A-66 Connect J101B-27 to 0V Connect J101A-66 to 0V Remove signal from between J101A-50 and J101A-51 Connect J101A-50 to J101A-51</td>
<td>Voltage at J101A-85 shall be:</td>
<td>Measure</td>
<td>0V</td>
</tr>
<tr>
<td>5.19.23</td>
<td>Adjust signal at J102-3 to +5.000V Voltage at J102-11 (FLAPS &gt;22.5) shall be:</td>
<td>Measure</td>
<td>+14.500V</td>
<td>±1.500V</td>
</tr>
<tr>
<td>5.19.24</td>
<td>Remove signal from J102-3 Connect J103-9 to signal set to -9.000V Wait at least 3 seconds</td>
<td>Connect J103-9 to -6.000V as a step signal at time t = 0 seconds</td>
<td>Voltage at J102-15 shall change from</td>
<td>Measure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>to</td>
<td>Measure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>at time t =</td>
<td></td>
</tr>
<tr>
<td>5.19.25</td>
<td>Adjust signal at J103-9 to 0V Wait at least 3 seconds Connect J103-9 to -3.000V as a step signal at time t = 0 seconds</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Test Title: Autothrottle Computer

#### Test Procedure

<table>
<thead>
<tr>
<th>Test No.</th>
<th>Test Procedure</th>
<th>Value</th>
<th>Tolerance</th>
<th>REV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.19.25</td>
<td>Voltage at J103-13 shall change from to at time ( t = )</td>
<td>Measure +11.000V ±1.500V</td>
<td>Measure +0.250V ±1.250V</td>
<td>0.27s ±0.10s</td>
</tr>
<tr>
<td>1.19.26</td>
<td>Wait at least 3 seconds Connect J103-9 to 0V as a step signal at time ( t = 0 ) seconds Voltage at J103-13 shall change from to at time ( t = )</td>
<td>Measure +0.250V ±1.250V</td>
<td>Measure +11.000V ±1.500V</td>
<td>0.71s ±0.10s</td>
</tr>
<tr>
<td>1.19.27</td>
<td>Connect J103-12 to a signal set to +6.000V Voltage at J101A-85 shall be:</td>
<td>Measure -6.000V ±0.130V</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.19.28</td>
<td>Connect J103-12 to -6.000V Voltage at J101A-85 shall be:</td>
<td>Measure +6.000V ±0.130V</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Remove signal from J103-12 J101B-27 J103-9 Open circuit J101B-124 Connect J101B-65 to 0V J101B-47 to 0V J101B-6 to 0V J101B-8 to 0V</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SECTION 19

MODIFIED FORM
5.19 AIRSPEED SELECT

5.19.1 CONNECT J101B-6 TO +28.000V (B)
CONNECT J101B-8 TO +28.000V (B)
VOLTAGE BETWEEN J101A-79 (VCMD) AND J101A-78 (IAS SEL) SHALL BE: MEASURE OV +0.005V

5.19.2 DISCONNECT J101A-78 (IAS SEL) FROM OV
CONNECT J101A-78 TO A SIGNAL SET TO +10.000V
VOLTAGE AT J101A-79 SHALL BE: MEASURE +10.000V +0.010V
ADJUST SIGNAL AT J101A-78 (IAS SEL) TO -10.000V
VOLTAGE AT J101A-79 SHALL BE: MEASURE -10.000V +0.010V

5.19.3 CONNECT J101B-47 (TAT EPRL COMP VALID) TO +28.000V (B)
CONNECT J101B-65 (MASTER DIM) TO +28.000V (B)
CONNECT J101B-115 (ALT VCMD VALID) TO +28.000V
CONNECT J101B-116 (ALT VCMD SEL) TO +28.000V
CONNECT J101B-31 (ALT MODE SEL) TO -16.000V
CONNECT J101B-31 (ALT MODE SEL) TO +16.000V
VOLTAGE AT J101A-79 SHALL BE: MEASURE OV +0.005V

5.19.4 DISCONNECT J101A-80 (ALT VCMD) FROM OV
CONNECT J101A-80 TO A SIGNAL SET TO 10.000V
VOLTAGE AT J101A-79 SHALL BE: MEASURE -10.000V +0.025V

5.19.5 ADJUST SIGNAL AT J101A-80 TO +10.000V
VOLTAGE AT J101A-79 SHALL BE: MEASURE +10.000V +0.025V

5.19.6 REMOVE SIGNAL FROM J101A-80
CONNECT J101A-80 TO OV
CONNECT J101B-115 TO OV
CONNECT J101B-116 TO OV
REMOVE SIGNAL FROM J101A-78
CONNECT J101A-78 TO OV
END
5.19.7 ADJUST THE SYNCHRO TRANSMITTER (CX) UNTIL THE VOLTAGE AT J103-9 (CAS) IS -3.000V (A)
CONNECT J102-3 (FLAP POSN) TO A SIGNAL SET TO -6.750V
VOLTAGE AT J103-12 (FVC) SHALL BE: MEASURE 0V ←0.040V
SAVE IN RO
END

5.19.8 CONNECT J101A-66 (ALT V) TO A SIGNAL SET TO -10.000V
REMOVE CONNECTION BETWEEN J101A-50 AND J101A-51 (V)
CONNECT A 5.500V, 400HZ, 0 SIGNAL BETWEEN J101A-51 (C) AND J101A-50 (H)
VOLTAGE CHANGE AT J103-12 FROM RO SHALL BE:
   MEASURE +8.996V ←340.0 V (B)
VOLTAGE AT J101B-66 (IAS AIDS O/P) SHALL BE:
   MEASURE +5.020V ←0.250V
END

5.19.9 CONNECT J101B-110 TO +28.000V (D)
CONNECT J101B-109 TO +28.000V
VOLTAGE AT J103-12 SHALL BE:
   MEASURE +3.214V ←0.120V
ADJUST SIGNAL AT J101A-66 TO +10.000V
VOLTAGE AT J103-12 SHALL BE:
   MEASURE +3.214V ←0.120V
CONNECT J101B-110 TO 0V
CONNECT J101B-109 TO 0V (D)
REMOVE SIGNAL BETWEEN J101A-51 AND J101A-50
VOLTAGE AT J103-12 SHALL BE:
   MEASURE 0V ←0.040V (A)
SAVE IN RO
END

5.19.10 CONNECT A 2.000V, 400HZ, 180 SIGNAL BETWEEN J101A-51 (C) AND J101A-50 (H)
   VOLTAGE CHANGE AT J103-12 FROM RO SHALL BE:
   MEASURE -3.270 ←0.130V (A)
VOLTAGE AT J101B-66 SHALL BE:
   MEASURE +1.700V ←0.140V
END

5.19.11 ADJUST THE SYNCHRO TRANSMITTER (CX) UNTIL THE VOLTAGE AT J103-9 IS MEASURE -1.000V (A)
VOLTAGE AT J103-12 SHALL BE: MEASURE -5.794 ± 0.500 (A)

BEGIN 5.19.12
ADJUST THE SYNCHRO TRANSMITTER (CX) UNTIL THE
VOLTAGE AT J103-9 IS MEASURE +0.450V  
VOLTAGE AT J103-12 SHALL BE: MEASURE -8.160V ± 0.420V  (D)

BEGIN 5.19.13
ADJUST THE SYNCHRO TRANSMITTER (CX) UNTIL THE
VOLTAGE AT J103-9 IS MEASURE -3.000V  
CONNECT J101B-27 TO A SIGNAL SET TO -4.000V  (A)
THE VOLTAGE AT J103-12 SHALL BE: MEASURE -3.270V ± 0.075V  (A)

BEGIN 5.19.14
ADJUST SIGNAL AT J101B-27 TO +4.000V
THE VOLTAGE AT J103-12 SHALL BE: MEASURE -3.920V ± 0.075V

BEGIN 5.19.15
ADJUST SIGNAL BETWEEN J101A-51 (C) AND
J101-50 (H) TO 2.000V, 400HZ, 0 DEGREES
ADJUST SIGNAL AT J101B-27 TO -2.500V  
THE VOLTAGE AT J103-12 SHALL BE: MEASURE +2.450V ± 0.050V  (A)

BEGIN 5.19.16
ADJUST SIGNAL BETWEEN J101A-51 (C) AND
J101A-50 (H) TO 4.000V, 400HZ, 0 DEGREES
ADJUST SIGNAL AT J101B-27 TO +4.000V
CONNECT J103-24 TO -13.000V  
THE VOLTAGE AT J103-12 SHALL BE: MEASURE -3.920V ± 0.075V  (A)

BEGIN 5.19.17
CONNECT J101B-124 TO +28.000V
ADJUST SIGNAL AT J101B-27 TO -4.000V
THE VOLTAGE AT J103-12 SHALL BE: MEASURE 0V ± 0.050V  (A)
THE VOLTAGE AT J101B-92 SHALL BE: MEASURE +28.000V ±2.000V
END

BEGIN
5.19.18 ADJUST SIGNAL AT J101B-27 TO +4.000V
(A)
THE VOLTAGE AT J103-12 SHALL BE: MEASURE -3.920V ±0.075V
THE VOLTAGE AT J101B-92 SHALL BE: MEASURE 0V ±0.500V
END

BEGIN
5.19.19 OPEN CIRCUIT J101B-124
OPEN CIRCUIT J103-24
CONNECT J101B-30 TO -16.000V
CONNECT J101B-30 TO +16.000V
CONNECT J101B-124 TO +28.000V
THE VOLTAGE AT J103-12 SHALL BE: MEASURE 0V ±0.050V
THE VOLTAGE AT J101B-92 SHALL BE: MEASURE +28.000V ±2.000V
END

BEGIN
5.19.20 CONNECT J101B-29 TO -16.000V
CONNECT J101B-29 TO +16.000V
(A)
THE VOLTAGE AT J103-12 SHALL BE: MEASURE -6.560V ±0.320V
END

BEGIN
5.19.21 ADJUST SIGNAL AT PIN J101B-27 TO -4.000V
ADJUST SIGNAL BETWEEN J101A-51 (C) AND J101A-50 (H) TO 2.00V, 400HZ, 0 DEGREES
(A)
THE VOLTAGE AT J101A-85 SHALL BE: MEASURE -3.250V ±0.130V
B)
END

BEGIN
5.19.22 REMOVE SIGNAL FROM J101A-66
CONNECT J101B-27 TO OV
CONNECT J101A-66 TO OV
REMOVE SIGNAL BETWEEN J101A-50 AND J101A-51
CONNECT J101A-50 TO J101A-51
VOLTAGE AT J101A-85 SHALL BE: MEASURE 0V ±0.040V
SAVE IN RO
END
5.19.23 ADJUST SIGNAL AT J102-3 TO +5.000V
VOLTAGE AT J102-11 (FLAPS>22.5)
SHALL BE: MEASURE +14.500V ± 1.500V
VOLTAGE CHANGE AT J101A-85 FROM RO
SHALL BE: MEASURE -0.250V ± 0.025V (B)
END

5.19.24 REMOVE SIGNAL FROM J102-3
CONNECT J103-9 TO SIGNAL SET TO -9.000V
WAIT 3 SECONDS
CONNECT J103-9 TO -6.000V
MONITOR A TRANSITION AT J102-15 FROM +11.000V ± 1.500V TO +0.250V
± 1.250V AT TIME 0.32S ± 0.05S
END

5.19.25 ADJUST SIGNAL AT J103-9 TO OV
WAIT 3 SECONDS
CONNECT J103-9 TO -3.000
MONITOR A TRANSITION AT J103-13 FROM +11.000V ± 1.500V TO +0.250V
± 1.250V AT TIME 0.27S ± 0.10S
END

5.19.26 WAIT 3 SECONDS
CONNECT J103-9 TO OV
MONITOR A TRANSITION AT J103-13 FROM 0.250V ± 1.250V TO +11.000V
± 1.500V AT TIME 0.71S ± 0.10S
END

5.19.27 CONNECT J103-12 TO A SIGNAL SET TO +6.000V
THE VOLTAGE AT J101A-85 SHALL BE: MEASURE -6.000V ± 0.130V
END

5.19.28 CONNECT J103-12 TO -6.000V
VOLTAGE AT J101A-85 SHALL BE: MEASURE +6.000V ± 0.130V
REMOVE SIGNAL FROM J103-12
REMOVE SIGNAL FROM J103-9
OPEN CIRCUIT J101B-124
CONNECT J101B-65 TO OV
CONNECT J101B-47 TO OV
CONNECT J101B-6 TO OV
CONNECT J101B-8 TO OV
END
FINAL REPORT

AUTOTHROTTLE TEST PROCEDURE COMPILER

By

J E. Doss
E.S. Acree
C.A. Batterman
S.N. Cole
J.B. Thompson

Software Research Division
Computer Science and Technology Laboratory
Engineering Experiment Station
Georgia Institute of Technology
August 20, 1979

