GEORGIA INSTITUTE OF TECHNOLOGY — OFFICE OF CONTRACT ADMINISTRATION

PROJECT ADMINISTRATION DATA SHEET

[x] ORIGINAL [ ] REVISION NO. ____

Project No. E-24-669 GTRI/~~GIT~~ DATE 10 / ~~21~~ / 83

Project Director: D. B. Young School/~~Lab~~ ISyE

Sponsor: MERADCOM, Procurement & Production Directorate, Ft. Belvoir, VA

Type Agreement: D.O. #0017 under BOA DAAK70-79-D-0087 AIRMICS (OCA File #42)

Award Period: From 9/29/83 To ~~9/30/84~~ 3/30/85 (Performance) ---- (Reports)

Sponsor Amount:	This Change 5/15/85	Total to Date
Estimated:	$ 111,090	$ 111,090
Funded:	$ 111,090	$ 111,090

Cost Sharing Amount: $ None Cost Sharing No: N/A

Title: Stand-alone Operation - Automated Project Management System (Phase IV)

ADMINISTRATIVE DATA

OCA Contact William F. Brown x4820

1) Sponsor Technical Contact:

Capt. Richard D. Lee
ADP Officer
AIRMICS
O'Keefe Bldg.
Georgia Tech, Atlanta, GA 30332
(404) 894-3107

2) Sponsor Admin/Contractual Matters:

Mr. T. A. Bryant
ONR RR
206 O'Keefe Bldg.
Georgia Tech
Atlanta, GA 30332
(404) 881-4213

Defense Priority Rating: DO-S1 Military Security Classification: ____

(or) Company/Industrial Proprietary: N/A

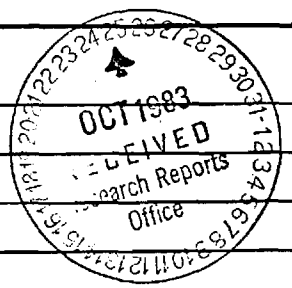
RESTRICTIONS

See Attached Gov't Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval – Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with Gov't

COMMENTS:



COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting
~~Research Security Services~~
Reports Coordinator (OCA)
Research Communications (2)
~~OCA~~
~~Library~~
Project File
Other I. Newton

GEORGIA INSTITUTE OF TECHNOLOGY OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

Date 12/30/86

Project No. E-24-669 School/~~XXX~~ ISYE

Includes Subproject No.(s) N/A

Project Director(s) D. B. Young GTRC / ~~GXX~~

Sponsor MERADCOM

Title Stand-alone Operation-Automated project Management System (Phase IV)

Effective Completion Date: 5/15/85 (Performance) _____ (Reports)

Grant/Contract Closeout Actions Remaining:

[x] None

No further reporting requirements per telecon Bill Brown/Sponsor.

[ ] Final Invoice or Final Fiscal Report

[ ] Closing Documents

[ ] Final Report of Inventions

[ ] Govt. Property Inventory & Related Certificate

[ ] Classified Material Certificate

[ ] Other _____

Continues Project No. _____ Continued by Project No. _____

COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting
Procurement/GTRI Supply Services
Research Security Services
~~Reports Coordinator (OCA)~~
Legal Services
Library
GTRC
Research Communications (2)
Project File
Other I. Lashley
A. Jones
R. Embry

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

Atlanta, Georgia 30332 (404) 894-2300

3 January 1984

M E M O R A N D U M

TO: Cpt. Larry Frank, AIRMICS

FROM: Donovan Young

SUBJECT: E-24-658 Retrofit Software Delivery

Enclosed is a tape and diskette constituting the current version of GITPASE, dated 12/20/84, along with a "Table of Current GITPASE Features" which summarizes all the features which have been added since the previous delivery. This version contains all features specified in the contract, plus several extra features beyond those – explanatory text, inverse text, milestones, and printed reports. Those features that might cause the code to be too large for the PC version can be easily removed according to the procedures given in the Table. The Table also documents all the new variables, which are also, of course, documented in the code itself. User documentation will be given in the Users Manual to follow, and in the separate specifications "WH1 – Printed Reports" and "WH2 – Milestones," which will be appended to the Users Manual.

The new version contains corrections to all of the miscellaneous bugs that had been identified in the previous version. With this delivery, we have fulfilled all requirements for contract E-24-658 execept for a new edition of the Users Manual.

With respect to the current contract E-24-669 (PC version), I recommend you use this version as the VAX version from which to download code for the PC version.

cc: Ron Rardin, Purdue Univ.
Clark Weeks
Pat Heitmuller
M. E. Thomas
E-24-669 File

TABLE OF CURRENT GITPASE FEATURES 12/20/84

IFEAT()	FEATURE
1	EXPLANATORY TEXT
2	INVERSE TEXT
3	PRINTED REPORTS
4	FILE VERIFICATION
5	MILESTONES

HOW TO REMOVE EXPLANATORY TEXT

1) Set IFEAT(1) to 0
2) Dummy all common block arrays ending in FT1 to a size of 1 (these common blocks exist in CONTRL.F, HIERH.F, and NETSAV.F).
3) Delete all common block nonsubscripted variables in common blocks and ending in FT1.
4) Remove all references to these variables in INITLZ.
5) Dummy or delete routines in PH3RUT2 as directed by the header comment blocks.

HOW TO REMOVE INVERSE TEXT

1) Set IFEAT(3) to 0
2) Modify print routine in PH3NIM to only give 'UNIMPLEMENTED' error message.
3) Remove PH3*3.FOR from compilation and link.

HOW TO REMOVE MILESTONES

1) Set IFEAT(5) to 0
2) Remove or dummy routines in PH3TRN2,3 as appropriate (headers will indicate).
3) Delete All/CFT5/ITEMS from CONTRL.F.
4) Drop the dimension of ISGLE in netsav.F to 1.
5) Drop the dimension of ITGLE in trnwk.F to (1,4).

NEW VARIABLES

Documentation of New Contrl.F Variables

Feature 5

LIMG - maximum number of milestones (Global Events)
NUMGLE - current number of milestones
IGNAM - names of milestones
IGMNET - member network numbers of milestones
IGMNNM - member network names of milestones
IGMPSM - finish time in determinant network
IGMHDR - Leader pointer to IGMNET, IGMNNM
IKGLE - Active global event
KOMGLE - Last touched global event
IGLETX - Explanatory text for milestone

New Working Storage Files

KURRPT - name of file used for report
NETINF - temporary storage of network information
RSTINF - unused
IRPTFL - name of working storage file for reports
LURP1 - unit for original copy of explanatory text
LURP2 - unit for final copy of explanatory text

Feature 1 ends in

L - number of text records of explanatory text
W - number of interger words per record of explanatory text

First four characters

IACT - Activity explanatory text
IRES - Resource explanatory text
IPLN - Plan explanatory text
IFIL - File explanatory text

IFILTX - Storage place for file explanatory text

Documentation of New HIERH.F Variables

IHRSTX - resource explanatory text workspace for hierarchial computations.

New Variables NETSAV.F

IACTTX - activity explanatory text
IRESTX - resource explanatory text
IPLNTX - plan explanatory text
ISGLE - global event flags 0, none + Dependant Network,
- Determinant Network.

Documentation of Variables in PRINT.F

PRINT.F is a new common block used in all printed reports.

IPAGE - storage set aside for one page of a printed report in memory
IPLOC - current location on the page
IPRTPW - the line printer's page width
IPRTPL - the line printer's page length
ISTCOL - the column in printer pages of this page of output
ISTORD - the order in which the pages are output to the new report file

Documentation of Variables In New Common SCHPRT.F

SCHPRT.F is a new common block used exclusively for Schedule Reports.

IALIST - list of activities using a given resource
IALNUM - number of activities using a given resource
ICHECK - list of activities to check for criticality
ICNCOT - cycle number of cycle where resource consumption rates change
IELYST - earliest start time possible in this network
ILTEFN - latest possible finish time in this network
IPCRT - list of tight predecessors for a given activity
ISCRIT - activity numbers of critical activities
ISCRT - list of tight successors of a given activity
NCHECK - number of activities that need to be checked for criticality
NCONOT - real value of rates of consumption of a resource (See ICNCOT)
NCRIT - number of critical activities
NLPRED - number of actual predecessors (by direct data reference) of a given activity
NLPSUC - number of actual successors (by direct data reference) of a given activity
NUMENS - unused
NUMCYC - number of cycles of a given resource that appear in this network's time range
NUMP - number of tight predecessors of a given activity
NUMPD - number of general predecessors of a given activity
NUMS - number of tight successors of a given activity
NUMSC - number of general successors of a given activity
RCONOT - value of resource consumption
RUTL1 - cycle by cycle availability of resource
RUTL2 - cycle by cycle consumption of resource
NAPD - activity numbers of general predecessors of a given activity
NAPDN - precedence ration between activity and predecessor
NASC - activity numbers of general successors of a given activity
NASCN - precedence ration between activity and successor

Documentation of Variables in TPRINT.F

TPRINT.F is used for variables exclusively used in the transition mode report.

JNETNM - network names
JNETNB - network numbers
JDEPT - depth from Root Node: Root node is depth 1
JPAR - network number of parent network
JLINO - line number when network name is to be displayed
JISPR - flags denoting whether vertical bars need drawing
in the network diagram 1 yes, 0 no
JNUMNT - number of networks in file
JMXDPT - maximum depth of file
JLINES - number of lines in report

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

Atlanta, Georgia 30332-0205

(404) 894-2300

May 22, 1984

M E M O R A N D U M

TO: Cpt. Larry Frank, AIRMICS

FROM: Donovan Young

SUBJECT: E-24-669 Software Development Plan for Stand-Alone APMS, and Milestone A and B Specifications

Georgia Tech hereby submits a software development plan for Phase IV of APMS, subject to your approval. Details of the plan are given in the attached memo.

Under this plan we are immediately issuing the first two of eight specifications for programming, level-1 testing, and program documentation to be performed by the Army. Upon Army completion of the tasks detailed in these eight specifications, and testing by Georgia Tech, the software for the IBM PC version will be complete. (The limited-capability portable version is not covered in these specifications except to the extent that it will be built from a subset of the code.)

The first specification, Milestone A, calls for the Army to compile GITPASE common blocks and service routines on an IBM PC, create and exercise a dummy calling program for the service routines, and to create and execute a test program that uses the common blocks.

A very important secondary purpose of the work under Milestone A is to familiarize new programmers with FORTRAN coding standards to assure clean code that will be easy to read, debug, document, enhance, and transport. Our experience with both external and internal programming personnel has been that it is difficult to enforce these standards, but that deviations from them have been costly in both time and money. A large portion of the final effort in Phase III has been to clean up hasty kludges that caused more problems than they solved.

Therefore I urge you to be very strict in adherence to the FORTRAN standards, and to set aside sufficient time for programmers to study them.

The second specification Milestone B, calls for the Army to implement new routines for collecting keyed input, to incorporate BASIC code now running on the Chromatics.

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

Atlanta, Georgia 30332-0205

(404) 894-2300

May 15, 1984

M E M O R A N D U M

TO: Donovan Young, Project Director

FROM: Ronald L. Rardin, Consultant

SUBJECT: IBM PC Software Development Plan

In accordance with my contract CI-E-24-669, I am hereby submitting a plan for software design of the IBM PC version.

Our current project calls for development of FORTRAN software that provides a version of GITPASE similar to the Phase III deliverable and operational entirely of IBM PC's (or XT's). As you know, the Phase III code includes a FORTRAN portion with all model intelligence and program control, together with a BASIC program executing terminal input and output (I/O). The IBM PC version will essentially replace BASIC with new FORTRAN.

The present FORTRAN contains subroutines that are called by functional routines to call on BASIC for I/O. For example OUTGR4 is called to produce 4-coordinate graphics primitives, and OUTKEY called to solicit keyed input. These routines send communications to BASIC which, in turn, accomplishes the requires outputs and/or solicits and formats needed inputs.

I propose to accomplish the conversion by replacing such routines as OUTGR4 and OUTKEY by new ones with the same names and parameters. However, the new routines will directly call out input and output (via SCION/HALO) instead of sending communications. This approach is a "plug compatible" one in that functional routines need not be modified in any substantial way. Thus linking new code to old FORTRAN should be a simple matter of erasing the old routines (and their subordinates) and substituting new ones (with their service routines).

The "plug compatible" approach also facilitates software development and testing. Since the routines being replaced are elementary primitives, they have very little interaction with the remainder of the code except through their call parameters. Thus, they can be coded and tested independent of the main code. "Dummy" test main

programs can be used to call out and test the routines quickly. Only at the end of the effort does actual mating with the main program need to occur.

Our schedule is still tentative because of uncertainties about hardware. However, I anticipate issuing specifications for a conversion in the following sequence and schedule:

Milestone A: Download Phase III Service Routines and Commons

Transfer from the VAX Phase III source code all included common files and the service routines of files and compile on IBM PC (issued with this memo).

Milestone B: Keyed Input

Prepare and test in stand alone format all routines to accept and edit keyed input. (Issued with this memo).

Milestone C: Graphic Primitives

Prepare and test in stand alone format all routines to initialize graphics, to convert coordinates and to execute graphics (not string) primitives. (Issued one week after hardware is operational).

Milestone D: String Primitives

Prepare and test in stand alone format all routines to display primitive strings. (Issued 1/2 week after Milestone C).

Milestone E: Complex Displays

Prepare and test in stand alone format all needed routines to create such complex displays and screens for modes, menus, error messages, status summaries, help, etc. (Issued 1 1/2 weeks after Milestone C).

Milestone F: Locator Input

Prepare and test in stand alone format all routines to collect a queue of locator (mouse) inputs. (Issued 2 1/2 weeks after Milestone C).

Milestone G: Download Main Phase III

Transfer all needed code in the delivered Phase III VAX code to the IBM PC and compile (issued upon delivery of Phase III unless experimentation with hardware shows significant reformatting of displays is required).

Milestone H: Mating

Link and test in full functional operation the old code of Milestones A and G with new code of Milestone C-F (issued late Summer 1984).

If hardware acquisition and checkout proceeds rapidly, there is no reason Milestone A-G should not be in your hands by mid-June, 1984. The only unknown other than hardware is whether reformatting will be required (see Milestone G).

I hope you find these plans satisfactory.

Milestone A: Download Phase III Service Routines and Commons

By Ronald L. Rardin

The first required step in IBM PC conversion will be familiarization of AIRMICS staff with FORTRAN coding standards, and service routines and common storage areas of the Phase III FORTRAN code.

A.1 Download

Table A1 lists common areas used by the Phase III code that will be transferred to the IBM PC. All are presently '.F' files on the VAX. Transfer these files to the IBM PC and demonstrate their compatibility by $INCLUDEing all in a test program, then compiling and executing it. The only conversion that should be required is dividing overlong areas into two or more blocks with similar names (e.g. NETSAV may becomes NETSV1, NETSV2, etc.)

Table A2 lists minor service routines employed throughout the Phase III code. Download each from the indicated VAX module and compile and link on the IBM PC. Then create a dummy calling program to exercise all routines. The calling program should include all required commons and call first on INITLZ (to define internal constants). Use compiler options which default INTEGER's and REALs to '*4.'

Compilation of the service routines should be extremely easy. However, minor changes may be required for IBM PC FORTRAN. For example, INCLUDE commands in column 7 must become $INCLUDE's of column 1. Do not under any circumstances change the number, type, order or significance of the calling parameters for the routines.

A.2 New Code Conventions

In anticipation of new code for later milestones, create and $INCLUDE new common block(s) /IBM PC/. All variables and arrays needed for new code should be located there. Do not modify the definition of any Phase III common blocks.

Prior to beginning coding, programmers should also study the attached FORTRAN coding standards. These standards are generally enforced throughout the Phase III code, and should be maintained in the conversion. Please note in particular prohibitions on implicit functions and string operations, the requirement that passed scalars not be constants or expressions, and the indentation and commenting concepts. See module PH3RPN.FOR of the Phase III code for convention examples.

TABLE A1. GITPASE COMMON BLOCKS TO BE TRANSFERRED

Filename	Purpose
CONTRL.F	All system control variables
HIERH.F	Work areas for hierarchial processing
NETSAV.F	Saved data on the current network
NETWRK.F	Temporary data on the current network
SCHWRK.F	Work areas for schedule computation
TRNRWK.F	Work areas for Transition Window operations

TABLE A2. SERVICE ROUTINES TO DOWN LOAD

Name	File Location	Function
MATINT	PH3RUT.FOR	match item in list
MAXINT	PH3SED.FOR	max of integers
MIWINT	PH3SED.FOR	min of integers
DECINT	PH3RPN.FOR	decode string to integer
DECREL	PH3RPN.FOR	decode string to real
A1A2	PH3RUT.FOR	convert A1 format to A2
A2A1	PH3RUT.FOR	convert A2 format to A1
COPINT	PH3RUT.FOR	copy integer array
COPREL	PH3RUT.FOR	copy real array
INCINT	PH3RUT.FOR	add to all integer array elements
INITLZ	PH3.FOR	initialize program constants

Variable IOPSYS in /CONTRL/ common is now used to switch code that is operating system specific. Set IOPSYS=3 in INITLZ, and test on IOPSYS to bypass or insert statements needed only on the IBM PC version.

The "stand alone" format of most conversion testing will require that various dummy routines exist until they are replaced by true ones from new or downloaded code. For initial work create two:

ERROR(NUM)	Prints to unit 8 that error number NUM was detected and sets common /CONTRL/ variable IERFLG=NUM.
KNCERR(NUM,MARK)	Prints to unit 8 that error number NUM was detected at (CHARACTER *8) point MARK and divides by zero to kill the program.

In testing it is often necessary to 'print' material to a file for analysis. When terminal input/output FORTRAN units 0 are employed with main displays and graphics, write to unit 8. This will avoid conflict with other GITPASE operations.

FORTRAN STANDARDS

May 11, 1984

The following are a series of rules defining severe restrictions on the range of options usually open to a FORTRAN programmer. However, adherence to the rules leads to FORTRAN code that is relatively transparent to a reader, easy to modify and enhance, and readily converted between different machines and different FORTRAN compilers. Unless otherwise approved, all FORTRAN code should conform to these rules.

STATEMENT FORMATTING

--All FORTRAN statements should be entered in standard, fixed format. Specifically,

--The first line of any statement begins at a specified indentation level (see indentation below), but not to the left of column 7.

--Continuation lines have the '$' symbol in column 6 and substantive characters beginning 3 spaces to the right of the first line they continue.

--Comment lines have the 'C' character in column 1. Substance begins with '---' at the same column as the succeeding non-comment line.

--Continuation comment lines have the same format at the lines they continue.

--All FORTRAN lines end at or before column 72.

--No literal string, i.e. string enclosed in quotes, should extend beyond one FORTRAN line. If a long string is required, break it into two consecutive parts.

--If a statement has a number, the number begins in column 1.

--Stored code lines should include no tab or other line control characters.

--All alpha characters in FORTRAN statements will be upper case.

STATEMENT NUMBERING

--Only 'CONTINUE' and 'FORMAT' statements may carry numbers.

--All statement numbers of a subroutine, function, or main program will be either 3 or 4 digits. 3 is generally preferred.

--Within a main program, subroutine, or function, all statement numbers should be in ascending sequence.

FORBIDDEN STATEMENTS

--With the exceptions specifically noted, the following FORTRAN statement forms will not be used:

--Any type declaration statement (including 'INTEGER', 'REAL', 'DOUBLE PRECISION', 'LOGICAL', 'CHARACTER', 'BYTE', etc., except where 'CHARACTER' is explicitly authorized or integers must exceed 2 bytes.

--'IMPLICIT' except when needed for machine compatibility.

--'COMMON LIST', i.e. blank common

--'RETURN'

--'ENTRY'

--'EXTERNAL'

--'BLOCK DATA'

--'IF( ) N1,N2', i.e. logical if with 2 branches

--'IF ( ) THEN...ELSE'

--'ENCODE' except as required for machine compatibility

--'DECODE' except as required for machine compatibility

--'PRINT n'

--'READ n'

--'ACCEPT'

--'ASSIGN'

--'DO...UNTIL'

--'DO...WHILE'

--'WHILE'

--'END DO'

--'END WHILE'

--'WRITE (array', i.e. core write where required for compatibility

--'READ (array', i.e. core read except where required for compatibility

DIMENSIONING

--Generally speaking, all dimensioned variables should appear in labelled (block) common.

--'DIMENSION' statements may be used only for subscripted variables employed as

A. Passed parameters of functions or subroutines.
B. Objects of 'DATA' statements
C. Objects of 'EQUIVALENCE' statements.
D. Local work areas of less than 20-25 values.

--In all cases except B above, the rightmost dimension shown in a dimension statement will be 1.

--No variable should have more than 3 subscripts.

VARIABLE NAMING

--Every variable name shall be at least 3 characters long, and no more than 6 characters long.

---Variance names shall include only digits 0-9 and alphabetic characters A-Z, with the first character alphabetic.

--Within the above length restrictions and the dictates of FORTRAN default type conventions, names should meaningfully indicate their significance. A good rule is to shorten by dropping vowels. If the variable reflects a quantity in supporting mathematics, triple it. For example, Use 'XXX' for 'X'.

--Avoid meaningless names such as IDUM and ITEMP, and where such names are used be sure their meaning is not required to persist for more than 5-10 lines.

--All variable names held in common should be defined in the documentation file associated with your program.

--All variable names held in common should be unique throughout the program.

--String/character quantities should be processed in integer variables, i.e. names beginning with I-N, except where character type is explicitly allowed.

--Each word of an integer variable containing strings should have either 1 or 2 characters, i.e. be in either format 'A1' or format 'A2'.

SUBROUTINES AND FUNCTIONS

--String/character functions will not be used

--Subroutine and function names should conform to the same naming rules as variables.

The first statements following the 'SUBROUTINE' or 'FUNCTION' statement should be a series of comments briefly defining the purpose of the function or subroutine. Such comments should also define the meaning of any parameters in the subroutine or function.

--Every subroutine or function name should also be defined in the documentation file that accompanies the program.

--Every subroutine or function should have 1 and only 1 'RETURN'. That return should be preceded by the statement '9000 CONTINUE'.

--Generally, subroutines and functions should not contain more than 50 statements exclusive of commons and comments.

--CALL's to subroutines and functions will match in default variable type the parameters declared. Specifically, scalar integers in calls may not be integer constants or the results of calculations (both INTEGER *4). Instead assign a value to a variable and call with that variable. For routine constants 0,1,...,100 variable NUMBR0, NUMBR(1), NUMBR(2)...,NUMBR(100) are maintained in COMMON.

--Avoid as much as possible use of implicit functions (e.g. MIN, MAX, ABS that may be type-specific.

--Every subroutine or function should begin with
IMPLICIT ß INTEGER*2 (I-N).

COMMONS

--Blank COMMON will not be used.

--All variables with significance beyond a few adjacent lines of code should be declared in labelled (block) COMMON except those that are parameters of subroutines or functions.

--Variable names in common should be unique throughout the entire program.

--COMMON declarations should not appear explicitly in subroutines. Instead 'INCLUDE' statements should be provided to copy in a stored common declaration.

--Any particular COMMON block shall have 1 and only 1 form indicated by the copyable version mentioned in 'INCLUDE' statements.

--COMMON blocks should group related variable quantities, i.e. quantities likely to be simultaneously used by a portion of program logic.

--Main programs, functions and subroutines should declare only COMMON blocks they actually employ unless otherwise necessary for overlaying.

MODULARIZATION

--At all times an effort should be made to keep the code modular, i.e. execute logic in relatively short subroutines and functions, called as needed.

--Generally, no more than 10 lines should be duplicated at different points in a program. If lines would be duplicated, create and call a function or subroutine.

--Within main programs, subroutines, and functions, the possibility should be anticipated that groups of lines will be extracted later as a subroutine or function. Thus, for example, variable should be initialized in the immediate area where they are used rather than at the beginning of all logic. Similarly, 'FORMAT" statements should be placed adjacent to the 'READ' or 'WRITE' statements that reference them.

INDENTATION AND STRUCTURING

--To improve readability and to keep logic simple and transparent, statements will be indented and control transfers limited as indicated in the following:

--Statements having the property that if one is executed, the other must also be executed, and be indented to the same column.