Prepared for

Marconi Avionics, Inc.
4500 N. Shallowford Road
Atlanta, Georgia 30338
Under Contract 40125
<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>i</td>
</tr>
<tr>
<td>Summary</td>
<td>ii</td>
</tr>
<tr>
<td>1.0 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.01 A Sample Program</td>
<td>2</td>
</tr>
<tr>
<td>1.1 Test Procedures in the ATP Language</td>
<td>4</td>
</tr>
<tr>
<td>2.0 The ATP Language</td>
<td>7</td>
</tr>
<tr>
<td>2.1 Symbols in the ATP Language</td>
<td>7</td>
</tr>
<tr>
<td>2.1.1 Key Words</td>
<td>8</td>
</tr>
<tr>
<td>2.1.1.1 Aliases</td>
<td>9</td>
</tr>
<tr>
<td>2.1.2 Pin Numbers</td>
<td>9</td>
</tr>
<tr>
<td>2.1.3 Values and Tolerances</td>
<td>10</td>
</tr>
<tr>
<td>2.1.4 Character Strings</td>
<td>11</td>
</tr>
<tr>
<td>2.1.5 Comments</td>
<td>12</td>
</tr>
<tr>
<td>2.2 ATP Statements</td>
<td>15</td>
</tr>
<tr>
<td>2.2.1 Preliminary Statements</td>
<td>15</td>
</tr>
<tr>
<td>2.2.2 Adjust Statements</td>
<td>16</td>
</tr>
<tr>
<td>2.2.3 Set Statements</td>
<td>18</td>
</tr>
<tr>
<td>2.2.4 Connect Statements</td>
<td>18</td>
</tr>
<tr>
<td>2.2.5 Disconnect and Remove Statements</td>
<td>20</td>
</tr>
<tr>
<td>2.2.6 Voltage Measure Statements</td>
<td>21</td>
</tr>
<tr>
<td>2.2.7 READ-YES and READ-NO Statements</td>
<td>22</td>
</tr>
<tr>
<td>2.2.8 READ-NUM Statement</td>
<td>23</td>
</tr>
<tr>
<td>2.2.9 Print Statement</td>
<td>23</td>
</tr>
<tr>
<td>2.2.10 Save Statement</td>
<td>24</td>
</tr>
<tr>
<td>2.2.11 Wait Statement</td>
<td>25</td>
</tr>
<tr>
<td>SECTION</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>2.2.12 Inline P-Code</td>
<td>25</td>
</tr>
<tr>
<td>2.2.13 Transition Statement</td>
<td>26</td>
</tr>
<tr>
<td>2.2.14 Monitor Statement</td>
<td>27</td>
</tr>
<tr>
<td>3.0 Definition of P-Code</td>
<td>28</td>
</tr>
<tr>
<td>3.1 P-machine Implementation</td>
<td>29</td>
</tr>
<tr>
<td>3.2 Number Representation</td>
<td>32</td>
</tr>
<tr>
<td>3.3 P-Code Formats</td>
<td>32</td>
</tr>
<tr>
<td>3.4 Branch Instructions</td>
<td>33</td>
</tr>
<tr>
<td>3.5 Standard Procedures</td>
<td>34</td>
</tr>
<tr>
<td>3.6 P-Codes</td>
<td>36</td>
</tr>
<tr>
<td>3.6.1 NOP</td>
<td>38</td>
</tr>
<tr>
<td>3.6.2 OUTSTR</td>
<td>38</td>
</tr>
<tr>
<td>3.6.3 GETSTR</td>
<td>39</td>
</tr>
<tr>
<td>3.6.4 CMPSTR</td>
<td>39</td>
</tr>
<tr>
<td>3.6.5 OUTDEC</td>
<td>39</td>
</tr>
<tr>
<td>3.6.6 GETDEC</td>
<td>39</td>
</tr>
<tr>
<td>3.6.7 SETCLK</td>
<td>40</td>
</tr>
<tr>
<td>3.6.8 GETCLK</td>
<td>40</td>
</tr>
<tr>
<td>3.6.9 BRA</td>
<td>40</td>
</tr>
<tr>
<td>3.6.10 BLT</td>
<td>40</td>
</tr>
<tr>
<td>3.6.11 BGT</td>
<td>40</td>
</tr>
<tr>
<td>3.6.12 BLE</td>
<td>41</td>
</tr>
<tr>
<td>3.6.13 BGE</td>
<td>41</td>
</tr>
<tr>
<td>3.6.14 BEQ</td>
<td>41</td>
</tr>
<tr>
<td>SECTION</td>
<td>PAGE</td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
</tr>
<tr>
<td>3.6.15 BNE</td>
<td>41</td>
</tr>
<tr>
<td>3.6.16 BM</td>
<td>41</td>
</tr>
<tr>
<td>3.6.17 BP</td>
<td>41</td>
</tr>
<tr>
<td>3.6.18 BZ</td>
<td>42</td>
</tr>
<tr>
<td>3.6.19 BNZ</td>
<td>42</td>
</tr>
<tr>
<td>3.6.20 ABS</td>
<td>42</td>
</tr>
<tr>
<td>3.6.21 NEG</td>
<td>42</td>
</tr>
<tr>
<td>3.6.22 CLA</td>
<td>42</td>
</tr>
<tr>
<td>3.6.23 STOP</td>
<td>42</td>
</tr>
<tr>
<td>3.6.24 CMP RX</td>
<td>43</td>
</tr>
<tr>
<td>3.6.25 CMP #</td>
<td>43</td>
</tr>
<tr>
<td>3.6.26 STA Rx</td>
<td>43</td>
</tr>
<tr>
<td>3.6.27 LDA Rx</td>
<td>43</td>
</tr>
<tr>
<td>3.6.28 LDA #</td>
<td>43</td>
</tr>
<tr>
<td>3.6.29 SUB Rx</td>
<td>43</td>
</tr>
<tr>
<td>3.6.30 SUB X</td>
<td>44</td>
</tr>
<tr>
<td>3.6.31 ADD Rx</td>
<td>44</td>
</tr>
<tr>
<td>3.6.32 ADD #</td>
<td>44</td>
</tr>
<tr>
<td>3.6.33 MULT Rx</td>
<td>44</td>
</tr>
<tr>
<td>3.6.34 MULT #</td>
<td>45</td>
</tr>
<tr>
<td>3.6.35 DIV Rx</td>
<td>45</td>
</tr>
<tr>
<td>3.6.36 DIV #</td>
<td>45</td>
</tr>
<tr>
<td>3.6.37 WAIT</td>
<td>45</td>
</tr>
<tr>
<td>3.6.38 OPEN</td>
<td>45</td>
</tr>
<tr>
<td>3.6.39 CLOSE</td>
<td>46</td>
</tr>
<tr>
<td>SECTION</td>
<td>PAGE</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
</tr>
<tr>
<td>3.6.40 INPUT</td>
<td>46</td>
</tr>
<tr>
<td>3.6.41 CONV</td>
<td>46</td>
</tr>
<tr>
<td>3.6.42 MEAS</td>
<td>47</td>
</tr>
<tr>
<td>3.6.43 SETDEV</td>
<td>47</td>
</tr>
<tr>
<td>3.6.44 CHECK</td>
<td>48</td>
</tr>
<tr>
<td>3.6.45 MONV</td>
<td>48</td>
</tr>
<tr>
<td>3.6.46 TRANS</td>
<td>49</td>
</tr>
<tr>
<td>3.6.47 INCD</td>
<td>50</td>
</tr>
<tr>
<td>3.6.48 DECD</td>
<td>50</td>
</tr>
<tr>
<td>3.6.49 CHANGE</td>
<td>50</td>
</tr>
<tr>
<td>3.6.50 ADJDEV</td>
<td>51</td>
</tr>
<tr>
<td>3.6.51 READEV</td>
<td>52</td>
</tr>
<tr>
<td>4.0 Hardware Definition File/Tables</td>
<td>53</td>
</tr>
<tr>
<td>4.1 Hardware Definition File - Source Form</td>
<td>54</td>
</tr>
<tr>
<td>4.1.1 Overall Structure</td>
<td>54</td>
</tr>
<tr>
<td>4.1.2 Formatting of Lines</td>
<td>55</td>
</tr>
<tr>
<td>4.1.2.1 Format of Identifier Line</td>
<td>56</td>
</tr>
<tr>
<td>4.1.2.2 Format of &quot;DAC_&quot; Segment Records</td>
<td>57</td>
</tr>
<tr>
<td>4.1.2.3 Format of &quot;GAIN&quot; Segment Records</td>
<td>58</td>
</tr>
<tr>
<td>4.1.2.4 Format of &quot;COND&quot; Segment Records</td>
<td>58</td>
</tr>
<tr>
<td>4.1.2.5 Format of &quot;ANGL&quot; Record</td>
<td>58</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (CONT'D)

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1.2.6 Format of &quot;VOLT&quot; Segment Records</td>
<td>59</td>
</tr>
<tr>
<td>4.1.2.7 Format of &quot;PIN_&quot; Segment Records</td>
<td>59</td>
</tr>
<tr>
<td>4.2 Hardware Definition File - Converted Form</td>
<td>62</td>
</tr>
<tr>
<td>4.2.1 Common Block File (x.D3)</td>
<td>62</td>
</tr>
<tr>
<td>4.2.2 Connection File (x.D1)</td>
<td>62</td>
</tr>
<tr>
<td>4.2.2.1 Format of Connection File</td>
<td>63</td>
</tr>
<tr>
<td>4.2.3 Pin Id File (x.D2)</td>
<td>66</td>
</tr>
<tr>
<td>5.0 Operating Procedures</td>
<td>67</td>
</tr>
<tr>
<td>5.1 Compiler Usage</td>
<td>68</td>
</tr>
<tr>
<td>5.2 Assembler Usage</td>
<td>70</td>
</tr>
<tr>
<td>5.3 Hardware Definition Conversion Process</td>
<td>71</td>
</tr>
<tr>
<td>Appendices</td>
<td></td>
</tr>
<tr>
<td>1 BNF for the ATP Language</td>
<td>73</td>
</tr>
<tr>
<td>2 Generated P-Code</td>
<td>79</td>
</tr>
<tr>
<td>3 Error Messages</td>
<td>85</td>
</tr>
<tr>
<td>4 P-Code Object File Format</td>
<td>93</td>
</tr>
</tbody>
</table>
PREFACE

This report was prepared at the Georgia Tech Engineering Experiment Station under Contract 40125. The work covered by this report was performed in the Computer Science and Technology Laboratory under the supervision of Mrs. Edith W. Martin. The progress reported herein was performed during a 4-month program to develop a compiler for automating the testing of analog flight computers.

This project has been monitored by Brian Wood and Gene Lowe of Marconi Avionics, and their helpful guidance is gratefully acknowledged. Appendix 4 was supplied by W.A. Ritch of Marconi Avionics.
SUMMARY

The objective of this study was to develop a high-level test generation language and compiler for the automated testing of Marconi's Boeing 747 Autothrottle computer. This language allows Marconi's engineers to produce quality control check-out procedures for the Autothrottle computer in an easy to understand format.

The project has been performed in two phases: the analysis phase and the implementation phase. The analysis phase consisted of defining (1) the syntax and semantics of an English-like user-oriented test generation language, (2) a low-level pseudo-code (P-code) to be produced from the test generation language and (3) the architecture of an intermediate virtual computer which would execute the P-code. The virtual computer concept simplifies system implementation, since its definition could be tailored specifically to this type of application (i.e. to automated testing).

The test generation language, developed in the analysis phase, is a regular or type 3 language which is input to the compiler in a user-oriented English style format. The implementation phase consisted of writing the compiler which would convert the test generation language into P-code. An interpreter for the P-code generated by the compiler is being developed concurrently by Marconi Avionics.

The process of automated testing of the Autothrottle computer consists of the following steps. First the electrical checkout procedures are written in the test generation language (about 5000 lines of commands). The procedures are then compiled into P-code which is down loaded to the microcomputer within the test set hardware. The microcomputer interprets its P-code instructions
and applies sequences of analog signals to the autothrottle computer. For each sequence of signals applied, the microcomputer checks for correct responses from the Autothrottle computer's outputs.

The automated check-out of the Autothrottle computers will permit a substantial decrease in the amount of time required to test each unit. Previously each Autothrottle computer had to be tested manually requiring about a man-month of a technician's time.
1.0 INTRODUCTION

The Acceptance Test Procedure (ATP) language is an English-like language developed for writing automatic test procedures. Statements are written in an easy to use, self explanatory structure similar in format to a technicians test manual. Test statements are converted into an automatic test sequence using the ATP compiler. The primary design goal of the ATP compiler was to automate the execution of test specifications which have previously been written for manual testing of Marconi autothrottle computers. The language structure of the original test procedures has generally been retained. However, this structure has been limited and constrained to standardized formats and wording. Standardization helps simplify ATP programming by avoiding an overly complex language.

The ATP compiler converts the ATP language statements into mnemonic pseudo-code (P-code). This P-code is then translated by the ATP assembler into object P-code which is, in turn, converted by an interpreter on a TI 9900 microprocessor into instructions for the Autothrottle Semi-Automatic Test Equipment. Thus, the test specification procedures generated in the ATP language will ultimately control the test performed by the test equipment.
1.0.1 A SAMPLE PROGRAM

BEGIN 1.1

BEGIN 1.1.1

CONNECT J101B-6 TO +28.000V
THE VOLTAGE AT J101-B122 SHALL BE:
MEASURE +22.000V ± 2.000V

END

BEGIN 1.1.2

PRINT "MOVE BIASED SWITCH TO +28.0V";
PRINT "INPUT VOLTAGE AT J101A-60";
READ-NUM 28.000V ± 0.100V

END

BEGIN 1.1.3

MONITOR A TRANSITION AT J101B-122
FROM 9.500V ± 1.000V to 28.000V ± 2.000V
AT TIME 0.600S ± 0.400S
OPEN CIRCUIT J101B-120 (GA INIT)

$ START-P
LDA R4
MULT R5
STA R6
$ STOP-P
PRINT "R4 * R5 =" R6

END

END
Listed above is a sample ATP test program. Notice that the program is broken into blocks which contain various numbers of statements. The outer block of the test sequence is called section 1.1. The inner blocks numbered from 1.1.1 to 1.1.3 are called subsections. Within each subsection are statements which accomplish various objectives. Each statement is written in a specific format or syntax. All of the valid ATP statements are described in Section 2.0 of this Manual.

Note that subsection 1.1.1 of the sample program contains the statement 'CONNECT J101B-6 TO +28.000V'. In order to generate an automated test procedure to complete this statement, a number of conditions must be satisfied. First, the statement must be verified to be certain it is syntactically correct. The ATP compiler verifies the syntax using the syntactic rules described formally in Appendix 1. After verifying that a statement's syntax is correct, the compiler interprets the meaning (semantics) of the statement. The ATP hardware definition tables described in section 4.0 are checked to see if the desired statement can actually be completed. For the example statement to be valid, it must be possible to connect either a 28-volt logic discrete or a DAC set to 28.0 volts to the test pin J101B-6.

After having determined that a statement can actually be completed, the ATP compiler must produce instructions for the actual test computer. In the example the states of switches must be set, and perhaps a DAC value must be adjusted.
Instructions to the test computer are written using a special P-code. The ATP P-code defines a virtual computer tailored for automatic testing. Section 3.0 describes the P-code instruction set. It is possible to use P-code directly within the automatic test sequence when the complete flexibility of the P-machine is required. Section 1.1.3 of the sample test program illustrates the use of inline P-code.

After P-code has been generated by the ATP compiler, it is assembled into a compact binary form by the ATP assembler. Section 5.2 describes the use of the ATP assembler. After assembly, P-code may finally be down-loaded into the test computer for execution.

1.1 TEST PROCEDURES IN THE ATP LANGUAGE

The ATP language is basically a free format language. The free format aspect of the language allows the user to structure the test specification procedures in a natural way, without the restrictions imposed by an inflexible fixed format.

The following guidelines involving statement format must be observed:

1) A high-level statement in the ATP language may be placed on more than one line of the text of a test specification procedure. However, there are some restrictions placed on character strings and comments which should be noted in sections 1.2.3 and 1.2.5.

2) Each line of test specification text may contain up to 132 characters.

3) Tab characters, blanks, and commas are all considered to be delimiters (separators).
4) Four of the basic components of the language (key-words, pin-numbers, numeric values, and character strings) must be separated from each other by one or more delimiters, the only exception is that numeric values and their unit tokens (VOLTS, SECONDS, etc.) do not have to be separated by delimiters.

To specify tests using the ATP language a test procedure must be written following the standard formulation. A test procedure is composed of one or more sections. A section consists of a BEGIN verb, a section number, a section name, zero or more subsections and is concluded with an END statement.

For example:

BEGIN 5.1 THIS IS A SECTION NAME

SUBSECTION 0

.

.

.

SUBSECTION N

END
A subsection consists of a BEGIN verb, a subsection number, a series of zero or more statements and is concluded by an END statement. For example:

BEGIN 5.1.1 STATEMENT 0

.

.

.

STATEMENT N

END

The following items should be noted about sections, subsections, BEGIN's, and END's:

1) The section name is limited to 40 characters in length. The character count for the name begins with the first nonblank character to the right of the section number, and the character count ends with either the last nonblank character of the name or the fortieth character of the name, whichever comes first.

2) Subsections do not typically have names attached to them. However, if it is desired to name a subsection then this may be accomplished by putting the name in parenthesis (as a comment).

3) Statements of the language are found in the subsections and not in the sections.

4) The first statement of a subsection may appear either on the same line as the begin statement appears (as in 5.1.1), or it may appear on the following line.

5) Each section and each subsection must be concluded with an END statement.
2.0 THE ATP LANGUAGE

2.1 SYMBOLS OF THE ATP LANGUAGE

The ATP language allows the user to specify such operations as CONNECT, DISCONNECT, ADJUST, and MONITOR in a high-level language. Additional operations which are not available in the high-level language may be specified with in-line P-code statements.

The high-level statements of the ATP language consist of five basic components:

1) Key words - such as ADJUST, BETWEEN, TO, VOLTAGE, and PRINT.

2) Pin-numbers - consisting of mnemonic identifiers of the pin connections on the equipment being tested such as J101A-1, J103-9, etc.

3) Numeric values - consisting of positive and negative integers and real numbers such as +16, +16.0, 0.005, .5 and so on.

4) Character strings - which are found in the print statement and consist of alphanumeric strings of eighty characters or less enclosed with single or double quote marks.

5) Comments - consisting of any descriptive information the user would like to have in the text of the ATP source file. Comments are enclosed in parenthesis. Any information enclosed in parenthesis will not be processed by the compiler and is simply "stripped" from the source file.
2.1.1. **KEY WORDS**

The key words of the ATP language consist of approximately 88 words commonly used in the ATP language high-level statements. These words must be spelled correctly or they will not be identified by the ATP compiler. Some words have aliases which allow variations on the standard form of the word used in the syntax of the ATP language. The following is a complete list of keywords:

1) $START-P 45) PERIOD
2) $STOP-P 46) POINTS
3) ; 47) POS
4) A 48) POSITIVE
5) ADJUST 49) PRINT
6) AND 50) RO
7) AT 51) R1
8) BE 52) R2
9) BE: 53) R3
10) BEGIN 54) R4
11) BETWEEN 55) R5
12) BY 56) R6
13) CHANGE 57) R7
14) CHANGES 58) R8
15) CHECK 59) R9
16) CIRCUIT 60) READ-NO
17) CLOCKWISE 61) READ-NUM
18) CONNECT 62) READ-YES
19) CONNECTION 63) REMOVE
20) COUNTERCLOCKWISE 64) S
21) D 65) S1
22) DEG 66) S2
23) DEGREE 67) S3
24) DEGREES 68) SAVE
25) DIRECTION 69) SEC
26) DISCONNECT 70) SECOND
27) END 71) SECONDS
28) FOLLOWING 72) SET
29) FOR 73) SHALL
30) FROM 74) SIGNAL
31) HZ 75) SYNCHRO
32) IN 76) THE
33) IS: 77) TIME
34) MAX 78) TO
35) MEASURE 80) TRANSITION
37) MIN 81) TRANSMITTER
38) MONITOR 82) UNTIL
39) NEG 83) V
40) NEGATIVE 84) VOLT
41) OF 85) VOLTAGE
42) OF: 86) VOLTS
43) OPEN 87) WAIT
44) PER 88) WITHIN
2.1.1.1 ALIASES

There are some key words that are treated as aliases of each other. For example, D, DEG, DEGREE, and DEGREES are all aliases of each other and any one of them may be used in place of the other. The following is a list of allowable aliases for the ATP language:

1) BE, BE:
2) CHANGE, CHANGES
3) D, DEG, DEGREE, DEGREES
4) IS, IS:
5) NEG, NEGATIVE
6) OF, OF:
7) POS, POSITIVE
8) S, SEC, SECOND, SECONDS
9) TO, TO:
10) V, VOLT, VOLTS

2.1.2 PIN NUMBERS

Pin-numbers consist of mnemonic identifiers of the different pin connections on the autothrottle equipment being tested. The mnemonic identifier consists of from one to ten characters in length. Each pin must be properly defined in the hardware definition file so that all the legal connections for a pin are specified. The following examples illustrate some typical pin numbers:

J101A-57
J103-9
J101B-55
J102-22
J101B-124
2.1.3 VALUES AND TOLERANCES

Two different types of numbers occur in the ATP language: Values and tolerances. Values may be real or integer, signed or unsigned, and may or may not be followed by a unit token such as VOLTS, DEGREES, SECONDS, or HZ. One form of the ADJUST statement allows the user to specify a step size which should not be followed by a unit token.

Tolerances, on the other hand, are immediately preceded by the symbols \(+\) (plus minus) and are always followed by a unit token. Tolerances are found only after a value with a unit token. The unit token of the tolerance must match the unit token of the value that it follows. The unit token of the tolerance is always checked to ensure that it is of the same type as the unit token of the preceding value.

Numeric values and tolerances may range between the values -32767 and +32767. If a number is found that is out of this range an appropriate error message will be printed. Up to three decimal digits to the right of the decimal point will be processed; any additional digits will be ignored and an error message printed. The following values and tolerances are examples of legal value/tolerance usage:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>TOLERANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>16.000</td>
<td>+/-0.005V</td>
</tr>
<tr>
<td>16.000V</td>
<td>+/-0.050 V</td>
</tr>
<tr>
<td>16.000 V</td>
<td>+/-1 SEC</td>
</tr>
<tr>
<td>16 SEC</td>
<td>+/-1 SEC</td>
</tr>
<tr>
<td>16 SECONDS</td>
<td>+/-1 SEC</td>
</tr>
</tbody>
</table>
The following values and tolerances are examples of illegal value/tolerance usage:

<table>
<thead>
<tr>
<th>VALUE</th>
<th>TOLERANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>+3.125</td>
<td>(NO VALUE AND NO UNITS)</td>
</tr>
<tr>
<td>16.000 S</td>
<td>+-1.0V (UNITS DO NOT MATCH)</td>
</tr>
<tr>
<td>1.0V</td>
<td>0.005V (NO +- ON TOLERANCE)</td>
</tr>
<tr>
<td>+45000 V</td>
<td>+100.0 V (VALUE OUT OF RANGE)</td>
</tr>
<tr>
<td>6,000 SEC</td>
<td>+1,000 SEC (COMMAS ARE NOT ALLOWED)</td>
</tr>
</tbody>
</table>

2.1.4 CHARACTER STRINGS

Character strings occur in the PRINT statement. Anything enclosed in single or double quote marks is considered a part of the character string and will be passed without further processing to the output buffer. Character strings are limited to eighty characters in length, including the quote marks.

Because of the maximum length limitations on the character string, character strings should not extend over one line; otherwise, blank characters may be improperly inserted into the character string. The extra blanks will be counted as part of a character string's eighty character limit and therefore may cause an error in processing. The following examples of the PRINT statement illustrate the correct usage of the character string:

PRINT "THIS CHARACTER STRING WOULD BE ENTIRELY TOO"
"LONG SO IT HAS BEEN SPLIT INTO TWO STRINGS"
PRINT 'SINGLE OR DOUBLE QUOTE MARKS MAY BE USED'
The following examples illustrate some illegal character strings:

PRINT "THIS CHARACTER STRING IS ENTIRELY TOO LONG AND SHOULD HAVE BEEN SPLIT UP TO AVOID TAKING UP TOO MUCH SPACE"
PRINT "THIS STRING HAS NO CLOSING QUOTE MARK"
PRINT "THIS STRING HAS NO OPENING QUOTE MARK"

2.1.5 COMMENTS

A comment convention has been adopted for the ATP language to allow explanatory information to be inserted into the text of a test specification procedure. Comments are literally "stripped" from the test procedure at compile time and they do not cause any code to be generated.

Comments are enclosed in parenthesis which indicate to the ATP compiler that the information enclosed in the parenthesis is not to be processed any further. This implies that parentheses cannot be included within the comment itself. The one exception to the comment convention involves character strings. Any information enclosed inside parenthesis in a character string will be output with the rest of the character string.

A comment may be up to twenty lines in length. If a comment is not concluded within twenty lines an error message will be printed. The twenty line limitation is enforced to avoid having the remainder of a test procedure treated as a comment by the ATP compiler in the event that the closing parenthesis is missing. If it is desired to have longer comments, simply close the first part of the comment when the line limit has been reached and begin on the next line with a "new" comment containing the second portion of the original comment and so on. The following examples illustrate correct
usage of the comment convention:

(THIS IS A SIMPLE ONE LINE COMMENT)

(THIS IS A MULTI-LINE COMMENT WHICH MAY EXTEND OVER AS MANY AS TWENTY LINES OF TEXT, WITH UP TO 132 CHARACTERS PER LINE).

(BLANK LINES MAY ALSO BE INSERTED INTO A COMMENT LIKE THIS ONE BUT, THE BLANK LINE WILL BE COUNTED ALONG WITH THE OTHER LINES IN THE COMMENT TOWARD THE TWENTY LINE MAXIMUM)

The following examples demonstrate some incorrect uses of the comment convention:

(THIS COMMENT HAS NO CLOSING PARENTHESIS, WHICH WILL CAUSE THE COMPILER TO SCAN UP TO TWENTY LINES OF TEXT LOOKING FOR THE CLOSING RIGHT PARENTHESIS INDICATING THE END OF THE COMMENT.

THIS COMMENT HAS NO OPENING PARENTHESIS WHICH WILL CAUSE THE COMPILER TO ATTEMPT TO TREAT THIS COMMENT AS THOUGH IT WAS A HIGH-LEVEL ATP STATEMENT. THE CLOSING RIGHT PARENTHESIS WILL BE IGNORED AND MAY CAUSE A SYNTAX ERROR.)
(COMMENT LINE 1: IF A COMMENT EXTENDS OVER MORE THAN

COMMENT LINE 2: TWENTY LINES, THEN THE COMMENT IS TOO

COMMENT LINE 3: LONG AND WILL GENERATE AN ERROR MESSAGE

COMMENT LINE 4: TO THAT EFFECT.

.

.

.

COMMENT LINE 20: THIS COMMENT SHOULD END ON THIS LINE, BUT

COMMENT LINE 21: SINCE IT DOES NOT IT WILL GENERATE AN

ERROR)
2.2. **ATP STATEMENTS**

2.2.1 **PRELIMINARY STATEMENTS**

Two statements are required as the first and second statements of an ATP source file. The first statement is of the form

```
TEST SPECIFICATION NUMBER = <string>
```

where `<string>` is a string of characters with a maximum length of 10. This statement gives the test specification number which is to be copied to the output object file.

The second statement is of the form

```
HARDWARE DEFINITION FILE = <filename>
```

where `<filename>` is the name of the hardware definition file to be used by the compiler. The filename should end with a '.DF' extension (e.g. HW747.DF would be a valid name) and have at most 14 characters.
2.2.2 **ADJUST STATEMENTS**

The adjust statements may be used to step the output of a device (associated with <PIN 1>), starting at its current value, until a measurement (associated with <PIN 2>) reaches a desired value or until the value of the output reaches a limit value. The specified output (from a DC–DAC or D/C) is incremented by the given step size once after the desired step time has elapsed. A measurement is then made from the A/D associated with <PIN 2>. If the voltage is within the desired range (\( V_{\text{min}} - V_{\text{max}} \)) the statement successfully terminates. The process of stepping the output and measuring the A/D voltage repeats until the desired value is measured or the output reaches a limit value. If the limit value occurs, an error message and appropriate status values are printed on the user's console.
The following two statements are used to adjust DC-DAC values:

ADJUST SIGNAL AT <PIN 1> BY <STEP SIZE>
PER <TIME> SECONDS IN A <SIGN>
DIRECTION UNTIL THE VOLTAGE AT
<PIN 2> CHANGES TO \( V \pm V \)

ADJUST SIGNAL AT <PIN 1> BY
<STEP SIZE> PER <TIME> SECONDS UNTIL
THE VOLTAGE AT <PIN 2> IS \( V \pm V \)

The following type of statement may be used to adjust the synchro transmitter until the voltage at a pin reaches the desired value:

ADJUST SYNCHRO TRANSMITTER BY
<STEP SIZE> PER <TIME> SECONDS
UNTIL THE VOLTAGE AT <PIN #>
IS \( V \pm V \)
2.2.3 SET STATEMENT

The set synchro statement is utilized to set the synchro transmitter to a specific angle. The set statement appears within the ATP test sequence as follows:

```
SET THE SYNCHRO TRANSMITTER TO __ DEGREES +— DEGREES

<DIRECTION>
```

<DIRECTION> can be either omitted (defaults to 'CLOCKWISE'); or 'CLOCKWISE' or 'COUNTERCLOCKWISE' can be substituted for <DIRECTION>.

2.2.4 CONNECT STATEMENTS

The connect statements are used to connect object devices to automatic test equipment pins. A connection object may be another pin, a D/S source, or a DAC set to a specified voltage.

To connect a test pin to another test pin use the statement:

```
CONNECT <PIN 1> TO <PIN 2>
```

To connect a test pin to a DC voltage use one of the following statements. Note that the ATP compiler first checks if a logic discrete can be connected. If not, a DC-DAC is used.

```
CONNECT <PIN 1> TO _V
CONNECT <PIN 1> TO A SIGNAL SET TO _V
```
To apply an AC voltage to a test pin use the following ATP statement:

CONNECT _V, _HZ <PHASE> TO <PIN 1>

Either omit <PHASE> (defaults to 0 degrees), or substitute 'DEGREES' in place of <PHASE>.

To connect a digital to synchro source to a test pin use the statement:

CONNECT <SYNCHRO PIN> TO <PIN 1>

Substitute either 'S1', 'S2', or 'S3' for <SYNCHRO PIN> in the statement.
2.2.5 DISCONNECT AND REMOVE STATEMENTS

The group of disconnect and remove statements may be utilized to remove connections between test objects and pins. All statements in this group function similarly. The first pin number in a statement is used to locate an open circuit entry in the hardware definition table. Any switches associated with the open circuit entry are placed in the specified state required to attain an open circuit status.

Following are the various forms of the disconnect and remove statements:

- DISCONNECT <PIN 1> FROM __V
- DISCONNECT <PIN 1> FROM <PIN 2>
- DISCONNECT <PIN 1> FROM SYNCHRO TRANSMITTER
- DISCONNECT SIGNAL FROM <PIN 1>
- REMOVE SIGNAL BETWEEN <PIN 1> AND <PIN 2>
- REMOVE CONNECTION BETWEEN <PIN 1> AND <PIN 2>
- REMOVE __V __HZ [__DEGREES] BETWEEN <PIN 1> AND <PIN 2>
- REMOVE SIGNAL FROM <PIN 1>
- REMOVE __V FROM <PIN 1>
- REMOVE <PIN 1> FROM <PIN 1>
- REMOVE <PIN 1> FROM SIGNAL
- REMOVE <PIN 1> FROM __V
2.2.6 **VOLTAGE MEASURE STATEMENTS**

Voltage measure statements are used to insure that a voltage measured at a test pin is within tolerance. The following two statements both check that the voltage at <PIN 1> is within the range: \( V \pm V \)

**VOLTAGE AT THE FOLLOWING POINTS SHALL BE:**

\(<\text{PIN 1}> \text{ MEASURE } _V \pm _V \)

**VOLTAGE BETWEEN <PIN 1> AND <PIN 2> SHALL BE: MEASURE \_V \pm \_V**

When a large number of voltages must be checked, the following statements may be used:

**VOLTAGE AT THE FOLLOWING POINTS SHALL BE:**

\(<\text{PIN 1}> \text{ MEASURE } _V \pm _V \)

\(<\text{PIN 2}> \text{ MEASURE } _V \pm _V \)

. . .

\(<\text{PIN N}> \text{MEASURE } _V \pm _V \)

There are two alternative forms of the voltage measure statement. When it is necessary to wait a period of time before a voltage measurement is made, the following statement can be used:

**VOLTAGE AT <PIN 1> SHALL BE:**

\( \text{MEASURE } _V \pm _V \text{ WITHIN A PERIOD OF: CHECK } _s \text{ MAX} \)
Sometimes requirements provide that a voltage must remain within
tolerance during some time interval. The following, which is equivalent
to a monitor statement, provides a means to insure that a measured voltage is
within tolerance over a selected time interval.

THE VOLTAGE AT <PIN 1> SHALL BE: MEASURE V ± V FOR A PERIOD
OF: CHECK S MIN

2.2.7 READ-YES AND READ-NO STATEMENT

At certain points within the ATP test sequence it may be appropriate
to solicit YES or NO responses from the test operator. If the operator
answers with an appropriate response the test should continue normally,
otherwise an abnormal termination is required. The READ-YES and READ-NO
statements both request YES or NO answer from the operator. In the case of
the READ-YES construct, the program continues normally if the operator answers
with a YES response. For the READ-NO statement program execution continues
normally provided a NO response is received.

These operator interaction statements appear within the ATP test
sequence as follows:

READ-YES
READ-NO
2.2.8 READ-NUM STATEMENT

A few measurements required within automatic test sequences must be performed manually by the test operator. The READ-NUM statement provides a simplified method of retrieving analog data from an operator and then checking the response to insure that it is within a desired value. When the READ-NUM statement is executed, operator response is solicited. The value of the response is checked for the range: \( V \pm V \). If the value is within range, the test sequence continues. Otherwise, operator response is requested a second time. If the retrieved analog data is again not within range the test sub-section terminates. The READ-NUM statement syntax is:

```
READ-NUM \( V \pm V \)
```

2.2.9 PRINT STATEMENT

Print statements allow the ATP programmer to send messages to an operator's console during ATP program execution. Either strings or virtual machine registers may be output using the print statement. The syntax of the print statement is:

```
PRINT <ITEM 1><ITEM 2> ... <ITEM n>
```

where \( n \) (number of items) is arbitrary, and \( <ITEM j> \) is either a string of the form

```
"<1 OR MORE CHARACTERS>"
```

or a register reference of the form RO...R15.