--Statements which are only executed conditionally should begin 3 spaces to the right of 'DO,' 'IF(logical)GO TO N', 'IF(Arithmetic)N1,N2,N3', 'GO TO (N1,N2...,NN),IDX', 'GO TO N' or 'CONTINUE' that precedes it and specifies the conditions under which it is to be executed.

--In particular, all statements in the range of a 'DO' loop will be indented 3 spaces (or more if other statements intervene) than the 'DO' which begins the loop and the 'CONTINUE' that ends it. Such a 'DO' and a 'CONTINUE' will be indented at the same level because they encounter exactly the same logic cases.

--Similarly, loops implemented by 'GO TO' and 'IF' statements will be indented throughout the range of the loop. Generally, those intended to effect a 'WHILE' begin with a 'CONTINUE' and an 'IF' at the normal margin, with loop statements indented 3 columns further. Loops effecting an 'UNTIL' notion begin with a 'CONTINUE' on the normal margin, with loop statements following at a 3 space indentation.

--Likewise, a series of cases will be introduced by a collection of 'IF' and 'GO TO' statements transferring control to the appropriate case. All such control statements would be at the same level of

indentation because they constitute one logical operation. Statements for each case would be indented 3 spaces beyond the control statements.

--Each non-comment statement indented further than its immediate predecessor non-comment statement should be preceded by a comment statement detailing the case or conditions it represents. The comment should be indented to the same level as its successor.

--Statements transferring control such at 'GO TO N', 'IF(logical)GO TO N', 'IF(Arith)N1,N2,N3', 'GO TO (NS,N2,...,NN),IDX', etc. may reference only numbered 'CONTINUE' statements. Furthermore, such references are restricted to the following cases.

A. The referenced statement may be the next statement at the same level of indentation.

B. The referenced statement may be the next statement at any higher (more left) level of indentation.

C. The referenced statement may be the last preceding statement at any higher (more left) level of indentation (or a 'CONTINUE" immediately preceding that statement).

D. The statement reference may be a 'CONTINUE' at the next subsidiary (more right) level of indentation that introduces a case as explained above.

--In general, long loops and transfers of control past more than 10-20 statements should be avoided. Call a subroutine to perform the component steps of the loop or execute a case.

--Transfers of control to preceding statements should be employed only when necessary for looping. Normally the flow of logic should be top to bottom in a main program, subroutine or function.

--Although they may occasionally be useful, 'flags' or similar indicators should only rarely be used to control logic. 'GO TO' statements within the limits outlined above are far preferred to 'flags'.

MISCALLENEOUS EXCLUSIONS

--Generally, no FORTRAN construct not typically considered part of FORTRAN IV should be employed.

--Subscripts should not be 0, negative, or contain subscripted variables.

--'DO' loops should not employ a negative step, and all 'DO' parameters should be simple integer variables (i.e. not subscribed or expressions).

--Whenever possible, implicit functions should be avoided. In particular, do not use implicit functions for absolute value, type

conversion, logical operations, modulo arithmetic, max of a set, or min of a set. Instead, do explicit logic. For example, 'ABS(XXX)' can be simulated by 'ABX=XXX' followed by 'IF(XXX.LT.0.)ABX=-XXX'.

--Strings should be assigned to variables only through 'DATA' statements and 'READS,' i.e. constructs such as 'IVBL='String' are not allowed.

--Formats should always be given in 'FORMAT" statements, not within Read's and WRITE's.

MACHINE/COMPILER PORTABILITY

--One effect of the rules in these standards is to keep FORTRAN code as independent as possible of different computers and compilers. However, some activities are inherently machine specific. When they are used in a program references to them should be confined to 1 or 2 subroutines or functions. Then, only 1 or 2 places need to be modified for a conversion.

--Specific activities that should be modularized in this way include

A. File openings and closings

B. Routines manipulating strings, (e.g. combining character groups of form 2A1 into A2)

C. Random-access reads and writes

D. Free-field reads and writes

E. References to system clocks and dates

F. Special error handling routines

G. Overlay calls

DOCUMENTATION SECTION

--Rules above provide for comments throughout the program indicating functions of major logic block, subroutines and functions.

--In addition to such within program documentation, a separate documentation file should be maintained on each program. If a program is relatively short, that file may precede the beginning of the main program. If the program is long, it should be a separate file.

--All entries in the documentation file will be contained in columns 1-72, with a 'C' in column 1, i.e. they will be formatted as FORTRAN comments.

--A minimal list of entries in the documentation file is

A. Definitions of all variables held in common

B. Definitions of the functions of all 'SUBROUTINES' and 'FUNCTIONS' employed by the program.

C. Listings of job control language (and overlay language if appropriate) needed to compile, link and execute the program.

D. A description of the computer/compiler environment in which the program is designed to work.

E. An indication of the last date and time that the documentation file was modified.

--As appropriate, other technical information on program operation may be included in the documentation file.

Milestone B: Keyed Input

By Ronald L. Rardin

The Phase III version of the GITPASE project management system employs a hybrid of a VAX computer running FORTRAN and a Chromatics color graphics microprocessor running BASIC. Primary functions of the BASIC program are to produce various graphics and strings on command from FORTRAN, and to collect and edit user input. This Milestone 3 addresses keyed inputs processing.

B.1 Background and Interface

All GITPASE input is solicited from BASIC via subroutine COMIO. The type of input sought is indicated by a response code stored in /CONTRL/ common variable KOMRSP.

Table B1 shows the presently active response codes. Numbers 1 and 10 obtain light pen screen locations (x,y). Numbers 2-9 and 11-15 seek keyed strings of various formats.

Within FORTRAN touch inputs are processed through subroutine OUTPEN and string inputs through OUTKEY and OUTQA, all of which call COMIO. The basic strategy of IBM PC conversion will be to substitute new FORTRAN for these three 'OUT' routines. The new routines will be demonstrated in stand alone operation and linked to GITPASE in Milestone H.

B.1.1 Touch Queue

Touch input (OUTPEN) will be the subject of Milestone F. However, it is important to understand now that it consists of a queue of coordinate pairs entered without interspersed screen update. Common /CONTRL/ variable KUEON≠0 when this queue acceptance is underway and KUENUM shows how many pairs remain in the queue. When the queue is empty subroutine CACHUP(NUMBR(2)) is called to restore the screen before further processing. A dummy routine CACHUP(IDIR) is required for this Milestone B that merely sets KUEON=0.

B.1.2 Windows

The GITPASE code describes screen locations in terms of windows. Table B2 shows numbers used. Window 0 is a special 'no conversion' one corresponding to the physical screen. All others have internal logical dimensions that are mapped to external ones by routines to be developed in Milestone C. Routines of the present Milestone that employ coordinates must always also indicate a window.

TABLE B1. PRESENT GITPASE RESPONSE CODES

CODE	MEANING
1	Obtain the next light pen touch from the touch queue, or if touch queue is empty, obtain a new queue of touches.
2	Obtain a (CHARACTER*20) file name from the user (appending .DAT if the returned name has no '.')
3	Obtain a 13 character resource code-name string (inserting 'ß' after the 3rd character to total 14 characters). (Format 7A2)
4	Obtain a 12 character activity name string (Format 6A2)
5	Obtain an alphanumeric string of arbitrary length (Format A2)
6	Obtain up to 10 resource availability segment inputs, each one integer followed by one real (setting KOMBTN = how many pairs)
7	Obtain up to 3 consumption values, all real
8	Obtain a single integer input
9	Obtain up to 4 integer inputs
10	Obtain a confirming light pen touch (and cancel the touch queue)
11	Obtain a single real input
12	Obtain a time period code in Format A2, A1, A2
13	Obtain up to 10 resource code-weight pair inputs, each an alphanumeric code of up to 3 characters (Format A2, A1) and a real (setting KOMBTN = how many pairs)
14	Obtain a precdence offset code of two characters (Format 2A1), both 0-9 or ':'.
15	Obtain any message ended by carriage return

TABLE B2. GITPASE LOGICAL WINDOWS

0 = The Physical Screen

69 = Lower Window

79 = Title Window

89 = Message Window

91 = Transition Mode Window 1 (top)
92 = Transition Mode Window 2 (middle)
93 = Transition Mode Window 3 (bottom)

98 = Above Main Window
99 = Main Window

B.1.3 Highlighting

Most GITPASE input protocols use blinking to cue the user about the portion of data to which his or her next action will apply. For example, if a changed activity name is anticipated, the program will blink the present name and position the cursor to echo typed modifications over the existing value.

IBM PC's HALO does not directly implement blink. Thus, an effective substitute will be required.

Present GITPASE logic implements highlighting via rectangles,i.e., rectangles for which the blink plane is activated and all included data will blink. Conversion will be easiest if the selected highlighting scheme retains this rectangle notion.

Until suitable coordinate systems have been established, create dummy routines.

HLIT(IWIN,NXI,NYI,NXZ,NYZ) that will highlight a rectangle and DEHLIT(IWIN,NXI,NYI,NX2,NY2) to dehighlight one.

B.1.4 Help

All GITPASE input allows users to call on dynamic help routines for guidance. For the present Milestone a dummy routine HELP is required to substitute for the collection of help routines. Also /CONTRL/ common variables KMRSP and KOMQUS are set for HELP reference.

B.2 OUTKEY Processing

Subroutine OUTKEY(KEYRSP,KOLOR,KORDX1,KORDY,KORDX2,NWIN) is the main GITPASE mechanism for soliciting keyboard input. It seeks a response of response code KEYRSP. The use is signaled as to the information sought by highlighting rectangle (KORDX1,KORDY) through (KORDX2,KORDY-KARY+1) of wi ndow NWIN where KARY = the number of dots high in a string character. Input echo should be in color KOLOR, at coordinates (KORDX1,KORDY).

Normally OUTKEY merely solicits an input, decodes it, stores results in relevant variables and exits. However, certain conditions require special processing.

(i) If KUEON ≠ 0, i.e., touch queue processing has been underway. OUTKEY sets KUENUM = 0 and calls CACHUP (NUMBR(2)) to bring the screen to correct state (NOTE: Dummy that routine for testing).

(ii) If the keyed response is an empty carriage return, set KOMBTN to 49 and return.

(iii) If the keyed response is ',', set KOMBTN = 48 and return.

(iv) If the keyed response is 'HELP', subroutine HELP is called to write help messages, and processing loops back to solicit another keyed input (NOTE: Dummy HELP for testing).

B.2.1 Keyed Input Variables

To support such processing add to /IBMPC/new variables

KEYIN(80) = characters returned from READ (one per word format A1)

LENKEY = length of KEYIN returned from READ

KEYPCL(21,80) = characters of parcels 1 through 21

LENPCL(21) = length of parcels 1 through 21

B.2.2 Parcing Subroutine PARC

Input from keyboard READ's will be stored with format A1 in vector KEYIN. The first required processing is to subdivide the message into string parcels separated by at least one blank.

New subroutine PARC(MAXPCL,NUMPCL) will perform this role. It creates up to (input variable) MAXPCL parcels and stores results in array KEYPCL. The number created is output variable NUMPCL. Processing begins by striping leading and trailing blanks of KEYIN length LENKEY. Then parcels 1 through (MAXPCL-1) or end of KEYIN are created as "characters up to the next blank separator." LENPCL records each parcel length. If characters remain after parcel (MAXPCL-1), all remaining go to the parcel MAXPCL.

3.2.3 Edit Subroutines

Two special edit routines will convert single parcels into suitable nonstring forms.

Subroutine EDREAL(NPCL,VALU,IERROR) creates a real number VALU from characters in parcel NPCL using DECREL. DECREL error sets IERROR to 10. If VALU < EPS set IERROR = 9. If no error is detected, set IERROR to 0.

3.2.4 Replacement Subroutine OUTKEY

Write a replacement subroutine OUTKEY to process keyed input as outlined above, using these new service routines. The logic flow is as follows:

(a) If KUEON ≠ catchup from a touch queue by setting KUENUM to 0 and calling CACHUP(NUMBR(2)).
(b) Set KOMRSP and KMRSP to KEYRSP and KOMBTN to 0.
(c) Unless KORDX1=KORDX2, call KLIT to highlight rectangle (KORX1,KORY) through (KORX2,KORY-KARY+).

(d) Read from the terminal (unit 0) as KEYIN a format A1 character string. The string should be echoed as it is typed. In later Milestones echo will be graphically controlled, but now simply do next alpha screen line.