Character strings and register references may be listed in any desired order within the print statement. Semi-colons can be used within strings to effect a carriage-return-line feed operation.
2.2.10 SAVE STATEMENT

Certain test sequences require that an intermediate result be temporarily saved for use latter. For example, the voltage change statement may be used several statements after a reference point for the change value has been established. It is necessary to save the reference value until it is needed by the CHANGE statement. The P-machine registers R0 through R9 may be used to save intermediate data. Resultant values from MEASURE, ADJUST, or MONITOR statements may be saved with the value contained in the register in P-machine registers using the save statement. A voltage change statement may then be used later to check the changed value with the value contained in the P-machine register.

For example:

(CHECK VOLTAGE AT PIN J101A-58)

VOLTAGE AT J101A-58 SHALL BE:

MEASURE 15.0V +- 0.01V

(SAVE RESULT OF MEASUREMENT)

SAVE IN R4

.

.

.

.(ANY STATEMENTS WHICH DO NOT REFERENCE R4)

.

.

.

.(CHECK FOR 4.0 V CHANGE)

VOLTAGE CHANGE AT J101A-58 FROM

R4 SHALL BE: MEASURE 4.0V +- 0.10V
2.2.11 **WAIT STATEMENT**

A wait statement may be used whenever a delay in the automatic test sequence is required. The effect of a wait statement is to delay the test sequence for the specified period of time.

The WAIT statement is used as follows:

```
WAIT ___ SECONDS
```

2.2.12 **INLINE P-CODE**

The output of the ATP compiler is P-code for a virtual machine. The syntax of the P-code is defined in section 3.0 of this document. At times it may be necessary to write P-code for the virtual machine directly within the ATP test sequence. Certain special test cases may be difficult or impossible to implement using the standard ATP syntax. In these cases the user may utilize the full flexibility of the P-code machine by writing instructions inline. P-code instructions must be enclosed between the start statement $START-P$ and the ending statement $STOP-P$. All intervening instructions are copied directly to the p-code output file for assembly by the P-code assembler. The user must obey the syntax of the P-code assembler when he uses in-line P-code.
For example:

\[
\begin{align*}
&\text{(COMPILER STATEMENTS)} \\
&\text{(COMPILER STATEMENTS)}
\end{align*}
\]

$START-P$

\[
\begin{align*}
\text{INPUT} & \quad \% \ 0A092 \ G1 \ (\text{THIS IS A COMMENT}) \\
\text{LDA} & \quad R0 \\
\text{CONV} & \quad G1 \ C3 \\
\text{SUB} & \quad R2 \\
\text{STA} & \quad R4 \\
\text{GETDEC} \ \\
\text{STA} & \quad R5 \\
\text{OUTSTR} & \quad \"\text{THIS IS THE VALUE}\" \ R5 \\
$STOP-P$
\]

2.2.13 TRANSITION STATEMENT

Monitoring voltage waveform transitions may be accomplished using the transition statement. The statement insures that an initial voltage is maintained within tolerance until the time a transition is to occur (less time tolerance). The measured voltage must then change to the second voltage before the specified time (plus tolerance) has elapsed.
The syntax of the transition statement is:

\[
\text{MONITOR} [A] \text{ TRANSITION AT } \langle \text{PIN #} \rangle \text{ FROM } \_\_V \rightarrow \_\_V \\
\text{TO } \_\_V \rightarrow \_\_V \text{ AT TIME } \_\_S \rightarrow \_\_S
\]

2.2.14 MONITOR STATEMENT

A monitor statement is used whenever a voltage must be guaranteed to remain within a certain tolerance for a minimum period of time. The monitor statement continually reads the A/D associated with a pin and checks that the voltage is within \_\_V \rightarrow \_\_V. When the specified time in seconds has elapsed the statement terminates.

The syntax of the monitor statement is:

\[
\text{MONITOR} [\text{THE}] \text{ VOLTAGE AT } \langle \text{PIN #} \rangle \\
\text{OF } \_\_V \rightarrow \_\_V \text{ FOR } \_\_S
\]
3.0 DEFINITION OF P-CODE

The process of converting English language test statements into an automated test sequence can be greatly simplified through the use of an intermediate P-code machine. Defining a virtual computer tailored specifically to a class of applications can simplify and speed system implementation. Using a P-code interpretive approach for automatic testing of autothrottle computers is likely to result in a flexible, general purpose test system.

The primary goal of the virtual P-machine is to achieve the flexibility to handle the various test sequences required for testing Marconi autothrottle computers. Test sequences may be categorized into instructions which occur frequently and a set of infrequent special test cases. Handling both classes of test instructions efficiently may be accomplished using two levels of P-code.

High Level P-code

High level P-code instructions were designed to handle the most frequent test cases using a minimum of P-code instruction space. Included in this category are the instructions to measure and check voltages, open and close switches, set a voltage, wait a period of time, and check a voltage transition. These categories occur frequently, hence it is desirable to have P-codes available which directly accomplish the required tasks. High level instructions could be coded using the low level P-code instruction set. However, for maximum speed and efficiency the high level instructions should be used where possible. Note that in this documentation many of the high level instruction sequences are actually specified in terms of low level
instructions. This approach is used only to add clarity. High level instructions are actually hard-coded within the interpreter. The instructions utilize many of the same utilities used to implement the low level instructions.

**Low Level P-code**

In order to achieve the required test system flexibility it is appropriate to specify a set of low level P-code instructions. The low level instructions are used only to code special cases which cannot be handled by the high level instructions.

Low level P-codes were chosen to provide a minimum set of codes which can be used to implement most automatic test applications. Included are the P-codes to accomplish simple looping control, comparisons, error processing, console I/O and the various arithmetic functions. Since most test applications do not require expression evaluation and subroutine linkages the P-machine does not include stack oriented instructions. These facilities could be added later if needed for future applications.

3.1 **P-MACHINE IMPLEMENTATION**

Marconi's automatic test P-machine will be implemented as a 32-bit virtual computer. The actual implementation of the P-machine utilizes a TI-9900 16-bit minicomputer. An interpretive program emulates the 32-bit pseudo machine.
Data within the pseudo machine are, for the most part, represented as 32-bit quantities. A 32-bit accumulator (referred to as A) is used as the source and destination for the results of most instructions. Sixteen 32-bit general-purpose registers can be used for temporary data storage. A 16-bit program counter is used as a pointer for fetching instructions. The status register records the results of compare and arithmetic instructions. String registers maintain test section data and console input. See Figure 3.1.

The sixteen general registers are categorized into user and system registers. Registers 0 to 9 are reserved for the user as temporary storage registers. Registers 10 through 15 are used by the system to pass data between P-codes and have the following definition:

- Register 10 - Available for system use only.
- Register 11 - A/D Data converted to the real 32 bit format.
- Register 12 - The correct value for the voltage at a pin.
- Register 13 - The tolerance associated with the value in Register 12.
- Register 14 - Initial value for change P-code.
- Register 15 - Temporary for adjust device P-code.
Status Register (S)

EQ | LT | GT | Z | M

Program Counter (PC)

16 bits

Accumulator (A)

32 bits

General Purpose Registers (R0 - R15)

32 bits

String Input Buffer (SI)

80 bytes

Test Section Number (TS)

20 bytes

Figure 3.1 Pseudo Machine Registers
3.2 NUMBER REPRESENTATION

The automatic test P-code interpreter utilizes 32-bit floating point arithmetic for all calculations, thus providing all the accuracy needed by the ATP. The compiler produces 32-bit values, which correspond to the floating point format used by the TI 9900 microprocessor.

3.3 P-CODE FORMATS

Pseudo-instruction opcodes are specified in two categories. The majority are numbered sequentially from 01hex to 2A hex. These opcodes may be implemented easily with a branch table. All opcodes which reference the P-machine registers have the high order bit of the opcode byte set. Register related instructions all have a decimal value greater than 127. Register reference pseudo instructions consist of two nibbles. The high order nibble specifies the operation to be performed, while the low order nibble contains a register reference. In all cases the accumulator is used for operand 1, and the register reference (0 . . F) hex is used for operand 2.

<table>
<thead>
<tr>
<th>Register reference instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nibble 1</td>
</tr>
<tr>
<td>8--&gt;E</td>
</tr>
</tbody>
</table>

| Op code | Register reference |
3.4 **BRANCH INSTRUCTIONS**

The branch instructions allow the P-machine to transfer control to either a standard procedure or to another section of the P-code. The branch (BRA) instruction causes an unconditional transfer of program control. All other instructions in this category cause a change in the P-code execution sequence only if the appropriate bit in the status register is set (cleared). Each branch instruction includes a single byte opcode and a two byte offset. The two byte offset specifies where program control should transfer if the specified condition is true. If the high order bit of the offset is set, then the low order bits indicate which standard procedure to execute. When the high order bit is cleared then the low order bits contain an offset from the current PC. The offset may be positive or negative, it is represented in 2's complement form (in the low-order 15 bits).

The branch opcodes are represented:

<table>
<thead>
<tr>
<th>8 bit opcode</th>
<th>x</th>
<th>y (15 bits)</th>
</tr>
</thead>
</table>

If $x = 0$ then $y =$ offset in bytes from PC

If $x = 1$ then $y =$ standard procedure reference number
3.5 STANDARD PROCEDURES

SP1 - Record successful completion of test subsection contained in the test section register (TS). Begin execution of next subsection. Note that the entire subsection must be memory resident.

SP2 - Normal test termination. Ready equipment for next test. The BRA SP2 and STOP opcodes have identical effect.

SP3 - Abnormal subsection termination. ADC value out of tolerance. Print results contained in registers R10 - R13 and appropriate ID's.

SP4 - Abnormal subsection termination. Unable to set DAC to proper value. Print results contained in registers R10 - R13 and appropriate ID's.

SP5 - Abnormal subsection termination. Monitored voltage out of tolerance. Print elapsed time and results in registers R10 - R13 along with appropriate ID's.

SP6 - Abnormal subsection termination. Voltage of a transition is out of tolerance. Print elapsed time, registers R10 - R13, and appropriate ID's.

SP7 - Print normal results of a monitor instruction.

SP8 - Print normal results of an ADC measurement. Results contained in registers R10 - R13.
SP9 - Print normal results of a transition.

SP10 - Restart section.

SP11 - Abnormally terminate testing.

SP12 - Abnormal subsection termination. ADC value changed incorrectly.
Print results contained in registers R10 - R14 along with appropriate ID's.

SP13 - Abnormal subsection termination. Unable to set synchro to correct value. Print desired value and tolerance from registers R12 and R13.

SP14 - Abnormal subsection termination. Operator measured value is out of tolerance.

SP15 - Operator responded incorrectly
The following P-codes have been implemented:

<table>
<thead>
<tr>
<th>OPCODE (HEXADECIMAL)</th>
<th>MNEMONIC</th>
<th>MEANING</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>NOP</td>
<td>NO OPERATION</td>
</tr>
<tr>
<td>02</td>
<td>OUTSTR</td>
<td>OUTPUT ASCII STRING</td>
</tr>
<tr>
<td>03</td>
<td>GETSTR</td>
<td>INPUT ASCII STRING</td>
</tr>
<tr>
<td>04</td>
<td>CMPSTR</td>
<td>COMPARE ASCII STRING</td>
</tr>
<tr>
<td>05</td>
<td>OUTDEC</td>
<td>OUTPUT DECIMAL NUMBER</td>
</tr>
<tr>
<td>06</td>
<td>GETDEC</td>
<td>INPUT DECIMAL NUMBER</td>
</tr>
<tr>
<td>07</td>
<td>SETCLK</td>
<td>SET REAL TIME CLOCK</td>
</tr>
<tr>
<td>08</td>
<td>GETCLK</td>
<td>READ REAL TIME CLOCK</td>
</tr>
<tr>
<td>09</td>
<td>BRA</td>
<td>BRANCH UNCONDITIONALLY</td>
</tr>
<tr>
<td>0A</td>
<td>BLT</td>
<td>BRANCH IF LESS THAN</td>
</tr>
<tr>
<td>0B</td>
<td>BGT</td>
<td>BRANCH IF GREATER THAN</td>
</tr>
<tr>
<td>0C</td>
<td>BLE</td>
<td>BRANCH IF LESS THAN OR EQUAL</td>
</tr>
<tr>
<td>0D</td>
<td>BGE</td>
<td>BRANCH IF GREATER THAN OR EQUAL</td>
</tr>
<tr>
<td>0E</td>
<td>BEQ</td>
<td>BRANCH IF EQUAL</td>
</tr>
<tr>
<td>0F</td>
<td>BNE</td>
<td>BRANCH IF NOT EQUAL</td>
</tr>
<tr>
<td>10</td>
<td>BM</td>
<td>BRANCH IF MINUS</td>
</tr>
<tr>
<td>11</td>
<td>BP</td>
<td>BRANCH IF POSITIVE</td>
</tr>
<tr>
<td>12</td>
<td>BZ</td>
<td>BRANCH IF ZERO</td>
</tr>
<tr>
<td>13</td>
<td>BNZ</td>
<td>BRANCH IF NOT ZERO</td>
</tr>
<tr>
<td>14</td>
<td>ABS</td>
<td>TAKE ABSOLUTE VALUE OF ACCUMULATOR</td>
</tr>
<tr>
<td>15</td>
<td>NEG</td>
<td>NEGATE ACCUMULATOR</td>
</tr>
<tr>
<td>16</td>
<td>CLA</td>
<td>CLEAR ACCUMULATOR</td>
</tr>
<tr>
<td>17</td>
<td>STOP</td>
<td>NORMAL TERMINATION</td>
</tr>
<tr>
<td>8X NOTE 1</td>
<td>CMP RX</td>
<td>COMPARE ACCUMULATOR WITH REGISTER</td>
</tr>
<tr>
<td>18 NOTE 2</td>
<td>CMP #</td>
<td>COMPARE ACCUMULATOR WITH IMMEDIATE DATA</td>
</tr>
<tr>
<td>9X NOTE 1</td>
<td>STA RX</td>
<td>STORE ACCUMULATOR IN REGISTER</td>
</tr>
<tr>
<td>AX NOTE 1</td>
<td>LDA RX</td>
<td>LOAD ACCUMULATOR FROM REGISTER</td>
</tr>
<tr>
<td>19 NOTE 2</td>
<td>LDA #</td>
<td>LOAD ACCUMULATOR WITH IMMEDIATE DATA</td>
</tr>
<tr>
<td>BX NOTE 1</td>
<td>SUB RX</td>
<td>SUBTRACT REGISTER FROM ACCUMULATOR</td>
</tr>
<tr>
<td>1A NOTE 2</td>
<td>SUB #</td>
<td>SUBTRACT IMMEDIATE FROM ACCUMULATOR</td>
</tr>
<tr>
<td>CX NOTE 1</td>
<td>ADD RX</td>
<td>ADD REGISTER TO ACCUMULATOR</td>
</tr>
<tr>
<td>1B NOTE 1</td>
<td>ADD #</td>
<td>ADD IMMEDIATE TO ACCUMULATOR</td>
</tr>
<tr>
<td>DX NOTE 1</td>
<td>MULT RX</td>
<td>SIGNED MULTIPLY OF ACCUMULATOR BY REGISTER</td>
</tr>
<tr>
<td>1C NOTE 2</td>
<td>MULT #</td>
<td>SIGNED MULTIPLY OF ACCUMULATOR BY IMMEDIATE</td>
</tr>
<tr>
<td>EX NOTE 1</td>
<td>DIV RX</td>
<td>SIGNED DIVIDE OF ACCUMULATOR BY REGISTER</td>
</tr>
<tr>
<td>1D NOTE 2</td>
<td>DIV #</td>
<td>SIGNED DIVIDE OF ACCUMULATOR BY IMMEDIATE</td>
</tr>
<tr>
<td>1E</td>
<td>WAIT</td>
<td>WAIT THE SPECIFIED TIME</td>
</tr>
<tr>
<td>Code</td>
<td>Opcode</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>1F</td>
<td>OPEN</td>
<td>OPEN SPECIFIED SWITCH</td>
</tr>
<tr>
<td>20</td>
<td>CLOSE</td>
<td>CLOSE SPECIFIED SWITCH</td>
</tr>
<tr>
<td>21</td>
<td>INPUT</td>
<td>READ THE A/D</td>
</tr>
<tr>
<td>22</td>
<td>CONV</td>
<td>CONVERT THE A/D DATA TO REAL</td>
</tr>
<tr>
<td>23</td>
<td>MEAS</td>
<td>TAKE A MEASUREMENT</td>
</tr>
<tr>
<td>24</td>
<td>SETDEV</td>
<td>SET A DAC OR D/S TO A VALUE</td>
</tr>
<tr>
<td>25</td>
<td>CHECK</td>
<td>CHECK THE OUTPUT OF A DAC</td>
</tr>
<tr>
<td>26</td>
<td>MONV</td>
<td>MONITOR A PIN'S VOLTAGE</td>
</tr>
<tr>
<td>27</td>
<td>TRANS</td>
<td>MONITOR A PIN FOR A TRANSITION</td>
</tr>
<tr>
<td>28</td>
<td>INCD</td>
<td>INCREMENT DAC OR D/S VALUE</td>
</tr>
<tr>
<td>29</td>
<td>DECD</td>
<td>DECREMENT DAC OR D/S VALUE</td>
</tr>
<tr>
<td>2A</td>
<td>CHANGE</td>
<td>CHECK FOR ADC VALUE CHANGE</td>
</tr>
<tr>
<td>2B</td>
<td>ADJDEV</td>
<td>ADJUST THE SPECIFIED DEVICE</td>
</tr>
<tr>
<td>2C</td>
<td>READEV</td>
<td>READ THE DEVICE'S CURRENT VALUE</td>
</tr>
</tbody>
</table>

NOTE: 1) 'X' in the opcode field refers to one of the sixteen general registers (0--F hexadecimal).
2) # refers to immediate data.

The definitions of pseudo instructions are given in sections 3.6.1 through 3.6.51. These definitions employ the following notation:

<table>
<thead>
<tr>
<th>P-CODE PARAMETER</th>
<th>SYMBOLIC FORMAT</th>
<th>OBJECT FORMAT</th>
<th>REMARKS</th>
</tr>
</thead>
</table>
| 1) <ADC address> | % <1 to 6 characters> | 24 bit word | 4 spare bits 
4 CRU Index 
8-bit Multiplex number 
8-bit Channel number |
| 2) <device address> | 'DSO'...'DS255' | Bit 15 = 1 
Bit 8 = 0 
Bits 7-0 = INDEX | DIGITAL/SYNCHRO, movement in a clockwise direction. |
| | 'DSC0'...'DSC255' | Bit 15 = 1 
Bit 8 = 1 
Bits 7-0 = INDEX | DIGITAL/SYNCHRO, movement in a counter clockwise direction. |
| | 'DC0'...'DC255' | Bit 14 = 1 
Bits 7-0 = INDEX | DC DAC |
| | 'AC0'...'AC255' | Bit 13 = 1 
Bit 8 = 0 
Bits 7-0 = INDEX | AC DAC, in phase. |
3) <gain spec> 'G0'...'G3' 8 bits A/D gain
4) <conditioning spec> 'C0'...'C16' 8 bits hardware pre-conditioning
5) <switch address> 'S0'...'SFFFF' 16 bits
6) <value> +9999.999 32 bits floating point (the sign is optional.)
7) <tolerance> +/-9999.999 32 bits floating point (The '+' is required.)
8) <time> 9999.999 32 bits seconds
9) <step size> +/-99.999 32 bits step value - volts, deg, etc. (the sign is optional.)
10) <register> R0 ... R15
11) <string> 'up to 80 characters' 8-bit ASCII characters always ended in object format by a null byte.
12) <label reference> 10 character label ID 16-bit offset see section 3.4

The referenced label precedes a P-instruction. It must be terminated by a colon (:). At most 10 labels are permitted within any subsection. Moreover, for each label, there can be at most 10 forward branches.

3.6.1 NOP

Opcode: 06 hex

Results: No operation is performed

Status bits affected: None

3.6.2 OUTSTR <String>

Opcode: 02 hex

Results: Copies the ASCII following the opcode byte to active I/O devices until a null (0) termination character is detected.
3.6.3 GETSTR

Opcode: 03 hex

Results: Retrieves characters from the operator console until a termination character is received. All characters are copied into the string input (SI) buffer. The termination character causes the instruction to terminate with the remainder of the SI buffer blank filled.

Status bits affected: None

3.6.4 CMPSTR <String>

Opcode: 04 hex

Results: The immediate ASCII string is compared byte by byte with the contents of the string buffer. Characters are compared until the termination character is encountered within the immediate string or non-matching characters are detected. Example: A CMPSTR 'Y' instruction would match string buffer data of 'Y', 'YES', 'YEAH', etc. A CMPSTR 'YES' instruction would match any string buffer data containing 'YES' as the first three characters.

Status bits affected: EQ

3.6.5 OUTDEC

Opcode: 05 hex

Results: Converts the contents of the A register to a decimal ASCII string. The string is copied to the active I/O devices.

Status bits affected: None

3.6.6 GETDEC

Opcode: 06 hex

Results: Converted ASCII numeric data is moved to the A register. Retrieves numeric characters from the operator console until a termination char is received. The ASCII data is converted to 32 bit binary and stored in the A register.

Termination chars: Carriage return.
Status bits affected: Z, M

3.6.7 SETCLK
Opcode: 07 hex
Results: Set the test system clock with the contents of A.
Status bits affected: None

3.6.8 GETCLK
Opcode: 08 hex
Results: Set the accumulator (A) with the contents of the system clock.
Status bits affected: Z, M

3.6.9 BRA <label reference>
Opcode: 09 hex
Results: PC <- PC + offset
Status bits affected: None

3.6.10 BLT <label reference>
Opcode: 0A hex
Results: If status bit 'LT' is set, then PC <- PC + offset.
Status bits affected: None

3.6.11 BGT <label reference>
Opcode: 0B hex
Results: If status bit 'GT' is set, then PC <- PC + offset.
Status bits affected: None
3.6.12 BLE <label reference>

 Opcode: OC hex

 Results: If status bit 'EQ' or status bit 'LT' is set, then PC <-- PC + offset.

 Status bits affected: None

3.6.13 BGE <label reference>

 Opcode: OD hex

 Results: If status bit 'GT' or status bit 'EQ' is set, then PC <-- PC + offset.

 Status bits affected: None

3.6.14 BEQ <label reference>

 Opcode: OE hex

 Results: If status bit 'EQ' is set, then PC <-- PC + offset.

 Status bits affected: None

3.6.15 BNE <label reference>

 Opcode: OF hex

 Results: If status bit 'EQ' is not set, then PC <-- PC + offset.

 Status bits affected: None

3.6.16 BM <label reference>

 Opcode: 10 hex

 Results: If status bit 'M' is set, then PC <-- PC + offset.

 Status bits affected: None

3.6.17 BP <label reference>

 Opcode: 11 hex

 Results: If status bit 'M' is not set, then PC <-- PC + offset.

 Status bits affected: None
3.6.18 BZ <label reference>
 Opcode: 12 hex
 Results: If status bit 'Z' is set, then PC <--- PC + offset.
 Status bits affected: None

3.6.19 BNZ <label reference>
 Opcode: 13 hex
 Results: If status bit 'Z' is not set, then PC <--- PC + offset.
 Status bits affected: None

3.6.20 ABS
 Opcode: 14 hex
 Results: Take absolute value of register A. A <--- |A|
 Status bits affected: Z, M

3.6.21 NEG
 Opcode: 15 hex
 Results: Negate register A. A <--- -A
 Status bits affected: Z, M

3.6.22 CLA
 Opcode: 16 hex
 Results: Clear accumulator (A). A <--- 0
 Status bits affected: Z, M

3.6.23 STOP
 Opcode: 17 hex
 Results: Normal termination of a test.
 Status bits affected: None
3.6.24 CMP RX

Opcode: 8X hex X = 0..F

Results: Compare contents of A with contents of register X and set appropriate status bits.
If A = RX then 'EQ' set.
If A < RX then 'LT' set.
If A > RX then 'GT' set.
Status bits affected: EQ, LT, GT

3.6.25 CMP <value>

Opcode: 18 hex

Results: Compare contents of A with immediate data and set appropriate status bits.
If A = <value> then 'EQ' set.
If A < <value> then 'LT' set.
If A > <value> then 'GT' set.

Status bits affected: EQ, LT, GT

3.6.26 STA RX

Opcode: 9X hex X = 0..F

Results: Store A into register X. RX <-- A

Status bits affected: None

3.6.27 LDA RX

Opcode: AX hex X = 0..F

Results: Load A from register X. A <-- RX

Status bits affected: Z, M

3.6.28 LDA <value>

Opcode: 19 hex

Results: Load A with immediate data. A <-- <value>

Status bits affected: Z, M

3.6.29 SUB RX
Opcode: BX hex \( X = 0..F \)
Results: Subtract register \( X \) from A. A \( \leftarrow A - RX \)
Status bits affected: \( Z, M \)

3.6.30 SUB <value>
Opcode: 1A hex
Results: Subtract immediate data from A. A \( \leftarrow A - <value> \)
Status bits affected: \( Z, M \)

3.6.31 ADD RX
Opcode: CX hex \( X = 0..F \)
Results: Add register to A. A \( \leftarrow A + RX \)
Status bits affected: \( Z, M \)

3.6.32 ADD <value>
Opcode: 1B hex
Results: Add immediate data to A. A \( \leftarrow A + <value> \)
Status bits affected: \( Z, M \)

3.6.33 MULT RX
Opcode: DX hex \( X = 0..F \)
Results: Signed multiply of A by register \( X \). A \( \leftarrow A \times RX \)
Status bits affected: \( Z, M \)
3.6.34 MULT <value>

Opcode: 1C hex
Results: Signed multiply of A by immediate data. A ← A * <value>
Status bits affected: Z, M

3.6.35 DIV RX

Opcode: EX hex X = 0..F
Results: Signed divide of A by register X. A ← A/RX
Status bits affected: Z, M

3.6.36 DIV <value>

Opcode: 1D hex
Results: Signed divide A by immediate data. A ← A / <value>
Status bits affected: A, M

3.6.37 WAIT <time>

Opcode: 1E hex
Results: Program execution suspends until the 32 bit <time> reference has elapsed. The time is given in seconds.
Status bits affected: None

The WAIT instruction could be written in the low level p-codes as follows:

```
CLA
SETCLK (0 clock)
LOOP:
  GETCLK
  CMP <time>
  BLT LOOP
```

3.6.38 OPEN <switch address>
Opcode: 1F hex
Results: The switch specified by the two byte <switch address> is opened.
Status bits affected: None

3.6.39 CLOSE <switch address>
Opcode: 20 hex
Results: The switch specified by the two byte <switch address> is closed.
Status bits affected: None

3.6.40 INPUT <ADC address>, <gain spec>
Opcode: 21 hex
Results: Set the ADC to the specified gain and input a word from the ADC address into the A register and R10. R10 <--- A
Status bits affected: Z, M

3.6.41 CONV <gain spec>, <conditioning spec>
Opcode: 22 hex
Results: Convert the contents of A into a 'REAL' value by applying the gain spec and then the conditioning spec. Leave the result in A and R11. R11 <--- A
Status bits affected: Z, M
3.6.42 MEAS <ADC address>, <gain spec>, <conditioning spec>, <desired value>, <tolerance>

Opcode: 23 hex

Results: Set the ADC to the appropriate gain, input the ADC word, convert according to gain and conditioning, and check if within tolerance. If not in tolerance branch to standard error processing 3 with registers set:

R10 <--- (ADC data)
R11 <--- (converted ADC data)
R12 <--- (desired value)
R13 <--- (tolerance)

Status bits affected: Undefined

The MEASURE statement is equivalent to the following P-code sequence:

LDA <desired value>
STA R12
LDA <tolerance>
STA R13
INPUT <gain spec>, <ADC address>
CONV <gain spec>, <conditioning spec>
SUB R12
ABS
CMP R13
BGT SP3
OUTSTR "MEASURED: "
LDA R11
OUTDEC
OUTSTR " WITHIN TOLERANCE"

3.6.43 SETDEV <device address>, <value>, <tolerance>

Opcode: 24 hex

Results: set the specified device to the desired value after applying the implicit scale factor. If the D/S is being addressed, bit 8 of the device address specifies the direction of movement; i.e. if set, counterclockwise; if cleared, clockwise.

R12 <--- (value)
R13 <--- (tolerance)

Status bits affected: Undefined
3.6.44 CHECK <ADC address>, <gain spec>, <conditioning spec>

Opcode: 25 hex

Result: checks that the device is set to the value in R12 within tolerance contained in R13. If not in tolerance branch to standard error processing 4 with registers set:

R10 <--- (ADC data)
R11 <--- (converted data)
R12 <--- (desired value)
R13 <--- (tolerance)

Check is most commonly used after a 'SETDEV'.

Status bits affected: Undefined

3.6.45 MONV <ADC address>, <gain spec>, <conditioning spec>, <desired value>, <tolerance>, <time>

Opcode: 26 hex

Results: Check that the converted value read from the ADC address remains in tolerance for the time specified. If not, branch to standard error processing 5 with registers set as follows:

R10 <--- (ADC data)
R11 <--- (converted ADC data)
R12 <--- (desired value)
R13 <--- (tolerance)

Status bits affected: undefined after execution
The Monitor P-code can be written with the following low level p-code sequence:

```
LDA <desired value> (set up for possible error processing)
STA R12
LDA <tolerance>
STA R13
CLA
SETCLK
LOOP: INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB R12
ABS
CMP R13
BGT SP5
GETCLK
CMP <time>
BLT LOOP
```

(OK)

```
3.6.46 TRANS <ADC address>,<gain spec>, <conditioning spec>, <value 1>, <tolerance 1>, <value 2>, <tolerance 2>, <time>, <time tolerance>
```

Opcode: 27 hex

Results: Continuously monitor the specified ADC for value 1. The voltage should remain within tolerance 1 until at least time minus time tolerance. The voltage must change to value 2 within tolerance 2 before time plus time tolerance.

Status bits affected: Undefined

The Transition P-code can be written as the following sequence:

```
CLA
SETCLK
CMP <time> (t=0 ?)
BEQ REPEAT 1
REPEAT: INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB <value 1>
ABS
CMP <tolerance 1>
BGT SP6
GETCLK
SUB <time>
CMP < - time tolerance>
BLT REPEAT
```
REPEAT1:  GETCLK
SUB <time>
CMP <+time tolerance>
BGT SP6  (the actual P-machine)
          (will expand the time tolerance before 
          erroring to SP 6)
INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB <value 2>
ABS
CMP <tolerance 2>
BGT REPEAT1

(OK)

3.6.47 INCD <device address>, <step size>

Increment the value of the DAC or D/S at device address.

Opcode:  28
Status bits affected: None

3.6.48 DECD <device address>, <step size>

Decrement the value of the DAC or D/S at device address.