(e) If LENKEY = 0, set KOMBTN to 49, and go to (k).

(f) If LENKEY = 1 and KEYIN(1 = ',' set KOMBTN to 48 and go to (i).

(g) If LENKEY = 4 and KEYIN(1 to 4) = 'HELP' call HELP and return to (d).

(h) According to KOMRSP, process KEYIN as indicated in Table B3 to load /CONTRL/ values for GITPASE processing.

(i) If any error was detected at (h), call ERROR(the number), clear IERFLG to 0, and return to (d).

(j) Return.

B.3 OUTQA Processing

Subroutine OUTQA(IQUES,IANSW,KOLOR,KORX,KORY,NWIN) has a similar function to OUTKEY. It solicits a keyed response of type IANSW. However, it first poses a question to the user and then calls OUTKEY to obtain an answer.

Create a substitute OUTQA with processing as follows:

(a) If KUEON $\neq$ 0, catchup form a touch queue by setting KUENUM to 0 and calling CACHUP (NUMBR(2)).

(b) Set KOMQUS = IQUES

(c) Activate the same display area where typed input is echoed and write question number IQUES (see Table B4) from IQUESN, length IQUELN (for now as in OUTKEY step (d)).

(d) Call OUTKEY(IANSW,KOLOR,KORX,KORY,KORX,NWIN).

(e) Return.

TABLE B3. Keyed Input Processing by Response Code

Response Code	MAXPCL to PARC	Parcel Processing	Errors Other Than in Edit Routines
2	1	Core WRITE parcel to KOMFIL, appending (CHARACTER*20) variable .DAT if the parcel has no '.'	None
3	1	A1A2 parcel (1) to KOMIN, two characters per word, inserting ƀ as character 3.	#16 if LENPCL(1) > 13
4	1	A1A2 parcel(1) to KOMIN, two characters per word.	#16 if LENPCL(1) > 12
5	1	A1A2 parcel(1) to KOMIN, two characters per word	None
6	21	Use EDINT and EDREAL on alternate parcels, storing values in KOMIN(pair) and COMIN(pair). Set KOMBTN to the number of pairs found, i.e., (NUMPCL/2).	#16 if NUMPCL > 20 or an odd number
7	4	Use EDREAL to place up to 3 parcels as real values in COMIN(1 to 3)	#16 if NUMPCL > 3
8	2	Use EDINT to convert parcel(1) to KOMIN(1).	#16 if NUMPCL > 1
9	5	Use EDINT to convert up to 4 parcels into KOMIN(1 to 4).	#16 if NUMPCL > 4
11	2	Use EDREAL to convert parcel(1) into COMIN(1).	#16 if NUMPCL > 1
12	2	A1A2 parcel(1) to KOMIN(1 to 3) in Format A2,A1,A2.	#16 if NUMPCL > 1

TABLE B3. Continued

Response Code	MAXPCL to PARC	Parcel Processing	Errors Other Than in Edit Routines
13	21	Check odd numbered parcels for length $\leq$ 3 and A2A1 2 bytes per word in KOMIN. Use EDREAL to convert even parcels to COMIN(pair) values. Set KOMBTN to number of pairs (NUMPCL/2).	#16 if NUMPCL > 20 or an odd number #16 if any odd parcel not $\leq$ 3 characters
14	2	Check that LENPCL(1) = 2, and both characters are '0', '1', '2',..., '9' or ':'. Copy to KOMIN(1 to 2), 1 character per word.	#16 if NUMPCL > 1 or LENPCL(1) $\neq$ 2 #15 if illegal characters included
15	Not Needed	No Conversion	None

*MAXPCL is typically set one greater than the expected max number of inputs. Then, if NUMPCL equals MAXPCL, "too much data" is detected.

TABLE B4. QUESTIONS OF OUTQA

Number	String *
1	'Save as which plan (1=min,2=nom,3=max,4,5<cr>=nosave?'
2	'Load which plan (1=min,2=nom,3=max,4,5<cr>=noload?'
3	'Save current files before next ß (0=no,1=yes)?'
4	'Convert data to new period units (0=no,1=yes)?'
5	'How many new per old (1=no conversion,2,...)?'
6	'Next file name?'
7	'Reposition ß how ß many ß (+-n) ß or ß to ß which ß (#n ß or ß *)?'

*These string should be initialized by OUTQA DATA statements into array IQUESN(40,7), two characters per word. Set string lengths in IQUELN(7).

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

Atlanta, Georgia 30332-0205 (404) 894-2300

May 22, 1984

M E M O R A N D U M

TO: Cpt. Larry Frank, AIRMICS

FROM: Donovan Young

SUBJECT: Hardware and Purchased Software for E-26-669
Stand-Alone APMS

Georgia Tech respectfully requests permission to procure necessary hardware and purchased software as detailed in the attached memorandum.

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

Atlanta, Georgia 30332-0205

(404) 894-2300

May 15, 1984

M E M O R A N D U M

TO: Donovan Young, Project Director

FROM: Ronald L. Rardin, Consultant

SUBJECT: Firmware for GITPASE IBM PC Version

In accordance with my contract CI-E-24-669, I have studied alternatives for IBM PC firmware capable of supporting GITPASE. On February 4, 1984 the results of that study were presented to you and to AIRMICS. This memo documents the analysis presented and conclusions reached at that meeting and since.

1. Requirements

The IBM PC version is to be implemented on four different machines. AIRMICS owns a PC that is to be upgraded. DSCM owns another. I have a Purdue-owned XT that is also to be upgraded. Finally, a new XT is to be purchased by Georgia Tech.

In all four cases several distinct issues must be considered. The following reviews each in turn.

1.1 FORTRAN

It is essential that a powerful FORTRAN compiler be available on all four machines. The 'standard' FORTRAN for PC's is from Microsoft. Purdue has available Microsoft FORTRAN 3.13 running under DOS 2.0 with and without the 8087 coprocessor chip. My tests of that compiler on a heavy operations research algorithm showed the compiler to have adequate flexibility. A program which ran in 5 time units on a busy VAX took 42 on the XT without the coprocessor, and 9 units with the coprocessor.

On the basis of this test, I conclude that all four units should be upgraded to have DOS 2.0 (or later) and FORTRAN 3.13 (or later) and 8087 coprocessors. This will require three copies of the FORTRAN compiler, three 8087 coprocessors and three copies of DOS 2.0 (none for Purdue).

1.2 Main Memory

IBM PC's come with 64K memory and XT's with 256K. At present the AIRMICS and DSMC PC's have a QUAD board to expand to 312K and 256K respectively. The current FORTRAN part of GITPASE requires 200-300K bytes. Since new code must be added to replace BASIC, I recommend all machines be upgraded to 512K. This will require purchasing AST 6 PAK or equivalent boards for the memory chips. Four boards and 11 64K chips are required.

1.3 Locator

GITPASE is locator driven, only the AIRMICS machine presently has a locator--in that case a light pen. Inquiries indicate light pens are generally inaccurate on PC's; most only map to character boundaries. Also, light pens are physically tiring and require sophisticated software to implement touch queues.

An alternative is a mouse. Two are available. Microsoft has one with resolution 640 x 200; Mouse Systems another. However, the Mouse Systems unit has long delivery time and operates on a simulated hit pad. For these reasons, I recommend four Microsoft mouses be purchased.

1.4 Printers

The two PC's presently have printers, but ones will be needed for the XT's. Major suppliers are Epson and Ohidata. Since Ohidata's are twice as fast for a comparable cost, I recommend them. Two model 93A (130 column) units should be acquired for the XT's.

1.5 Color Graphics

A number of "color boards" are available that produce "high" resolution graphics on IBM PC's. Most interact with IBM's digital color monitor and are thus restricted to 640 by 400 resolution in 8 colors (at two intensities). Suitable implementation of GITPASE requires at least twelve colors and more resolution. The lowest cost well-known alternative is provided by SCION (the PC 640). The board is driven by HALO subroutine calls, and requires an analog color monitor, and supports 16 colors and 640 X 480 resolution. Monitors may be purchased from several suppliers at $1500 for 13" and $3-5000 for 19". I recommend all four units be equipped with the SCION board, FORTRAN - callable HALO, and Electrohome 1301 (13") monitors.

1.6 Disks and Chassis

All units must have at least on floppy disk for transfer and random access features of GITPASE absolutely require all units be equipped with hard disks. (XT's automatically have hard disks). In addition chassis must have enough stats to support boards for other features reviewed above. PC's have five expansion slots and XT's have 6 long and 2 short expansion slots. Total required is as follows:

	PC's	XT's Long	XT's Short
Floppy Disk	1	1	-
Hard Disk	1	1	-
SCION Board	1	1	-
Mouse	1	-	1
Extra Memory	2	1	-
IBM Monitor Board	1	1	-
Total	7	5	1

From this tabulation it is clear expansion chassis will be required for the two PC's. I recommend acquiring IBM expansion chassis's with 10MB hard disks.

2. Estimates

Summarizing the above, Table I provides estimated cost of the required firmware. The project budget includes $8,600 for overhead-bearing procurement and $28,600 of non-overhead-bearing. Values in Table I total to $8,320 overhead-bearing and $28,600 nonoverhead-bearing. Thus, both values are within budget.

3. Procurement

On the basis of our February meeting and the urgent need to obtain experience with graphics features, procurement was started in February on the underlined items in Table I (2-SCION boards, 2-mouses, 2-13" monitors, 1-FORTRAN compiler). At this date this equipment has not yet been successfully mated and tested. For this reason, I recommend proceeding with all procurement except remaining SCION boards, monitors and mouses. The latter should be held until ones already received are operating suitably.

TABLE I. SUMMARY OF ESTIMATED COSTS

	AIRMICS PC	DSMC PC	PURDUE XT	NEW XT
Base unit	-	-	-	6000
DOS; FORTRAN 3.13	420*	420*	-	420
8087	160*	160*	160*	160*
Printer	-	-	750	750
512K RAM & Board	500*	550*	550*	550
SCION Board & HALO	1600*	1600*	1600*	1600
Mouse	200*	200*	200*	200
Hard disk (in expansion chassis)	3400	3400		
13" Analog Monitor	1500	1500	1500	1500
With overhead	2880	2930	2510	0
Without overhead	4900	4900	2250	11,180

*Indicates overhead bearing
Underline indicates February procurement

SCHOOL OF INDUSTRIAL AND SYSTEMS ENGINEERING

Atlanta, Georgia 30332-0205

(404) 894-2300

May 30, 1984

M E M O R A N D U M

TO: Cpt. Larry Frank

FROM: Donovan Young

SUBJECT: Stand-Alone APMS, E-24-669

1. Ron Rardin has tested the mouse, and it works. Therefore, we request permission to procure the mouses now, in addition to the procurements listed in our memo of May 31, 1984 (E-24-669 Software Development Plan). We understand that AIRMICS has a mouse, so the number of mouses will be reduced by one.

2. Material is on hand for upgrading the AIRMICS IBM-PC: the Scion board (with HALO diskette), the monitor (with interface box and cable), Fortran 3.2, and the mouse. If you would like to experiment now with these materials, they are available in my office. Alternatively, you can await Ron's configuration instructions.

3. Ron is sending a GITPASE tape today, which we hope will be the delivered VAX version for the previous contract (APMS Phase III, E-24-658). It will be suitable for demos and for downloading of code (see Milestone B).

4. Clarification of Milestone B. Milestone B specifications issued last week are not explicit as to how you should test the results of your work. The intent is that you should write a dummy interactive program that will ask which response is wanted, accept a response code from the terminal, then make a call to OUTKEY or OUTQA, accept typed input from the terminal, and display either the appropriate error message or what was received internally. Then this program should be exercised to verify proper interpretation (including error detection) of all the strings that are supposed to give outputs.

   Paragraphs 3.2.3 and 3.2.4 should be numbered B.2.3 and B.2.4.

   In paragraph B.2.3 the error condition should read VALU < 0 (not EPS).

Add the following text to paragraph B.2.4:

Subroutine EDINT(NPCL,IVALU,IERROR) creates an integer IVALU from characters in parcel NPCL using DECINT. DECINT errors sets IERROR to 10. If IVALU < 0 set IERROR = 9. If no error is detected, set IERROR to 0.