Opcode:  29
Status bits affected: None

3.6.49 CHANGE <register>, <ADC address>, <gain spec>, <conditioning spec>,
          <change value>, <tolerance>

Opcode:  2A hex
Results: Set the ADC to the appropriate gain, input the ADC value,
convert the value using the gain and conditioning specs.
The converted value shall have changed from the value
contained in the specified register by 'change value'
within tolerance. If not in tolerance branch to standard
error processing 12 with registers set:
  R10 <--- (ADC data)
  R11 <--- (converted ADC data)
  R12 <--- (change value)
  R13 <--- (tolerance)
  R14 <--- (initial value)

50
Status bits affected: Undefined

The CHANGE statement is equivalent to the following P-code sequence:

```
LDA <register>
STA R14
LDA <change value>
STA R12
LDA <tolerance>
STA R13
INPUT <ADC address>,<gain spec>
CONV <gain spec>, <conditioning spec>
SUB R14 (initial value)
SUB R12 (change amount)
ABS
CMP R13 (tolerance)
BGT SP12
```

3.6.50 ADJDEV <device address>, <ADC address>, <gain spec>, <conditioning spec>, <value>, <tolerance>, <step size>, <limit value>, <step time>

Opcode: 2B hex

Results: Adjust the device starting at its current value until the voltage at the ADC by step site per time address is at 'value' within tolerance. Note that <step size> can be positive or negative. If unable to set the device within tolerance branch to standard error processing with registers set:

```
R12 <--- (desired value)
R13 <--- (tolerance)
```

Status bits affected: Undefined

The adjust device P-code can be written using the following low-level P-codes:

```
READEV <device address>
STA R15 (loop counter)
LDA <value>
STA R12
LDA <tolerance>
STA R13
LOOP: INPUT <ADC address>, <gain spec>
CONV <gain spec>, <conditioning spec>
SUB R12 (desired value)
ABS
CMP R13 (tolerance)
BLE DONE WAIT <step time>
```
INCD <device address>, <step size> (increment/decrement device)
LDA R15
ADD <step size>
STA R15
LDA <step size>
BM NEG (NEGATIVE step size?)
LDA R15
CMP <limit value> (exceeded maximum value?)
BLT LOOP
BRA SP13
NEG:
LDA R15
CMP <limit value> (exceeded minimum value?)
BGE LOOP
BRA SP13
DONE:

3.6.51 READEV <device address>

Opcode: 2C hex

Results: Load the accumulator with the last value written to the specified device.

Status bits affected: Z, M
4.0 HARDWARE DEFINITION FILE/TABLES

The hardware definition file specifies the hardware configuration of the test set to the compiler. The file contains information relating to pins and their possible connections. Also contained in the file are data pertaining to DACS, voltage tolerances, and gains. All information required at compile time which relates to the hardware configuration is specified by this file.

The file should be supplied in the format as described in the following section. The ASCII source file will then be converted into an encoded form compatible with the compiler. For further information on the encoded file, its production, and its use, see Section 5.3 of this manual.
4.1 HARDWARE DEFINITION FILE - SOURCE FORM

4.1.1 OVERALL STRUCTURE

The hardware definition source file will be made up of several segments; each segment will contain a different type of information. For example, one segment will contain information about pins, their A/D addresses, types of connections, etc.; one segment will contain information pertaining to DACs; one segment will have voltage tolerances and ranges; etc.

Each segment will begin with an identifier line which will tell what type of information is in the segment and how many lines of that type of information are following. Thus, the form of the file will be essentially:

```
[ IDENTIFIER LINE
  INFORMATION LINE 1
  ...
  INFORMATION LINE a
]
```

```
[ IDENTIFIER LINE
  INFORMATION LINE 1
  ...
  INFORMATION LINE b
]
```

```
[ IDENTIFIER LINE
  INFORMATION LINE 1
  ...
  INFORMATION LINE c
]
```
The ordering of segments within the file is unimportant with the exception that the segment containing pin information should be the last segment in the file. (Since the pin segment is last, the number of information lines in it may be left blank on its identifier line).

4.1.2 FORMATTING OF LINES

Except for the identifier line, each segment will have a different line format. No line will exceed 80 characters in length. Unless otherwise indicated all information should be left justified within the given column boundaries. Also there should be no embedded blanks in any string; in other words, if a data item is left justified within the given column boundaries and is shorter than the allowable width, only trailing blanks will be considered valid. For example, if the allowable width is 8 characters, the data item is "DS01" and it is to be left justified

\[ D S 0 1 \quad _ { _ _ _ } \]

is the only correct entry.

\[ D S 0 1 \quad _ { _ _ _ } \]

would be incorrect because of the embedded blank.

\[ _ { _ _ _ } D S 0 1 \quad _ { _ _ _ } \]

would be incorrect because the data is not left justified.
4.1.2.1 FORMAT OF IDENTIFIER LINE

The identifier line, being the first line in every segment, will contain a code indicating which segment is upcoming and the number of lines in that segment that follow the identifier line.

FORMAT

2A2  COLUMNS 1-4 should contain the four character type code (left justified)

I3   COLUMNS 9-11 should indicate the number of lines following of the given type (right justified)

Valid type codes (ASCII strings) are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>TYPE OF INFORMATION IN THAT SEGMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAC_</td>
<td>DAC numbers and types, high and low values</td>
</tr>
<tr>
<td>GAIN</td>
<td>A/D gains and corresponding ranges</td>
</tr>
<tr>
<td>COND</td>
<td>conditioning factors</td>
</tr>
<tr>
<td>ANGL</td>
<td>angle tolerance</td>
</tr>
<tr>
<td>VOLT</td>
<td>voltage tolerances and ranges</td>
</tr>
<tr>
<td>PIN_</td>
<td>pin identifiers, their addresses, valid connections, switches involved, etc.</td>
</tr>
</tbody>
</table>
4.1.2.2 FORMAT OF "DAC " SEGMENT RECORDS

This segment gives a list of all DACS and their associated ranges. All information lines within the "DAC_" segment should be of the following format.

FORMAT

A2  COLUMNS 1-2  will indicate the type of DAC; "DC" or "AC" are valid entries (ASCII strings)
I3  COLUMNS 3-5  will contain the DAC number; an integer right justified
F10.4 COLUMNS 9-18 will contain a real number (volts) specifying the low value for the DAC
F10.4 COLUMNS 25-34 will contain a real number (volts) specifying the high value for the DAC
F10.4 COLUMNS 41-50 will contain a real number (degrees) specifying the phase of the DAC (NOTE: AC DACS only; this field should be left blank for DC DACS)

Ordering of records within the "DAC_" segment should result in having all DC DAC's in sequential order by number followed by AC DAC's in sequential order by number. DAC numbers should begin at zero and increase by increments of one.
4.1.2.3 FORMAT OF "GAIN" SEGMENT RECORDS

This segment contains gain specs and associated ranges. Gains should begin at zero and increase by increments of 1; their associated ranges should be in descending order.

FORMAT

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>COLUMN 1</td>
</tr>
<tr>
<td>F6.3</td>
<td>COLUMNS 9-14</td>
</tr>
</tbody>
</table>

4.1.2.4 FORMAT OF "COND" SEGMENT RECORDS

This segment contains the hardware conditioning factors. Conditioning specs should be in ascending order in increments of one beginning at zero.

FORMAT

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I2</td>
<td>COLUMNS 1-2</td>
</tr>
<tr>
<td>E14.7</td>
<td>COLUMNS 9-22</td>
</tr>
</tbody>
</table>

4.1.2.5 FORMAT OF "ANGL" RECORD (D/S DEFAULT TOLERANCE)

The ANGL segment contains exactly one information line.

FORMAT

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5.3</td>
<td>COLUMNS 1-5</td>
</tr>
</tbody>
</table>
4.1.2.6 FORMAT OF "VOLT" SEGMENT RECORDS

This segment indicates the default voltage tolerances for associated voltage ranges -- either AC or DC. "VOLT" records should be in the following format with DC ranges and tolerances followed by AC ranges and tolerances.

**FORMAT**

<table>
<thead>
<tr>
<th>Type</th>
<th>Low Voltage of Range</th>
<th>High Voltage of Range</th>
<th>Default Voltage Tolerance for Associated Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>A2 COLUMNS 1-2</td>
<td>F8.3 COLUMNS 9-16</td>
<td>F8.3 COLUMNS 25-32</td>
<td>F5.3 COLUMNS 41-45</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(positive value)

4.1.2.7 FORMAT OF "PIN " SEGMENT RECORDS

This segment contains pin identifiers, the A/D addresses (made up of CRU pointer, state, channel) possible connections that can be made to each pin, and A/D scale factors.

**FORMAT**

<table>
<thead>
<tr>
<th>Pin Identifier</th>
<th>CRU Pointer</th>
<th>State</th>
<th>Channel</th>
</tr>
</thead>
<tbody>
<tr>
<td>5A2 COLUMNS 1-10</td>
<td>A1 COLUMNS 17</td>
<td>A2 COLUMNS 25-26</td>
<td>A2 COLUMNS 33-34</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pin identifier. Up to 10 ASCII characters constitute a valid pin identifier. Left justified.

Ex. COLUMN 1 2 3 4 5 6 7 8 9 10
J 1 0 1 A - 2

CRU pointer. 1 Hex digit.

State 2 Hex digits

Channel 2 Hex digits
5A2

COLUMNS 41-50 Type of connection. Left Justified with trailing blanks. Valid types are:
1. a 10 character pin identifier
2. DC DACS (ex. "DCO", "DC1", ..., "DC255")
3. AC DACS (ex. "ACO", "AC1", ..., "AC255")
4. D/S PINS (ex. "DS01", "DS02", "DS03")
5. LOGIC DISCRETES:
   "LD+10", LD0, "LD+12", "OPEN"
6. MANUAL (operator controlled pins)
7. MONITOR
8. continuation (designated as "**"). The type
   of connection is the same as that of the
   preceding information line. The information
   in columns 57-59, 65-68, and 73-74 lists an
   additional switch closure required for
   connecting the given pin to the specified
   type.

A1,A2 COLUMNS 57-59 Switch identifier. 3 Hex digits.

2A2 COLUMNS 65-68 Switch state. "OPEN" or "CLOS"

I2 COLUMNS 73-74 A/D conditioning factor. Integer right
     justified. ("00" to "99" decimal)

Figure 1 illustrates the source form of the "PIN" segment.
<table>
<thead>
<tr>
<th>J101A-109</th>
<th>8</th>
<th>2A</th>
<th>16</th>
<th>MONITOR</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8</td>
<td>2A</td>
<td></td>
<td>OPENC</td>
<td>950</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LD+28</td>
<td>950</td>
</tr>
<tr>
<td>J101A-110</td>
<td>8</td>
<td>3A</td>
<td>16</td>
<td>MONITOR</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>3A</td>
<td></td>
<td>OPENC</td>
<td>A38</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LDO</td>
<td>A38</td>
</tr>
<tr>
<td></td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>A39</td>
</tr>
<tr>
<td>J101A-111</td>
<td>5</td>
<td>93</td>
<td>14</td>
<td>MONITOR</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 1: HARDWARE DEFINITION FILE - SOURCE FORM
4.2 HARDWARE DEFINITION FILE - CONVERTED FORM

The source file after conversion becomes three disk files: x.D1, x.D2, x.D3, where x is a file-name prefix. These files contain connection data, pin identifiers, and information needed in common by other programs, respectively. (See 5.3)

These files may be represented symbolically as interconnected tables of pin identifiers and their valid connections, along with necessary switch data to actually make the connections.

4.2.1 COMMON BLOCK FILE (x.D3)

This sequential file is used to contain information which is to be stored in the common blocks of the ATP compiler. This information includes:

1) A table of sorted pin id's and associated indexes into the random-access connection file (x.D1)
2) DAC specifications
3) A/D gain ranges
4) A table of conditioning factor indexes and associated conditioning factors
5) The default D/S tolerance
6) The voltage tolerances

4.2.2 CONNECTION FILE (x.D1)

This is a random-access file containing the information given in the "PIN__" segment (see 4.1.2.7). Given a specified pin id, the ATP compiler accesses this file by using an index retrieved from its common block table of sorted pin ids and associated indexes. This index points to the set of related information for the given pin.
4.2.2.1 FORMAT OF CONNECTION_FILE_(x.D1)

The connection file specifies a pin's A/D address as well as the connections that can be made to that particular pin. The connection file contains two types of records. One record specifies the A/D address and the number of the other type of records immediately following.

The second type of record contains the actual type of connection, switch settings, etc. Thus, there is only one record of the first type for each pin and a variable number of records of the second type for each pin (the number being specified in the first record).

Ex.  

<table>
<thead>
<tr>
<th>A/D ADDR</th>
<th>CONNECTION INFO</th>
<th>CONNECTION INFO</th>
<th>CONNECTION INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A/D ADDR</td>
<td>2</td>
<td>CONNECTION INFO</td>
<td>CONNECTION INFO</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The form of the first record:

1 16-bit word filled with 1's (to indicate record type)
3 16-bit words containing the ASCII A/D address
1 16-bit word containing the number of "connection-type" records following

The form of the "connection-type" records:

1 16-bit word containing a connection type code
2 16-bit words containing an ASCII switch ID
1 byte containing an index into the conditioning factor table
1 byte containing the switch status
   (OPEN =0  CLOS =1)
1 16-bit word containing the connection value

The first word of the second type of record, the connection type field, contains a type code which identifies the connection as follows:

=0  Continuation of previous records (used to specify more switch closures)
=1  Connection with another pin (NOTE: In this case, the associated connection value (the last word of this record) is an index into the pin id table where the second pin's identifier is stowed.)

=2  Connection with logic discretes — including open circuits (NOTE: An open circuit is indicated by type =2 with a value of 99 in the associated connection value.)

=3  Connection with a digital/synchro

=4  Connection with an AC DAC

=5  Connection with a DC DAC

=6  Monitor connection

=7  Manual connection
4.2.3 PIN ID FILE (x.D2)

The pin id file is a random access file which contains the identifiers of pins which connect to other pins. Each record in it is of the form:

5 16-bit words containing 10 character ASCII pin identifier

The pin id file is associated with entries of the connection file with a connection type of 1. The connection values for these entries are indexes into the appropriate record in the pin id file.
5.0 OPERATING PROCEDURES

Before the ATP compiler can be run, several files must be prepared: the source ATP file and the converted hardware definition files. The source ATP file contains a test specification procedure written in the ATP language. The converted hardware definition files contain information on the hardware configuration being used and are produced by a translator from the source hardware definition file. This translator needs to be rerun each time a change is made to the source hardware definition file.

To access the compiler the user must simply type: COMPILE. The files to perform the compilation should be found under the current directory.

The three following sections, 5.1, 5.2, and 5.3 contain more specific information on how to use the compiler. These three sections give additional details on how to properly answer the questions asked by the compiler, the assembler, and the hardware definition translator.
5.1 COMPILER USAGE

To invoke the ATP compiler a macro called COMPIL has been created. Once the compiler has been invoked it will then ask the user a series of questions:

  = = => INPUT SOURCE FILE NAME:

Here the name of the file containing the ATP test specification procedure should be typed followed by a carriage return. The compiler will then respond

  = = => SOURCE FILE IS: Source File Name

DO YOU WISH TO HAVE A SOURCE LISTING PRODUCED? (Y/N)

Here a simple Y for YES or N for NO will suffice to answer the question as to whether or not the user desires to have a source listing of the ATP test specification procedure.

  = = => TEST SPECIFICATION NUMBER NOT FOUND IN SOURCE

  = = => DO YOU WISH TO STOP? (Y/N)
This message will be printed only if a test specification number is not found on the first line of the ATP test specification procedure source file. If the user wishes to process the file, Y (for YES) should be typed followed by a carriage return. If the user wishes to stop at this point N (for NO) may be typed followed by a carriage return.

\[
= = = \Rightarrow \text{TEST SPECIFICATION NUMBER: Test specification number}
\]

Here the compiler simply displays the test specification number that was found in the source file. If no number was found then blanks will be printed for the test specification number.

After the compiler has completed processing the input file, it will print a message indicating the number of errors found, if any.
5.2 \textbf{ASSEMBLER USAGE}

Once the compiler has completed its processing the assembler is invoked and will ask the following series of questions:

\[ \text{= = = = => DO YOU WANT AN OBJECT FILE PRODUCED [Y/N]}? \]

Here the user is asked to respond to the question with Y for YES or N for NO. The object file will contain the assembled P-code. If the user responds with Y then the assembler will request a file name:

\[ \text{= = = = => SPECIFY THE OBJECT FILE NAME:} \]

\[ \text{= = = = => DO YOU WANT A LISTING OF THE SOURCE P-CODE [Y/N]}? \]

Again a yes or no question is posed to the user as to whether or not a listing of the source P-code is desired.
If the user responds with a Y, then the assembler will ask the following YES or NO question as to whether or not the user wishes to have the object code printed on the listing with the P-code:

```
= = = = => DO YOU WANT THE OBJECT CODE PRINTED ON THE LISTING [Y/N]?
```

After the assembler has completed processing the input file it will print a message indicating the number of errors found by the assembler. At this point the process of compilation and assembly have been completed.

5.3 HARDWARE DEFINITION CONVERSION PROCESS

The hardware definition source file is converted into a form suitable for the ATP compiler through the program "HCONV". "HCONV" is typed at the terminal to initiate the conversion process. The user will then be asked to specify the source file name with a .DF extension, (i.e., "HW747.DF"). The name of this file must be identical to the name cited in the HARDWARE DEFINITION FILE statement.

The conversion process consists of two parts. First all tab characters are removed from the source file; this new source file becomes HW747.DT. This file is then converted into three files used by the ATP compiler: HW747.D1, HW747.D2, HW747.D3. These files are the random access file containing pin connection information, the random access file containing pin ID's which connect to other pins, and the sequential file containing common block information, respectively.

Any changes made to the source form of the file requires "HCONV" to
be run in order to incorporate the changes. However, if no changes
are made to the hardware configuration and no errors result when executing
"HCONV", then the program needs only to be executed once.
APPENDIX 1 - BNF

The syntax of the language processed by the ATP compiler is given here in Backus Naur Form or BNF. BNF is a commonly used notation to write grammars which specify the syntax of programming languages.

BNF notation is used as a concise and explicit method of defining the language formally. English is not suitable for formal language definition due to its vagueness and because it leads to ambiguous definitions.

The BNF as the defining language is called the syntactic meta-language. The meta-language is a language which is used to talk about another language or the object language. The object language in this case is the language to be processed by the ATP compiler. Symbols of the object language are called terminals. Symbols of the meta-language which denote strings in the object language are called nonterminals.

Nonterminals in BNF are expressed as names written in corner-brackets '<<>'. The syntax of the object language is given as a set of "rewriting rules". These rules consist of a nonterminal on the left-hand side of the rewriting rule, the ::= sign indicating that left-hand side of the rule is replaced by the right-hand side, and the sequence of terminals and/or nonterminals which make up the right-hand side of the rewriting rule.

Within the right-hand side of the rule, alternative ways of rewriting a given nonterminal are separated by a vertical bar, |, (read "or"). Also on the right side items written in curly brackets, {}, may be repeated zero or more times. Any item which may be considered optional within a rule has been enclosed in square brackets. For example, in the case of the rewriting rule:

\[
\langle \text{SUBSECTION NUMBER} \rangle ::= \text{DIGIT STRING} \{ \text{DIGIT STRING} \}
\]
the nonterminal is SUBSECTION NUMBER (enclosed in corner-brackets) with ::= indicating that the nonterminal SUBSECTION NUMBER is to be replaced by the right-hand side of the rule. The right-hand side of the rule consists of a string of appropriate digits separated by periods. In this case the period followed by a digit string may be repeated zero or more times giving clause numbers of the form 5, 5.5, 5.4.3 etc.

In the case of the STATEMENT rewriting rule (see the BNF) the nonterminal <STATEMENT> may be replaced by any one of the types of statements found on the right-hand side of the rule. Here, <STATEMENT> may be replaced by an <ADJUST STATEMENT> or by an <SET STATEMENT> or by a <CONNECT STATEMENT> and so on.

Not every form of every statement type has been kept in the object language. One of a kind statements can easily be converted to the standard format developed here, and their omission from the language will help to avoid overburdening the language and the compiler which must process it. Variations on the standard statement types have also been limited for the same reason.
BNF FOR THE ATP LANGUAGE

<TEST PROCEDURE> ::= <TEST SPECIFICATION STATEMENT> <HARDWARE DEFINITION FILE STATEMENT> <SECTION> {<SECTION>}

<SECTION> ::= BEGIN <SECTION NAME> {<SUBSECTION>} END

<SECTION NAME> ::= <SECTION NUMBER> <CHARACTER STRING>

<SECTION NUMBER> ::= <DIGIT STRING> [. <DIGIT STRING> ]

<SUBSECTION> ::= BEGIN <SUBSECTION NUMBER> <STATEMENT> {<STATEMENT>} END

<SUBSECTION NUMBER> ::= <DIGIT STRING> . <DIGIT STRING> [. <DIGIT STRING> ] [<VARIANT>]

<VARIANT> ::= / <DIGIT STRING> /

<STATEMENT> ::= <ADJUST STATEMENT> | <SET STATEMENT> | <CONNECT STATEMENT> | <DISCONNECT STATEMENT> | <VOLTAGE STATEMENT> | <WAIT STATEMENT> | <OPEN CIRCUIT STATEMENT> | <OPERATOR INTERACTION STATEMENT> | <SAVE STATEMENT> | <VOLTAGE CHANGE STATEMENT> | <START P-CODE STATEMENT> | <STOP P-CODE STATEMENT> | <MONITOR STATEMENT>

<CHARACTER STRING ::= <CHARACTER> { <CHARACTER> }

<CHARACTER> ::= any printable ASCII character

<DIGIT STRING> ::= <DIGIT> { <DIGIT> }

<DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Test Specification Statement> ::= TEST SPECIFICATION NUMBER = <CHARACTER STRING>

<HARDWARE Definition File Statement> ::= HARDWARE DEFINITION FILE = <CHARACTER STRING> .DF

<Adjust Statement> ::= ADJUST SIGNAL AT <PIN#> BY <STEP SIZE> PER <TIME> IN A <SIGN> DIRECTION UNTIL THE VOLTAGE AT <PIN#> CHANGES TO <VOLTS> ↔ <VOLTS> |

ADJUST SIGNAL AT <PIN#> BY <STEP SIZE> PER <TIME> UNTIL THE VOLTAGE AT <PIN#> IS <VOLTS> ↔ <VOLTS>

ADJUST SYNCHRO TRANSMITTER BY <STEP SIZE> PER <TIME> SECONDS UNTIL THE
VOLTAGE AT <PIN#> IS <VOLTS> ± <VOLTS>

<SIGN> ::= POSITIVE | NEGATIVE

<SET STATEMENT> ::= 
    SET THE SYNCHRO TRANSMITTER TO <ANGLE> ± <ANGLE> [CLOCKWISE | COUNTERCLOCKWISE]

<CONNECT STATEMENT> ::= 
    CONNECT <PIN#> TO <CONNECT OBJECT> |
    CONNECT <SYNCHRO OBJECT> TO <PIN#> ;
    CONNECT <VOLTS>, <FREQUENCY> [,,<ANGLE>] TO <PIN#>

<CONNECT OBJECT> ::= <PIN#> | <VOLTS> | A SIGNAL SET TO <VOLTS>

<SYNCHRO OBJECT> ::= S1 | S2 | S3

<DISCONNECT STATEMENT> ::= 
    DISCONNECT <PIN#> FROM <DISCONNECT OBJECT> |
    DISCONNECT SIGNAL FROM <PIN#>

<DISCONNECT OBJECT> ::= <VOLTS> | <PIN#> | SYNCHRO TRANSMITTER

<OPEN CIRCUIT STATEMENT> ::= OPEN CIRCUIT <PIN#>

<REMOVE STATEMENT> ::= 
    REMOVE <REMOVE SUBJECT I> BETWEEN <PIN#> AND <PIN#> |
    REMOVE <REMOVE SUBJECT II> FROM <PIN#> ;
    REMOVE <PIN#> FROM <VOLTAGE>

<REMOVE SUBJECT I> ::= SIGNAL | CONNECTION | <VOLTS>, <FREQUENCY> [,,<ANGLE>]

<REMOVE SUBJECT II> ::= <VOLTAGE> | <PIN#>

<VOLTAGE> ::= SIGNAL | <VOLTS>
<VOLTAGE STATEMENT> ::= 
[THE] VOLTAGE AT THE FOLLOWING POINTS SHALL BE:
  {<PIN#> MEASURE <VOLTS> ← <VOLTS> }

[THE] VOLTAGE AT <PIN#> SHALL BE:
  MEASURE <VOLTS> ← <VOLTS>
  [WITHIN A PERIOD OF: CHECK <TIME> MAX ;
  FOR A PERIOD OF: CHECK <TIME> MIN] ;

[THE] VOLTAGE BETWEEN <PIN#> AND <PIN#> SHALL BE:
  MEASURE <VOLTS> ← <VOLTS>

<OPERATOR INTERACTION STATEMENT> ::= <PRINT STATEMENT> ; READ-YES ;
  READ-NO ; READ-NUM <VOLTS> ← <VOLTS>

<PRINT STATEMENT> ::= PRINT {<REGISTER> ; "<CHARACTER STRING>"} [;]

<SAVE STATEMENT> ::= SAVE [IN] <REGISTER>

<VOLTAGE CHANGE STATEMENT> ::= 
  VOLTAGE CHANGE AT <PIN#> FROM <REGISTER> SHALL BE:
  MEASURE <VOLTS> ← <VOLTS>

<WAIT STATEMENT> ::= WAIT <TIME>

<START P-CODE STATEMENT> ::= $START-P

<STOP P-CODE STATEMENT> ::= $STOP-P

<MONITOR STATEMENT> ::= 
  MONITOR [A] TRANSITION AT <PIN#> FROM <VOLTS> ← <VOLTS>
  TO <VOLTS> ← <VOLTS> AT TIME <TIME> ← <TIME> ;
  MONITOR [THE] VOLTAGE AT <PIN#>
  OF <VOLTS> ← <VOLTS> FOR <TIME>

<STEP SIZE> ::= <REAL NUMBER>

<TIME> ::= <REAL NUMBER> <SECONDS>

<VOLTS> ::= <REAL NUMBER> <POTENTIAL>
<ANGLE> ::= <REAL NUMBER> <DEGREES>
<PIN #> ::= <PIN PREFIX> <CHARACTER STRING>
<REGISTER> ::= R <REGISTER NUMBER>
<REAL NUMBER> ::= fortran-compatible integer or real number
<SECONDS> ::= SECONDS | SECOND | SEC | S
<POTENTIAL> ::= VOLTS | VOLT | V
<DEGREES> ::= DEGREES | DEGREE | DEG | D
<PIN PREFIX> ::= character other than digit, +, or -
<REGISTER NUMBER> ::= 0 | 1 | 2 ... | 15
APPENDIX 2 - GENERATED P-CODE

This section describes the P-code generated from each high-level ATP statement. Note that only a single 'OPEN' or 'CLOSE <switch address>' is used in each example even though multiple switch closures would be generated if defined in the hardware definition file.

1) ADJUST SIGNAL AT <PIN #1> IN A <SIGN> DIRECTION BY <STEP SIZE> PER <TIME> SECONDS UNTIL THE VOLTAGE AT <PIN #2> CHANGES TO __ V +- __ V

when <SIGN> is POSITIVE

ADJDEV DC?, <PIN #2 A/D ADDR>, <GAIN SPEC> <CONDITIONING SPEC>, <VALUE>, <TOLERANCE>, <STEP SIZE>, <MAX LIMIT VALUE>, <STEP TIME>

when <SIGN> is NEGATIVE

ADJDEV DC?, <PIN #2 A/D ADDR>, <GAIN SPEC>, <COND SPEC>, <VALUE>, <TOLERANCE>, <NEGATIVE STEP SIZE>, <MIN LIMIT VALUE>, <STEP TIME>

2) ADJUST SIGNAL AT <PIN #1> BY <STEP SIZE> PER <TIME> SECONDS UNTIL THE VOLTAGE AT <PIN #2> IS __ V +- __ V

ADJDEV DC?, <PIN #2 A/D ADDR>, <GAIN SPEC>, <COND SPEC>, <VALUE>, <TOLERANCE>, <STEP SIZE>, <LIMIT VALUE>, <STEP TIME>

3) ADJUST SYNCHRO TRANSMITTER BY <STEP SIZE> PER <TIME> SECONDS UNTIL THE VOLTAGE AT <PIN #1> IS __ V +- __ V

ADJDEV DS?, <PIN # 1 A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, <VALUE>, <TOLERANCE>, <STEP SIZE>, <LIMIT VALUE>, <STEP TIME>
4) SET THE SYNCHRO TRANSMITTER TO __ DEGREES ← __ DEGREES CLOCKWISE

SETDEV DS?, <VALUE>,<TOLERANCE>

SET THE SYNCHRO TRANSMITTER TO __ DEGREES ← __ DEGREES COUNTERCLOCKWISE

SET DS?C, <VALUE>,<TOLERANCE>

5) CONNECT <PIN #> TO <PIN #>

CLOSE <SWITCH ADDRESS>

6) CONNECT <PIN #> TO ___V

SETDEV DC__, <VALUE> <DEFAULT TOLERANCE>

CLOSE <SWITCH ADDRESS>

CHECK <PIN # A/D ADDRESS>, <GAIN>, <COND>

7) CONNECT <PIN #> TO A SIGNAL SET TO ___V

SETDEV DC__, <VALUE>, <DEFAULT TOLERANCE>

CLOSE <SWITCH ADDRESS>

CHECK <PIN # A/D ADDRESS>, <GAIN>, <COND>

8) CONNECT <S1>S2S3> TO <PIN #>

CLOSE <SWITCH ADDRESS>

9) CONNECT ___V, ___Hz [, ___DEGREES] TO <PIN #>

SETDEV AC__, <VALUE>,<TOLERANCE>

CLOSE <SWITCH ADDRESS>

CHECK <PIN # A/D ADDRESS>, <GAIN>, <COND>
10) DISCONNECT <PIN #> FROM _V
OPEN <SWITCH ADDRESS>

11) DISCONNECT <PIN #> FROM <PIN #>
OPEN <SWITCH ADDRESS>

12) DISCONNECT <PIN #> FROM SYNCHRO TRANSMITTER
OPEN <SWITCH ADDRESS>

13) DISCONNECT SIGNAL FROM <PIN #>
OPEN <SWITCH ADDRESS>

14) OPEN CIRCUIT <PIN #>
OPEN <SWITCH ADDRESS>

15) REMOVE SIGNAL BETWEEN <PIN #> AND <PIN #>
OPEN <SWITCH ADDRESS (USE ONE OF THE PINS)>

16) REMOVE CONNECTION BETWEEN <PIN #> AND <PIN #>
OPEN <SWITCH ADDRESS (USE ONE OF THE PINS)>

17) REMOVE _V _HZ [__ DEGREES] BETWEEN <PIN #> AND <PIN #>
OPEN <SWITCH ADDRESS (USE ONE OF THE PINS)>

18) REMOVE VOLTAGE FROM <PIN #>
OPEN <SWITCH ADDRESS>
19) REMOVE <PIN #> FROM <PIN #>
OPEN <SWITCH ADDRESS>

20) REMOVE <PIN #> FROM SIGNAL
OPEN <SWITCH ADDRESS>

21) REMOVE <PIN #> FROM _V
OPEN <SWITCH ADDRESS>

22) [THE] VOLTAGE AT THE FOLLOWING POINTS SHALL BE:
{<PIN # i> MEASURE _V +- _V}
MEAS <PIN # 1 A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, <VOLTAGE>, <TOLERANCE>
.
.
MEAS <PIN # N A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, <VOLTAGE>, <TOLERANCE>

23) [THE] VOLTAGE AT <PIN #> SHALL BE: MEASURE _V +- _V
MEAS <PIN # A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, <VOLTAGE>, <TOLERANCE>

24) [THE] VOLTAGE AT <PIN #> SHALL BE: MEASURE _V +- _V WITHIN A PERIOD OF:
CHECK _S MAX
TRANS <PIN # A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, 0, ← 0, <VOLTAGE>, <TOLERANCE>, 0, ← _S
25) [THE] VOLTAGE AT <PIN#> SHALL BE: MEASURE _V +- _V FOR A PERIOD OF:
CHECK _S MIN
MONV <PIN # A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, VOLTAGE>, <TOLERANCE>, _S

26) [THE] VOLTAGE BETWEEN <PIN #1> AND <PIN #2> SHALL BE: MEASURE _V +- _V
MEAS <PIN #1 A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, _V, +- _V

27) PRINT RO "STRING" R1
LDA RO
OUTDEC
OUTSTR "STRING"
LDA R1
OUTDEC

28) READ-YES
GETSTR
CMPSTR "Y"
BNE SP15

29) READ-NO
GETSTR
CMPSTR "N"
BNE SP15

30) READ-NUM _V +- _V
GETDEC
SUB <VOLTAGE>
ABS
CMP <TOLERANCE>
BGT SP14
31) SAVE [IN] R#  
    LDA R11  
    STA R#  

32) VOLTAGE CHANGE AT <PIN #> FROM R# SHALL BE: MEASURE __V +/- __V  
    CHANGE R#, <PIN # A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>,  
    <VOLTAGE>, <TOLERANCE>  

33) WAIT __ SECONDS  
    WAIT <TIME>  

34) $START-P  
    (NO GENERATED P-CODE)  

35) $STOP-P  
    (NO GENERATED P-CODE)  

36) MONITOR [A] TRANSITION AT <PIN #> FROM __V1 +/- __V1 TO __V2 +/-  
    __V2 AT  
    TIME __S +/- __S  
    TRANS <PIN # A/D ADDRESS>, <GAIN SPEC>, <COND SPEC>, <V1 VALUE>,  
    <V1 TOLERANCE>, <V2 VALUE>, <V2 TOLERANCE>, <TIME>, <TIME TOLERANCE>  

37) MONITOR THE VOLTAGE AT <PIN #> OF __V +/- __V FOR __S  
    MONV <A/D ADDRESS>, <COND SPEC>, <VOLTAGE>, <TOLERANCE>, <TIME>
APPENDIX 3 - ERROR MESSAGES

Different phases of the ATP compiler and its companion assembler generate error messages indicating that something in the input source file has been encountered which does not conform to a legal language construct (syntax errors) or does not constitute a valid use of the language with the given Hardware Definition Table (semantic errors). This section of documentation gives a complete listing of the error messages which may be encountered while using the ATP compiler and assembler and a brief explanation of what will cause such an error to occur.