5. Ron Rardin reports that there is a PC HALO manual that gives much more detail about HALO functions than does the Scion manual. He is using the PC HALO manual in the design work and suggests you might find it useful also. If you cannot readily obtain one for reference, please contact me.

INTERIM REPORT

Stand-Alone-Operation APMS

D.O. #0017 under DAAK70-79-D-0087

E-24-669

Project Director: Donovan Young

Date: 29 November 1983

(2) Capt. Larry Frank, AIRMICS
(2) Pat Heitmuller, PPC
(1) Michael E. Thomas, ISYE
(1) Ronald L. Rardin, Purdue Univ.
(3) File

1. As required by paragraph 5.1 of the Statement of Work, the enclosed Task Schedule is submitted. Highlights: PC hardware specifications by 13 Jan 84; stand-alone software specifications to be issued in four "milestones" 1 Mar, 1 Apr, 1 May and 1 June; system to be tested (by ISyE) upon completion of programming (by Government) of each "milestone," with a final system test to cover both the stand-alone and portable software; portable hardware specifications to be issued by 1 Sep. Project completion 30 Sep 84.

2. Ronald L. Rardin has accepted the offered Consulting Agreement No. C1-E-24-669. The Statement of Work for this agreement is enclosed herewith.

3. By copy of this Interim Report, Georgia Tech OCA is requested to draft and send two documents to Ronald L. Rardin:

A. Whereas it is the established custom of the Government to pass title for certain Government-furnished equipment and supplies used in research to Georgia Tech at the end of research projects; and whereas part of the Government-furnished equipment and supplies for this project is to upgrade a non-Government-owned IBM PC system available to the consultant in West Lafayette, Indiana; Georgia Tech agrees to relinquish ownership of this part only of equipment and supplies, if released by the Government, to the consultant or his assignee.

B. Whereas Article XIV of the consulting agreement appears to require Georgia Tech's written consent for the consultant to hire a sub-consultant at his own expense, and whereas the consultant expects to hire a sub-consultant for several days to help him develop hardware specifications, such consent (if indeed required) is hereby granted.

4. The Government-furnished Chromatics system with the following serial numbers has been transferred to Ronald L. Rardin for use in West Lafayette, Indiana, during the contract:

Diskette drives:	013072 and 012072
Terminal:	018072
Keyboard:	016072

The other Government-furnished Chromatics system, which is temporarily in the Washington, D.C. area to support installation of the IBM version of APMS under contract E-24-658, will remain at Georgia Tech ISyE upon its return.

5. To facilitate long-distance communication between Georgia Tech and its consultant Ronald L. Rardin, a long-distance telephone credit card has been applied for.

DY:sr
Attachment

STATEMENT OF WORK

Fixed-Price Consulting Agreement - Ronald L. Rardin, Consultant to Georgia Tech for E-24-669.

Please refer to the Statement of Work for contract E-24-669. In the parent contract, Georgia Tech is to analyze and procure hardware and software necessary for stand-alone operation of an Automated Project Management System (APMS) on personal and portable computer systems, to produce specifications from which the Government will perform the necessary programming, to perform validation testing on the Government-prepared programs, and to provide user documentation. The Consultant will:

1. Perform an analysis of the hardware and software necessary for stand-alone operation of APMS on IBM PC equipment and on portable equipment, and submit the results in draft form to Georgia Tech to aid in performance of para. 3.2.a and 3.4.a of the parent contract's Statement of Work (SOW).

2. Study the available alternatives to light-pen input for APMS and make recommendations to aid Georgia Tech in performance of para. 3.2.b of the parent SOW.

3. Make specific recommendations on procurement of equipment and materials to upgrade existing IBM PC systems to be used in design and test work, to aid Georgia Tech in planning for its procurement task, para. 3.3 of the parent SOW.

4. Prepare a functional specification in draft form for a limited-capability version of APMS for portable equipment, to aid Georgia Tech in performance of para. 3.4.b.

5. Prepare formal design specifications in draft form for IBM PC and portable APMS versions, to aid Georgia Tech in performance of para. 3.5 of the parent SOW.

6. Prepare a functional specification for desirable enhancements to APMS, in draft form, to aid Georgia Tech in performance of para. 3.9 of the parent SOW.

The work is to be performed mainly in West Lafayette, Indiana. Several trips to the Washington D.C. area and the Georgia Tech campus are anticipated; travel expenses will be reimbursed out of the parent contract. Long distance telephone expenses will be reimbursed out of the parent contract. The consultant will locally obtain access to an IBM PC system, and will utilize Government-furnished equipment and materials that will upgrade the system for use in doing the work.

TASK SCHEDULE

E-24-669

Stand-Alone-Operation APMS

Project Director: Donovan Young

	Georgia Tech (and Consultant) Tasks	Start Date	Finish Date
1.	Draft PC hardware recommendations	1 Dec 83	13 Jan 84
2.	Study input devices	1 Jan 84	13 Jan 84
3.	Revise PC hardware recommendations	23 Jan 84	25 Jan 84
4.	Procure PC hardware	30 Jan 84	24 Feb 84
5.	Prepare MS1 specifications	1 Jan 84	1 Mar 84
6.	Prepare MS2 specifications	1 Feb 84	1 Apr 84
7.	Test MS1 code	15 Apr 84	27 Apr 84
8.	Prepare MS3 specifications	1 Mar 84	1 May 84
9.	Prepare MS4 specifications	1 Apr 84	1 Jun 84
10.	Prepare portable hardware recommendations	15 Feb 84	1 Jun 84
11.	Test MS2 code	1 Jun 84	15 Jun 84
12.	Test MS3 code	20 Jul 84	27 Jul 84
13.	Procure portable hardware	8 Jun 84	3 Aug 84
14.	Prepare portable software specifications	1 Jul 84	17 Aug 84
15.	Test MS4 code	17 Aug 84	24 Aug 84
16.	Test portable software code	14 Sep 84	21 Sep 84
17.	Test systems	21 Sep 84	28 Sep 84
18.	Prepare user manual	1 Jun 84	28 Sep 84
19.	Prepare functional specifications of desirable enhancements	1 Jan 84	28 Sep 84

Task Schedule (continued)

	Government Tasks	Possible Start Date	Assumed Finish Date
1.	Review draft of PC hardware recommendations	13 Jan 84	20 Jan 84
2.	Approve PC hardware recommendations	25 Jan 84	27 Jan 84
3.	Write and verify MS1 code	1 Mar 84	15 Apr 84
4.	Write and verify MS2 code	1 Apr 84	1 Jun 84
5.	Approve portable hardware recommendations	1 Jun 84	8 Jun 84
6.	Write and verify MS3 code	1 May 84	20 Jul 84
7.	Write and verify MS4 code	1 Jun 84	17 Aug 84
8.	Write and verify portable software code	17 Aug 84	14 Sep 84

INTERIM REPORT

Stand-Alone-Operation APMS

D.O. #0017 under DAAK70-79-D-0087

E-24-669

Project Director: Donovan Young

Date: March 28, 1984

(2) Capt. Larry Frank, AIRMICS
(2) Pat Heitmuller, PPC
(1) Michael E. Thomas, ISYE
(1) Ronald L. Rardin, Purdue Univ.
(3) File

Equipment Procurement

Two copies of the Fortran-compatible Microsoft Mouse and two Scion graphics boards have been received; one of each is at Purdue and the other in Donovan Young's office. The Scion boards were delivered with a HALO diskette of the wrong type (BASIC, whereas FORTRAN was ordered); Scion will replace the diskettes (but see below). A copy of the Microsoft FORTRAN Compiler was recieved; it is version 3.20, which is later (and superior to) the 3.13 version ordered; a copy of the 4.0 version is on order, since that version is expected to be released within weeks. The two color monitors ordered have arrived, and one will be sent to Ron Rardin this week.

Ron Rardin expects to complete recommendations, for all equipment not affected by graphics-board selection, by the end of the week, and Don Young will initiate procurement next week. This equipment includes hard disks and expansion chassis for existing IBM PC's, as well as the new XT system.

Memory Difficulty and Graphics-Board Selection

Because of unclear capabilities and limitations of prospective high-resolution graphics systems, it was decided in February to order only two sets of monitors, mice and graphic boards, so that we could experiment to make sure the choices were suitable before making all the quipment purchases. Although it was not possible to tell before actually receiving it, the Scion board may be unsuitable.

The manual says that only 320K of user memory can be available after inserting the 256K Scion board. This user memory must hold several things:

1. The equivalent of the VAX executable code (now about 200K);
2. The HALO graphics software (probably small);
3. The net addition from converting the present Chromatic BASIC to FORTRAN and removing the communications code from the FORTRAN.

To make an estimate of suitability, Ron Rardin will research several questions:

1. Can we get 384K rather than 320K on the XT (contrary to the Scion manual but said to be likely)?

2. How big will be the equivalent of the present VAX FORTRAN code? If the Microsoft FORTRAN compiler and linker are less or more size-conserving than the VAX equivalents, this can be more or less than the 200K now being used. An estimate can be made on the basis of small test programs.

3. How much space will HALO occupy? This can be answered quickly once the correct HALO diskettes are received.

4. How much additional space will be required for converting the graphics logic now written in BASIC? The Pritskers version of GITPASE may give an accurate estimate of this.

If it turns out that the Scion board plus the program are simply too big for the IBM-XT, an alternative is to return the Scion boards and purchase a more expensive type of board that has its own logic chip and hence does not take up addresiable memory. Another alternative is to go to a smaller resolution. Neither alternative is attractive, and delay in issuing specifications would result if either of them are necessary. This question will be cleared up within two weeks.

Milestone C (and D): Graphics and Text Primitives

By Ronald L. Rardin

The Phase III version of the GITPASE project management system employs a hybrid of a VAX computer running FORTRAN and a Chromatics color graphics microprocessor running BASIC. This specification details new and substitute routines to replace graphics and string output primitive operations of BASIC with new FORTRAN for an IBMPC. It combines material originally planned as Milestone C and D. All routines should be kept alphabetically in 1-3 files: 'PC.FOR'

C.1 Environment

The environment assumed is an IBMPC-XT (or equivalent) running Microsoft FORTRAN, using the HALO software package to create high resolution graphics with the SCION PC640 board, and the Microsoft mouse software and board for locator input. An example program of this environment, with common declarations and compile/link commands is Attachment C1. A floppy disk of that program is also provided.

The indicated link command and the floppy disk also reference software called 'sbeep.' That package contains proprietary assembly routines loaned from Purdue Univesity that will be replaced in Milestone H. Calls to it have INTEGER*2 operands.

C.2 Screen Scaling and Initialization

C.2.1 Background

GITPASE screen scaling employs a variety of "windows". The Main Window contains most important information (see Figure C1). It is subdivided into three Transition Mode Windows during Transition Mode processing. The Title Window presents activity and resource names to label Main Window information. Thus, its vertical scaling corresponds to the Main Window. A Lower Window is used for a variety of supplemental information. The Message Window summarizes system state and reports errors.

Internally all windows are scaled over a 0-511 (x) by 0-8191 (y) dot logical rectangle. However, at any moment only a portion of that rectangle is visible on the physical screen.

The conversion from this visible rectangle to the physical screen is handled by a series of variables stored in COMMON/CONTRL/.

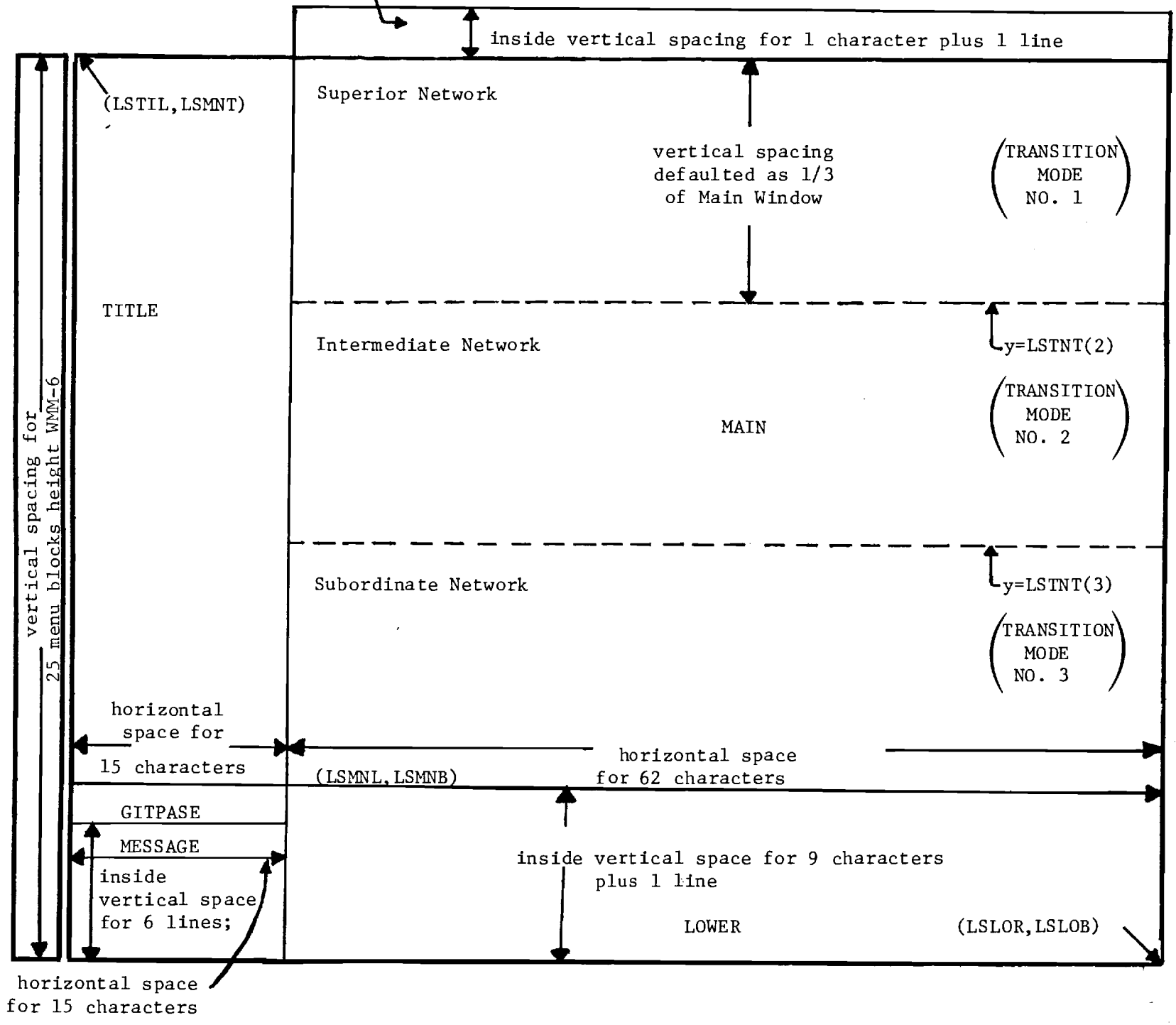


Figure C1. Coordinates for GITPASE Windows

Generally, the variable names are in the form LSwws or LNwws, where ww=MN, TI, LO, MS, or TN, depending on whether the window is Main, Title, Lower, Message, or Transition, and s=L, R, T, or B, depending on whether the variable refers to the line at left, right, top, or bottom. LSwws variables are permanent physical screen limits set in subroutine INITLZ. LNwws variables show temporary points in 511x8191 logical space which are to be mapped to corresponding LS coordinates.

Vertically, all screens except Message are "wrapped". That is, item 1 is logically treated as if it falls immediately after the last item. As screens roll, we see, for example, Activity 1, then Activity 2, then... until the the last Activity n, then Activity 1 recurs.

Wrapping is controlled by variables of the form LNwwW, where ww designates window. LNwwW is the line in 8191 space below which information repeats from the top (e.g., Activity 1). In other words LNwwW is logically equivalent to 8191.

GITPASE windows are known by identifying numbers:

Main = 99

Lower = 69

Title = 79

Message = 89

1st Transition = 91

2nd Transition = 92

3rd Transition = 93

GITPASE modes are selected by /CONTRL/ common variable KURMOD.

=0 for Transition

=1 for Activity

=2 for Resource

=3 for Schedule

=4 for Status

=5 for Select

=6 for Calendar

C.2.2 Initialization

Subroutine INITLZ now initializes GITPASE (a version was downloaded in Milestone A). Repeat the download of INITLZ and /CONTRL/ common to obtain the latest version. (Comment out lines to read the HELP directory). Then at the indicated spot call PCINIT to initialize for the PC version.

New subroutine PCINIT will perform all special PC-version initialization. It will closely parallel subroutine INITL of the demonstration program attached. Specifically it will initialize HALO for the SCION board, set character sizes, initialize the mouse, establish a color palette, and reset LS screen boundaries. Load the color palette as in the first part of demonstration routine COLORS. Set LS coordinates exactly as in INITL.

(Note: The effect of these settings is that characters 'bleed' into right window boundaries but scaling leaves no alternative. Boundaries will be refreshed in Milestone H.)

C.2.3 Coordinate Conversion Subroutine PCVTXY

New subroutine

PCVTXY(NWIN,INXY,NUMXY,I4XOUT,I4YOUT) will convert the NUMXY entries in I*2 vector INXY of internal x,y,x,... coordinates for window NWIN to I*4 output vectors I4XOUT and I4YOUT (HALO requires INTEGER*4). Values of INXY are transformed alternately as x-values of I4XOUT and y-values of I4YOUT. The first is x.

If NWIN=0, coordinates are already in external, LS coordinate form. Go directly to the second step. Otherwise, the first step in processing is to set local variables LSB, LST, LSL, LNB, LNT, LNL, and LNW to suitable choices for window NWIN. Generally, these are the corresponding LSwwB, LSwwT, etc. However, there are exceptions:

(i) There are no LSTIT, LSTIB, LNTIT or LNTIB values. Use corresponding variables for the Main Window.

(ii) If NWIN=79 (title) or 99 (main) and KURMOD=2 (resource), LSB=LSMNB+KARY+1.

The second step of processing is to separate INXY values into the output vectors I4XOUT and I4YOUT, converting as required. Conversion for an X-value is as follows:

$$\text{I4XOUT}(j) = \begin{cases} \text{INXY}(i) & \text{if NWIN=0} \\ \text{INXY}(i) + (\text{LSL}-\text{LNL}) & \text{otherwise} \end{cases}$$

For y-values it is

$$I4YOUT(j) = MXTMLY - \begin{cases} INXY(i) & \text{if NWIN=0} \\ INXY(i)+(LSB-LNB) & \text{if NWIN}\neq\text{0 and INXY(i)}>\text{LNTP} \\ INXY(i)+(LSTP-LSTP) & \text{otherwise} \end{cases}$$

where

$$LNTP = \begin{cases} 8191 & \text{if NWIN=69, LOHAS=2 and INXY(i)} \geq 8191 - KARY \\ LNT & \text{otherwise} \end{cases}$$

$$LSTP = \begin{cases} LST-(KARY+1) & \text{if NWIN=69 LOHAS=2 and INXY(i)}<8191-KARY \\ LST & \text{otherwise} \end{cases}$$

C.2.4 Termination

New subroutine PCTERM will end HALO processing at program end. It should merely call CLOSEG and SMODE as at the end of the demonstration program and RETURN. Call PCTERM at the end of all test programs.

C.3 Primitives Overview

C.3.1 Background

GITPASE output operations are called out by FORTRAN and executed by BASIC routines. All or nearly all messages invoking BASIC routines are sent by a series of "OUTGR" subroutines. OUTGR1, OUTGR2, OUTGR3, OUTGR4, and OUTGR send 1, 2, 3, 4, and multiple coordinate graphics respectively. OUTGRS, OUTGRC and OUTGRD send strings and parameters for BASIC routines.

The functions desired of any OUTGR call are controlled by an "op code" which is the first parameter of the call to each such routine. Table C-1 gives a composite list. A window parameter tells how coordinates should be transformed for external format (see C.2), and a color parameter determines the color number of the output.

Op Code	Function	Called Through	PC Implementation
1 & 2	Filled rectangle	OUTGR4	direct
3 & 4	Filled continuation rectangle	OUTGR2	setup OUTGR4 call with (LASXX,LASYY) as first coordinate
5	Unfilled rectangle	OUTGR4	direct
6	Unfilled continuation rectangle	OUTGR2	setup OUTGR4 call with (LASXX,LASYY) as first coordinate
7	Dashed x-bar	OUTGR3	direct (three coordinates are x_1,y,x_2). Setup OUTGR4 from (x_1,y) to (x_2,y))
8	Deblink rectangle	OUTGR4	log to file 8 and return
9	Blink rectangle	OUTGR4	log to file 8 and return
10	Change to red rectangle	OUTGR4	log to file 8 and return
11, 12 13, 14	Solid, concatenated vectors	OUTGR	direct (comes in (x,y) pairs)
15, 16 17, 18	Dashed, concatenated vectors	OUTGR	direct (comes in (x,y) pairs)
19	Correct composite color rectangle	OUTGR4	log to file 8 and return
20	Write asterisk	OUTGR2	call SYMBOL(-1,KOLOR,IWIN,KOORD1,KOORD2) (see Section C.4)
21	Write hierarchical tag	OUTGR2	call SYMBOL(Ø,KOLOR,IWIN,KOORD1,KOORD2) (see Section C.4)
22	String with last 7 characters reverse lettered	OUTGRS	direct

Op Code	Function	Called Through	PC Implementation
24	String requiring skip of character 4	OUTGRS	direct
23, 25, 26	String	OUTGRS or OUTGRC	direct
27	Blinking string	OUTGRS or OUTGRC	log to file 8 and return
28	"Write on top" string	OUTGRS or OUTGRC'	direct
29	Label menu button	OUTGRS	log to file 8 and return
30	Write error message	OUTGR1	N/A
31	Reverse lettered string	OUTGRS or OUTGRC	direct
32	String	OUTGRS or OUTGRC	direct
33	Blink rectangle and position cursor	OUTGR4	log to file 8 and return
34	Activate menu button n	OUTGR1	N/A
35	Deactivate menu button n	OUTGR1	N/A
36	Write special symbol n	OUTGR3	call SYMBOL(KOORD3,KOLOR,IWIN,KOORD1,KOORD2) (see C.4)
37	Configure screen for mode m	OUTGR1	N/A
38	Write question n and position for answer	OUTGRS	log to file 8 and return
39	Purge light pen queue	OUTGR1	log to file 8 and return
40	Draw auxiliary menu	OUTGR1	N/A

Op Code	Function	Called Through	PC Implementation
41	Filled rectangle for composite colors	OUTGR4	direct as 1 & 2
42	not used	OUTGR1	N/A
43	not used	OUTGR1	N/A
44	Transmit Select Mode options to BASIC	OUTGRS	log to file 8 and return
45	Correct composite colors in Lower Window	OUTGR1	N/A
46	String with right 14 characters reverse lettered	OUTGRS or OUTGRC	direct
47	String	OUTGRS	direct
48	Send HELP line	OUTGRS	log to file 9 and return
49	Send continued HELP line	OUTGRS	log to file 9 and return
51	Send parameters for Message Window status report	OUTGRD	N/A
52, 50	Not used	-	-
53	Send parameters to write tick marks for a time scale	OUTGRD	N/A
54	Send parameters to write file, network, save times in Lower Window	OUTGRD	N/A
55	Not Used	_	-
56	Send parameters to write header for Lower Window consumption table	OUTGRD	N/A
57	Send parameters to write start/ finish time line for Status Mode	OUTGRD	N/A

Op Code	Function	Called Through	PC Implementation
58	Send parameters to write Status Mode Message Window	OUTGRD	N/A
59	Send parameters to write cost resource parameters	OUTGRD	N/A
60, 61, 62, 63, 64	Not used	-	-
65	Dashed unfilled rectangle	OUTGR4	direct
66	Solid vector	OUTGR4	direct
67	Dashed vector	OUTGR4	direct

Milestone C replaces all these routines by new IPMPC ones. Also, several new subroutines will be created to replace BASIC-executed subroutines. In some cases these new routines are called from within OUTGR's, and in others the calling FORTRAN routine is completely substituted.

C.3.2 Output Inhibition

GITPASE has a 'touch queue' feature which inhibits screen updates while several locator inputs are given. When /CONTRL/ variable KUEON≠0, such an output block is in effect. All PC output routines that make any HALO calls should include a check of KUEON before sending any output. If KUEON≠0, merely return.