A.3.1 COMPILER ERRORS

1) UNMATCHED QUOTE MARKS BEGINNING ON LINE NUMBER

This error indicates that either the character string on the given line number has not been properly enclosed in quote marks or that the string exceeds the 80 character maximum size limit.

2) ILLEGAL UNITS ON LINE __ ** TOKEN = _______________

This error message is generated when the unit token following a tolerance is not one of the four legal unit token types (VOLTS, SECONDS, DEGREES, HZ) or a legal unit token alias. This message may also be generated when the unit tokens of a value and its tolerance are not of the same type.

3) ERROR IN NUMBER ON LINE __ ** TOKEN = _______________

This error indicates that the number token found on the stated line number does not constitute a legal number. This error may also be generated if there is a blank between the sign and the number.

4) ERROR COMMENT TOO LONG OR THE PARENTHESIS ARE NOT MATCHED *** SEARCH BEGAN ON LINE# __ AND ENDED ON LINE #_

This error indicates that a closing right parenthesis was not found but, was searched for within the range of the given line numbers.
5) ERROR ON LINE # TOO MANY END STATEMENTS FOR THE NUMBER BEGINS

This error message will be printed when more END statements are encountered than BEGIN statements. This error is most likely caused by an extra END statement in the source file.

6) ERROR ON LINE # TOO MANY DIGITS TO THE RIGHT OF THE DECIMAL POINT

When this message is presented it indicates that more than three digits to the right of the decimal point have been encountered. Three decimal digits is the maximum number allowed.

7) ERROR ON LINE # TOO MANY DIGITS TO THE LEFT OF THE DECIMAL POINT

When this error message is printed it indicates that a number has been encountered that is out of the range of -32767 to +32767.

8) SYNTAX ERROR ON LINE NUMBER OCCURRED WHILE PROCESSING TOKEN

A syntax error of this type will occur when the high-level statement being processed does not follow the correct syntax. Check the BNF of the statement to determine what is wrong.

9) INVALID PIN-1 ID REFERENCE ON LINE #

The first specified pin in an ATP statement could not be found in the hardware definition table.

10) INVALID PIN-2 ID REFERENCE ON LINE #

The second specified pin in an ATP statement could not be found in the hardware definition table.

11) UNABLE TO SET DAC TO REQUESTED VALUE ON LINE #

A DAC of the type and value requested is not available for connection to the requested pin.

12) CANNOT MEASURE DESIRED VALUE WITH A/D ON LINE #

The requested measurement is out of range of the A/D after scaling was applied.

13) UNABLE TO SET UP D/S TO DESIRED VALUE ON LINE #

The D/S cannot be set to the requested value.
14) USER MUST SUPPLY TIME TOLERANCE ON LINE __.
   A tolerance is not optional for time specifications.

15) INVALID PARSE TABLE ENTRY DETECTED ON LINE __.
   The integrity of the parse tables is suspect. This error should never occur.

16) INVALID H/W DEFINITION DATA ON LINE __.
   The H/W definition table entry for the requested pin is incomplete or invalid.

17) NO MONITOR SPEC. IN H/W DEFINITION ON LINE __.
   It is impossible to complete the requested measurement on the currently active pin. The H/W definition table does not contain a monitor entry.

A.3.2 ASSEMBLER ERROR MESSAGES

1) ****ASSR - INVALID FIRST SYMBOL
   This message indicates that a P-code statement began with an invalid or unrecognizable beginning symbol.

2) ****ASSR - INVALID LABEL REFERENCE
   This indicates that the object of a branch P-code operator was never declared within the subsection containing the branch operator. Check that the label is actually declared within the same subsection. In the case of a reference to a standard procedure name, check that it is a valid name (SPO TO SP14).

3) ****ASSR - INVALID HEX DIGIT
   This indicates that a A/D address or switch identifies contained something other than a valid hex digit ('0' to '9', 'A' through 'F').

4) ****ASSR - TOO MANY LABEL REFERENCES
   More than 10 forward references are made to the same label within the current subsection.

5) ****ASSR - MORE THAN 10 LABELS
   More than 10 forward different label declarations were specified in the current subsection.
6) ****ASSR - DUPLICATE LABELS

Two or more declarations of the same label were specified in
the current subsection.

7) ****ASSR - NOT A LITERAL STRING

The P-code parameter specified was not a literal string. A
literal string should be a string of alphanumeric characters
enclosed within single or double quotes. (e.g., "XYZ123
456")

8) ****ASSR - NOT A VALUE PARAMETER

The P-code parameter specified was not a value parameter. A
value parameter should be a single real number (e.g., -14.253)

9) ****ASSR - NOT A TOLERANCE PARAMETER

The P-code parameter specified was not a tolerance
parameter. A tolerance parameter should be a single real
number with a '+' prefix with no embedded blanks (e.g.,
+-1.234).

10) ****ASSR - NOT A GAIN PARAMETER

The P-code parameter specified was not a gain parameter.
The gain parameters are GO through G3.

11) ****ASSR - NOT A CONDITIONING PARAMETER

The P-code parameter specified was not a conditioning factor
index. The conditioning parameters are CO through C16 and
should have been declared in the hardware definition file.

12) ****ASSR - NOT A REGISTER PARAMETER

The P-code parameter specified was not a register reference.
The valid registers are R0 through R15.

13) ****ASSR - NOT AN A/D REFERENCE

The P-code parameter specified was not an A/D reference. An
A/D reference should be a 6 digit hex number preceded by an
'%' (e.g., %#0123AB)

14) ****ASSR - NOT A DEVICE REFERENCE

The P-code parameter specified was not a device reference.
A device reference should be a 'DS', 'DC', or 'AC' followed
optionally by a 'C' followed by the device number (e.g. DSO,
AC1, ACC1, DC255)
15) ****ASSR - NOT A SWITCH REFERENCE
The P-code parameter specified was not a switch reference. A switch reference should be a 'S' followed with up to four hex digits (e.g., SABC, S123, SOF1)

16) ****ASSR - INVALID REGISTER REFERENCE
The P-code parameter specified was an invalid register reference. The register references are R0 through R15.

17) ****ASSR - INVALID VALUE OR TOLERANCE
An invalid real number was specified. Note that embedded blanks are not allowed within a real number and its sign.

18) ****ASSR - INVALID SWITCH NUMBER
More than four hex digits were specified for a switch reference.

19) ****ASSR - INVALID GAIN NUMBER
A gain was specified outside of the range G0 to G3.

20) ****ASSR - INVALID CONDITIONING NUMBER
A conditioning factor index was specified outside of the range from C0 to C16.

21) ****ASSR - INVALID ADC NUMBER
More than six hex digits were specified for an A/D reference.

22) ****ASSR - LITERAL OVER 255 CHARS
More than 255 characters were specified within a literal string.

23) ****ASSR - SYMBOL EXCEEDS 10 CHARACTERS
A symbol (other than a literal string) was detected which exceeded 10 characters.

24) ****ASSR - COMMENT EXCEEDS 500 CHARS
A comment was detected that exceeded the 500 character maximum.

25) ****ASSR - COMMENT NOT TERMINATED
An end of file or I/O error occurred while processing a comment.
26) ****ASSR - ERROR READING P-CODE FILE

An error occurred from a FORTRAN read statement while accessing the source P-code file.

27) ****ASSR - SUBSECTION TOO LARGE

The size of the object code for the current subsection exceeded 600 bytes. Split the subsection into two subsections.

28) ****ASSR - INVALID P-CODE PARAMETER

The specified P-code parameter could not be identified as any known symbol.

A.3.3 ERROR MESSAGES FROM HARDWARE DEFINITION SOURCE FILE CONVERSION

The following error messages can occur during the conversion of the hardware definition file. Processing of the file will continue unless otherwise indicated; however, any error message will necessitate the process of conversion until no errors are generated.

1) ****ERROR - INVALID RECORD TYPE
   - HEADER RECORD SKIPPED

This error indicates that one of the identifier lines in the file had an invalid type. The valid types are DAC__, GAIN, COND, ANGL, VOLT, PIN__, or a blank line (allowed for readability). Any other abbreviations or the correct types not occurring in columns 1-4 will cause this error.

2) ****ERROR - NO PIN SECTION FOUND

This error indicates that no pin information was found in the hardware definition source file.

3) ***DC DAC ENTRIES OUT OF ORDER

This error occurs if the DC DACs are not numbered sequentially in increments of the beginning at zero.

4) ***AC DAC ENTRIES OUT OF ORDER

This error occurs if the AC DACs are not numbered sequentially in increments of one beginning at zero.
5) ****INVALID AC PHASE

This error occurs when an AC phase is specified as something other than 0 or 9.5. The phase will be set to 0 should this occur.

6) ****INVALID ENTRY IN DAC TABLES

This error occurs if something other than "DC" or "AC" is in columns 1-2. (Note: Blank lines are allowed for readability so they will not cause this error.)

7) ****ERROR - GAIN ENTRIES OUT OF ORDER

This error occurs if the gains are not numbered sequentially from zero in increments of one.

8) ****ERROR - GAIN RANGES NOT IN DESCENDING ORDER

This error occurs if the gain ranges are not in descending order; i.e., the largest range value must correspond to the smallest gain spec.

9) ****ERROR - CONDITIONING ENTRIES OUT OF ORDER

This error occurs if the conditioning specs are not numbered sequentially from zero in increments of one.

10) ****INVALID VOLTAGE TOLERANCE

This error occurs if the low voltage value is greater than or equal to the high voltage value specified in a line of the voltage tolerance segment.

11) ****INVALID VOLTAGE TOLERANCE RECORD

This error occurs if "DC" or "AC" is not in columns 1-2 in the voltage tolerance segment lines. (Note: Blank lines are allowed.)

12) ****OVER 50 CONNECTION

Indicates that there were more than fifty connections for the specified pin.

13) ****ERROR IN READING INPUT FILE

A FORTRAN error condition occurred while reading the specified pin.
14) ****TOO MANY PINS IN H/W DEFINITION

More than 500 pins were specified. Processing stopped.

15) ****ERROR IN WRITING TO CONNECTION FILE

A FORTRAN error occurred on the write to the random access connection file for the specified pin.

16) ****NOT A VALID HEX DIGIT

This error indicates something other than a valid hex digit was indicated for the CRUPTR, STATE, or CHANNEL for the specified pin.

17) ****ERROR IN WRITING TO PIN ID FILE

A FORTRAN error occurred while writing to the random access pin ID file.

18) ****WRITE ERROR IN WRITING OUT FILE - .D3

A FORTRAN error occurred while trying to write to HW747.D3 (the file which contains the pins, pointers, and DAC information for the common blocks).
APPENDIX 4 - P-CODE OBJECT FILE FORMAT

A P-code Object File (hereinafter PCOF) is a series of ASCII characters, in coded lines that is to (eventually) reside on a TI digital cassette. A PCOF has a logical and a physical structure.

A4.1 PHYSICAL STRUCTURE

A PCOF is made up of a series of lines. The beginning of the file is the beginning of medium, and is known as the beginning of information (BOI). The end of file (EOF) is an EOF line. Within the file, a PCOF is also divided into groups. A group is terminated by an end-of-group (EOG) line.

A line is just a series of ASCII characters terminated by a carriage return character (CR). Because of TI limitations, a line may be at most 86 characters long, including the CR. There are two basic types of lines, the coded line and the separator line.

The separator line is just a separator, followed by a CR. For example, the end-of-group (EOG) line is the group separator (ASCII character GS), followed by the CR. The end-of-file (EOF) line is the file separator (ASCII character FS), followed by the CR.

A coded line is just 1 to 85 printable ASCII characters (codes 20H to 7EH) followed by a CR.

A4.2 LOGICAL STRUCTURE

A PCOF is logically divided into sections, with each section divided into subsections. It is headed by an identification group.
A4.2.1 FIRST GROUP

The first group on a PCOF is the PCOF identification group. This group is a series of lines, as described below.

<table>
<thead>
<tr>
<th>Line nr.</th>
<th>description</th>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>date &amp; time of compilation</td>
<td>YY/MM/DD.HH.MM.SS</td>
</tr>
<tr>
<td>2</td>
<td>date &amp; time of assembly</td>
<td>YY/MM/DD.HH.MM.SS</td>
</tr>
<tr>
<td>3</td>
<td>test specification number</td>
<td>AAAAAAAA</td>
</tr>
<tr>
<td>4</td>
<td>variants, separated by commas, ended with a period</td>
<td>NN,NN,...,NN.</td>
</tr>
</tbody>
</table>

A4.2.2 SECTIONS

A section begins with a beginning of section line (BOSL) and is the first line of a group. It ends with the EOG line which is before the next section, or it ends with the EOF.

FORMAT for the BOSL:

.SSSSSSSSSSSSSSSSSSSS

A4.2.3 SUBSECTIONS

A subsection usually begins with the first line of a group, and always terminates with an EOG line. The only time a subsection identification line (BOSSL) is not the first line in a group is when the subsection is the first subsection in a section, in which case the BOSL precedes the BOSSL.

The format for BOSSL is:
identifies this line as a BOSSL

VV -variant identifier. A blank means that this subsection applies to all variants.

SSS...S -subsection number

After the BOSSL, the object code for this subsection follows, organized into Object Code Lines. (OCL)

The format for an OCL is:

:NNAAARRBBBBBBB....BBBBCC

: -identifies this as an OCL

NN -number of data bytes on this line

AAAA -address of first data byte

RR -record type
  00 - absolute address record

BB...BB -each two characters is a data byte

CC -checksum

The 8 bit truncated sum of all ASCII-encoded-bytes on this line should be zero

Note that each "number" takes two characters and is an ASCII-encoded-hex representation of a byte. The address AAAA is just two bytes.