C.4 Primitives Replacements

Routines OUTGR2, OUTGR3, OUTGRC, OUTGRS and OUTGR of the present system will be replaced by new HALO-oriented code. Others will be unneeded because routines that call them are replaced.

Table C2 details the call parameters of the needed new routines. Table C1 gave specific option processing.

C.4.1 OUTGR4

Subroutine OUTGR4 is to be a HALO rectangle generating routine similar to RECT in the demonstration program. First setup one call to PCVTXY with the 4 coordinates. Then write the required rectangles according to IOPT and KOLOR. Save the last input (x,y) pair as new comon /IBMPC/ variable (LASXX,LASYY).

C.4.2 OUTGR2

Subroutine OUTGR2 has 2 purposes. Option 3, 4 and 6 are 'continuation', i.e., the draw rectangles using the present cursor position (LASXX,LASYY) as one corner. Setup a call to OUTGR4 with those two and the passed two coordinates.

Options 20 and 21 write symbols at a location (see C.5). Merely generate the indicated call to SYMBOL.

C.4.3 OUTGR3

Subroutine OUTGR3 also has multiple purposes. When option 7 is used an OUTGR4 call is generated. The final 'coordinate' of option 36 is not a location, but a symbol number. Call SYMBOL.

C.4.4 OUTGR

Subroutine OUTGR is to be a HALO vector sequence calling routine. It should call PCVTXY once, then pass results to HALO's POLYLA after setting color and line style.

Table C2. OUTGR Subroutines

Name and Parameters*

OUTGR2(IOPT,KOLOR,IWIN,KOORD1,KOORD2)

OUTGR3(IOPT,KOLOR,IWIN,KOORD1,KOORD2, KOORD3)

OUTGR4(IOPT,KOLOR,IWIN,KOORD1,KOORD2, KOORD3, KOORD4)

OUTGRC(IOPT,KOLOR,IWIN,KOORD1,KOORD2,STR801)
-STR80=a CHARACTER*80 input with delimiters

OUTGRS(IOPT,KOLOR,IWIN,KOORD1,KOORD2,LENSTR,ISTRNG)
-LENSTR is the string length in characters
-ISTRNG is the vector containing the string
2 characters per word
(i.e., format A2 except A1 last on odd lengths)

OUTGR(IOPT,KOLOR,IWIN,NUMKRD,KOORDS)
-NUMKRD is the number of coordinates (x,y,x...) sent
-KOORDS is the vector containing the coordinates

---

*In all the above
- IOPT is the option code
- KOLOR is the color to be used
-IWIN is the window member in which coordinates apply
-KOORD is a coordinate (first x, second y, third x, etc.)

C.4.5 OUTGRS

Subroutine OUTGRS writes strings to the screen similarly to demonstration program routine WRISTR. First setup one call to PCVTX to convert x & y addresses. Then add INFONTY to I4YOUT to adjust for HALO/GITPASE cursor definition differences. The INTEGER*2 input is reduced to a string with delimiters and HALO called. In option 24, the 4th character is deleted. In several options the string is split in two parts, one normal letters and one reverse lettered. All writing except reverse letter portions should be with overstrike latched (SETTXT (1,1,0,0) to minimize difficulties with very tight screen spacing.

C.4.6 OUTGRC

OUTGRC is similar to OUTGRS except that input is a CHARCTER*80 string with delimiters already included instead of an INTEGER*2 vector. The string can be passed directly to HALO's TEXT unless reverse letters are used. As with OUTGRS overstrike is latched except on reverse letters.

C.5 Special Primitives

In addition (or through calls from) the above main routines certain special routines are required) in GITPASE.

C.5.1 Symbols

Certain special symbols are to be written by new subroutine

SYMBOL(NSYMB,KOLOR,IWIN,KOORDX,KOORDY)

The pair (KOORDX,KOORDY) defines the upper left corner of a KARX=8 by KARY=10 pixel box in which the symbol should appear in the left 7 x pixels and (top down) y-pixels 2 through 8. Begin by converting (KOORDX,KOORDY) via PCVTXY. For the following merely write the needed character in the usual overstrike-latched manner.

NSYMB	Character
-1	* (asterisk)
6	~ (tilde)

For other cases (NSYMB=0 to 5) draw the shape indicated in Table C4 with graphics. Fill the 7 by 7 block.

C.5.2 Highlight/Dehighlight

Elaborate the dummy routines HLIT and DEHLIT of Milestone B into callers of OUTGR4. HLIT should make an IOPT=65 call in KOLOR=14 to mark a rectangular area in dashed light yellow. DEHLIT should reverse the process by making the same call in KOLOR=0=black.

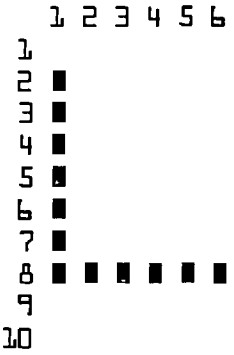
C.6 Test Program

Demonstrate the above routines by a dummy calling program. The program should call INITLE (which calls PCINIT), then it should solicit test requests from the tester, clear screen with OUTGR 4(NUMBR(1), NUMBR0,

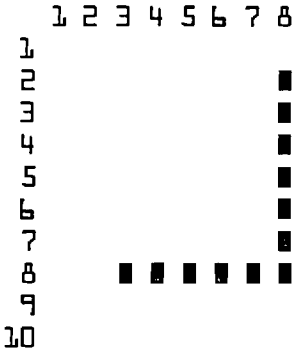
Table C3. Special Symbols

Symbols are 10 dots high, numbered from the top, with the font in the center 8 of the 10 dots, and are 8 dots wide, numbered from the left.

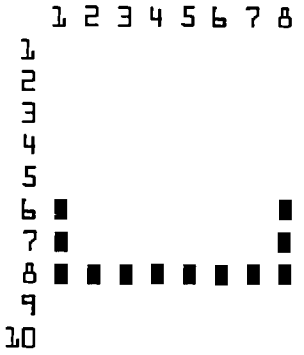
a. Fixed Start (op=36, n=1) SYMBOL(1,...)



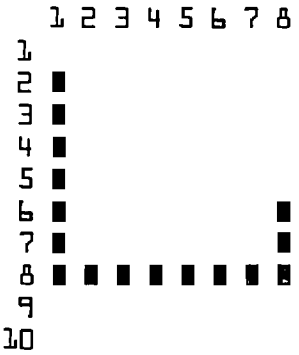
b. Fixed Finish (op=36, n=2) SYMBOL(2,...)



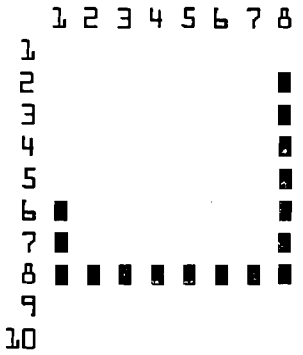
c. Fixed Duration (op=36, n=3) SYMBOL(3,...)



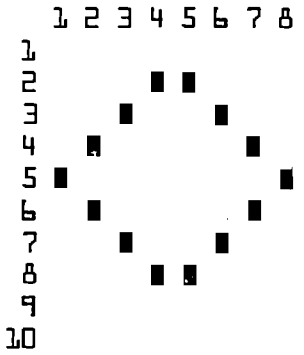
d. Fixed Start and Duration (op=36, n=4) SYMBOL(4,...)



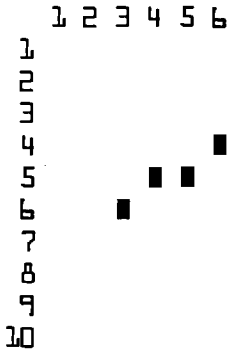
e. Fixed Finish and Duration (op=36, n=5) SYMBOL(5,...)



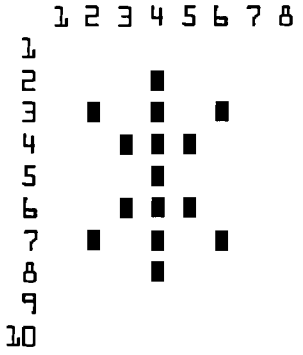
f. Hierarchical Tag (op=21) SYMBOL(0,...)



g. Member Tag (tilde) (op=36,n=6) SYMBOL(6,...)



h. Asterisk (op=20) SYMBOL(-1,...)



NUMBR0, MNTMLX, MNTMLY, MXTMLY, MXTMLY, and do indicated graphics. Check that each indicated routine works in colors 0-15, all appropriate windows, and with and without wrapped (split) screens.

To generate wrapped screens set

```
LNwwT = 8000
LNwwB = 8000 - (LSTP-LSB)
LNwwW = 6000
```

where LSTP and LSB are as in PCVTXY. To generate unwrapped screens set

```
LNwwB = 8000
LNwwW = 6000
LNwwT = 6000 + (LSTP-LSB) - 191
```

```
ls
em Fortran 3.2 compile and link of %1.for
or1 %1;
as2
ink %1 \fortran\sbeep,,,\fortran\fortran \fortran\math \fortran\halocf \mouse\m
use
em Link complete as %1.exe
```

```
  ---commons for program gittst.for

  integer*4 i4col,i4colr,i4row,i4x1,i4x2,i4y1,i4y2
  common /git1/i4col,i4colr,i4row,i4x1,i4x2,i4y1,i4y2

  common /git2/ifontx,ifonty,
 $    iopsy,karx,kary,lslob,lslol,lslor,lslot,
 $    lsmnb,lsmnl,lsmnr,lsmnt,lsmsb,lsmsl,lsmsr,lsmst,
 $    lstil,lstir,lstnb(3),lstnt(3),
 $    mmtmlx,mntmly,mxcol,mxtmlx,mxtmly,numbr0,numbr(100)

  character*80 str80
  common /git3/str80
```

```
      program gittst
      implicit integer*2 (i-n)
      ---test and demonstrate various features of halo and mouse
      ---processing in connection with the pc640 board, micorsoft
      ---mouse and gitpase.

$include: 'gitcom.for'

      dimension inline(80),mousxx(20),mousyy(20)

      ---notes
      1. halo calls require i*4 integers
      2. mouse calls require i*2 integers
      3. beep and smode require i*2 integers
      4. beep/smode halo conflict by duplication of 'scroll'
      (repaired in sbeep by renaming the fortpak 'scrole')
      5. halo text routines expect an 80 length character input
         with delimiters to specify its true length
      6. before running the mouse control must be installed by the
         command `mouse`
      7. halo text character is min 8x by 8y, including an empty
         pixel on right and bottom.
      8. microsoft fortran does not permit data statements to
         assign strings to integer variables.
      9. halo has 0,0 in upper left while gitpase assumes lower left
      10.halo has cursor coordinates in lower left while gitpase
         assumes upper left

      ---literal x space as integer
      data litx/8312/

      ---initialize
      call initl

      ---draw screen boundaries
      call setcol(0)
      call clr
      call rect(numbr0,numbr(7),lstil,lsmnt,lslor,lslob)
      call rect(numbr0,numbr(7),lstil,lsmnt,lstir,lslob)
      call rect(numbr0,numbr(7),lstil,lslot,lslor,lslob)
      call rect(numbr0,numbr(7),lsmsl,lsmst,lsmsr,lsmsb)

      ---draw menu
      call mnubtn(numbr(1),numbr(25),numbr0)
      call mnubtn(numbr(4),numbr(4),numbr(1))
      call mnubtn(numbr(14),numbr(14),numbr(1))

      ---try character spacing
      call spcchr

      ---mouse controlled loop
      kolor=0
00    continue
         ---get a queue of mouse input
         call mousin(mousxx,mousyy,numous)
         kolor=kolor+1
         if(kolor.gt.mxcol)kolor=1
```

```
            ---only 1 was received so solicit and write string
            call keyin(inline,inlen)
            ---check abort
            if(inlen.lt.1)goto 600
            if(inline(1).eq.litx.and.inlen.eq.1)goto 900
            ---send characters to the screen
            numrev=inlen-1
            numrev=max(numbr(1),numrev)
            numrev=min(numrev,numbr(5))
            numdir=inlen-numrev
            call wristr(kolor,mousxx(1),mousyy(1),numdir,
     $         numrev,inline)
            goto 600
600         continue
            ---more than 1 so make rectangle of first two
            call rect(numbr(1),kolor,mousxx(1),mousyy(1),
     $         mousxx(2),mousyy(2))
00       continue
         goto 200

      ---exit
00    continue
      ---end halo
      call closeg
      ---restore ibm-board mode
      mode=3
      call smode(mode,ierr)
      stop
      end

      ------------------------------------------------------------------

      subroutine brkstr(str,istr,len)
      implicit integer*2(i-n)
      ---break character string into single integer a1 formats until
      ---delimiter repeat sets length len
      ---str must be char*80 and istr dimensioned 80 in calling program

nclude: 'gitcom.for'
      character*80 str
      dimension istr(1)

      ---break into a1
      read(str,200)(istr(iii),iii=1,80)
00    format(80a1)

      ---scan to delimiter
      idelim=istr(1)
      do 300 llen=1,79
         if(istr(llen+1).eq.idelim)goto 900
         istr(llen)=istr(llen+1)
00    continue
      llen=80

      ---exit
00    continue
      len=llen-1
      return
      end

      ------------------------------------------------------------------

      subroutine colors
      implicit integer*2(i-n)
      ---establish a pitpase-compatible color pallet
```

```
include: 'gitcom.for'

      integer*4 ired,igrn,iblue,i4plet(16,3)
      character*15 str15
      dimension istr(15)

      data i4plet/
     $    15,10,13,15, 0, 0, 0, 0, 6, 0, 5, 0, 3,11, 0, 2,
     $    15,10, 0, 0,11,14, 0, 0, 6, 5, 7, 7, 6, 0, 0, 2,
     $    15, 0,13, 0,11, 0,13, 0, 6, 5, 9,15, 1, 5, 5, 2/

      ---revise color pallet to gitpase-compatible starting point
      do 70 idx=1,mxcol+1
         i4colr=idx-1
         call setpal(i4colr,i4plet(idx,1),i4plet(idx,2),
     $       i4plet(idx,3))
0     continue

      ---loop to correct if desired
00    continue
         call setcol(0)
         call clr
         iy1=mxtmly-4*kary
         maxi=mxcol+1
         do 200 idx=1,maxi
            kolor=idx-1
            iy2=iy1-20
            ix1=0
            ix2=60
            call rect(numbr(1),kolor,ix1,iy1,ix2,iy2)
50          format(i2,'=',3i4)
            i4colr=kolor
            if(iopsy.ne.3)goto 160
               call inqpal(i4colr,ired,igrn,iblue)
               write(str15,150)kolor,ired,igrn,iblue
               goto 165
60          continue
               str15=' '
62             format('color no.= ',i4)
               write(str15,162)kolor
65          continue
70          format(80a1)
            read(str15,170)(istr(iii),iii=1,15)
            ix1=70
            call wristr(kolor,ix1,iy1,numbr(15),numbr0,istr)
            ix1=195
            kolbk=kolor
            if(kolbk.eq.0)kolbk=mxcol
            call wristr(kolbk,ix1,iy1,numbr0,numbr(15),istr)
            iy1=iy1-25
00       continue
         ---see if any are to change
         write(0,*)'enter revise color,r,g,b else -1,0,0,0'
         call beep
         read(0,*) i4colr,ired,igrn,iblue
         if(i4colr.lt.0.or.iopsy.ne.3)goto 900
            call setpal(i4colr,ired,igrn,iblue)
            goto 100

      ---exit
00    return
      end

      ------------------------------------------------------------
```

```
      implicit integer*2(i-n)
      ---initialize various inputs and i/o systems
include: 'gitcom.for'

      ---set operating system flag 3=scion, 4=ibm board
      ---at med resolution, 5=ibm board at hi resolution
      iopsy=3
      if(iopsy.ne.3)write(0,*)' op system?'
      if(iopsy.ne.3)read(0,*)iopsy

      ---set number*2 constants
      numbr0=0
      do 200 num=1,100
         numbr(num)=num
00    continue

      ---terminal screen min limits
      mntmlx=0
      mntmly=0

      ---initialize halo
      if(iopsy.ne.3)goto 150
         ---scion board at default address region 320K-576K
         call setseg(5)
         call initgr
         mxcol=15
         mxtmly=479
         mxtmlx=639
         goto 190
50    continue
         ---ibm board
         call initgr(iopsy-4)
         mxcol=3
         mxtmly=199
         mxtmlx=319
         if(iopsy.eq.4)goto 190
            ---hi res
            mxcol=1
            mxtmlx=639
90    continue

      ---set string character sizes
      ifontx=8
      ifonty=8
      karx=ifontx
      kary=ifonty+2

      ---initialize the mouse
      nn1=0
      call mouses(nn1,nn2,nn3,nn4)
      ---halo crosshair cursor will mark mouse position
      i4x1=3*ifontx
      i4y1=3*ifonty
      if(iopsy.eq.3)i4colr=14
      if(iopsy.eq.4)i4colr=3
      if(iopsy.eq.5)i4colr=1
      call inithc(i4x1,i4y1,i4colr)
      ---set ibm board to hi-res graphics so that mouse will scale ok
      mode=6
      call smode(mode,ierr)

      ---demonstrate that read and write possible in graphics mode
      write(0,730)
30    format(' enter any 2-digit integer'/' ?'\)
      call beep
```

```
740   format(i2)
      write(0,750)intg
750   format(' integer',i3,' was received')

      ---initialize color pallet
      call colors

      ---initialize screen boundaries
      lsmnt=mxtmly-kary-1
      lstnb(1)=lsmnb+(2*(lsmnt-lsmnb)/3
      lstnb(2)=lsmnb+(lsmnt-lsmnb)/3
      lstnb(3)=lsmnb
      lstnt(1)=lsmnt-2*kary
      lstnt(2)=lstnb(1)-2*kary
      lstnt(3)=lstnb(2)-s*kary
      wmm=(lsmnt-lsmnb+6)/25.0
      wma=(lslor-lslol-100)/17.0
      lslob=0
      lslot=lslob+9*kary+2
      lsmnb=lslot
      lstil=3*karx-1
      lstir=18*karx-1
      lsmnl=lstir
      lsmnr=mxtmlx
      lslol=lsmnl
      lslor=lsmnr
      lsmsl=lstil
      lsmsr=lstir
      lsmsb=lslob
      lsmst=lsmsb+6*kary+1

      ---exit
900   continue
      return
      end

      ------------------------------------------------------------

      subroutine keyin(inline,inlen)
      implicit integer*2(i-n)
      ---read in a single character string line

include: 'gitcom.for'

      dimension inline(80)

      inblnk=32*257

100   continue

200   format(80a1)
      call prompt('  TYPE ')
210   format(/' ?'\)
      write(0,210)
      read(0,200,err=500,end=600)inline
      do 300 idx=1,80
         kar=81-idx
         if(inline(kar).ne.inblnk)goto 400
300   continue
      kar=0
400   continue
      inlen=kar
      call prompt('  WAIT ')
      goto 900
```

```
500   continue
      write(0,*)' error on read'
      stop 2

      ---end of file
600   continue
      write(0,*)' end of file detected'
      stop 1

      ---successful read
800   continue

      ---exit
900   continue
      return
      end

      ------------------------------------------------------------

      integer*2 function konvy(iny)
      implicit integer*2 (i-n)
      ---convert iny to halo out form.

include: 'gitcom.for'

      ---reverse origin to upper left
      konvy=mxtmly-iny
      return
      end

      ------------------------------------------------------------

      subroutine mnubtn(nbtn1,nbtn2,isfill)
      implicit integer*2 (i-n)
      ---fake menu buttons nbtn1,nbtn2--filled if isfill=1

include: 'gitcom.for'

      character*80 strmnu
      dimension mnu(80)

      strmnu='\mnu\'
      call brkstr(strmnu,mnu,idum)
      iyper=(mxtmly+6)/25

      ---first black behind
      ny1=iyper*nbtn2-6
      ny2=iyper*(nbtn1-1)-5
      if(ny2.lt.0)ny2=0
      nx2=lstil-1
      call rect(numbr(1),numbr0,numbr0,ny1,nx2,ny2)

      ---then write each in turn
      nx1=0
      nx2=lstil-2
      do 500 nbtn=nbtn1,nbtn2
         ---the colored box
         ny1=iyper*nbtn-6
         ny2=iyper*(nbtn-1)
         kolor=7
         if(1.le.nbtn.and.nbtn.le.6)kolor=nbtn
         if(7.le.nbtn.and.nbtn.le.12)kolor=nbtn+1
         if(nbtn.eq.25)kolor=4
         call rect(isfill,kolor,nx1,ny1,nx2,ny2)
         ---then the label
```

```
      if(isfill.eq.1)kolor=0
      call wristr(kolor,nx1,ny1,numbr(3),numbr0,mnu)
500   continue

      ---exit
900   continue
      return
      end

      ------------------------------------------------------------

      subroutine mousin(mousxx,mousyy,numous)
      implicit integer*2(i-n)
      ---collect a queue of up to 20 locations from the
      ---Microsoft mouse.  Number collected in numous.
      ---Coordinate pairs are (mousxx(i),mousyy(i)).

$include: 'gitcom.for'

      dimension mousxx(1),mousyy(1)

      ---clear the mouse buttons
      nn1=5
      nn2=1
      call mouses(nn1,nn2,nn3,nn4)
      nn1=5
      nn2=0
      call mouses(nn1,nn2,nn3,nn4)
      i4x1=-1
      i4y1=-1

      ---set the y-scaling factor
      scaly=mxtmly+1
      scaly=scaly/200

      ---signal ready
      call prompt(' TOUCH ')

      ---pickup up to two x,y pairs
      do 400 numous=1,20

         ---loop until button interrupt
100      continue
            ---mark the present position with the crosshair cursor
            nn1=3
            call mouses(nn1,nn2,nxx,nyy)
            nyy=nyy*scaly+.5
            if(i4x1.eq.nxx.and.i4y1.eq.nyy)goto 250
               i4x1=nxx
               i4y1=nyy
               call movhca(i4x1,i4y1)
250         continue

            ---check left button for hit and exit
            nn1=5
            nn2=0
            call mouses(nn1,nn2,mousxx(numous),mousyy(numous))
            if(nn2.gt.0)goto 500

            ---check right button for new hit
            nn1=5
            nn2=1
            call mouses(nn1,nn2,mousxx(numous),mousyy(numous))
            if(nn2.gt.0)goto 400
```

```
      goto 200

00    continue
      numous=20

      ---end touch solicit
00    continue
      call prompt('  WAIT ')
      call delhcu

      ---reverse and scale the x and y values
      do 600 iii=1,numous
         mousyy(iii)=(199-mousyy(iii))*scaly+.5
00    continue

      ---exit
00    continue
      return
      end

      ------------------------------------------------------------

      subroutine prompt(msg)
      implicit integer*2 (i-n)
      ---simulate giving prompt message msg
include: 'gitcom.for'
      character*7 msg
      dimension istr(7)
      read(msg,200)(istr(iii),iii=1,7)
00    format(80a1)
      nxx=lsmsl+4*karx
      nyy=lsmsb+kary
      call wristr(numbr(2),nxx,nyy,numbr0,numbr(7),istr)
      call beep
      return
      end

      ------------------------------------------------------------

      subroutine rect(isfill,kolor,nx1,ny1,nx2,ny2)
      implicit integer*2(i-n)
      ---draw a rectangle in color kolor between (nx1,ny1) and
      ---(nx2,ny2).  Fill if isfill=1.

include: 'gitcom.for'

      integer*4 i4hold

      ---set the color
      i4colr=kolor
      if(i4colr.gt.mxcol)i4colr=mxcol
      call setcol(i4colr)
      ---convert the x-addresses
      i4x1=nx1
      i4x2=nx2
      ---convert the y-addresses
      i4y1=konvy(ny1)
      i4y2=konvy(ny2)
      ---sort the x-addresses (l to r)
      if(i4x1.le.i4x2)goto 300
         i4hold=i4x1
         i4x1=i4x2
         i4x2=i4hold
00    continue
      ---sort the (converted) y-addresses (t to b)
```

```
      ' i4hold=i4y1
        i4y1=i4x2
        i4y2=i4hold
400   continue

      ---call the appropriate routine
      if(isfill.eq.1)call bar(i4x1,i4y1,i4x2,i4y2)
      if(isfill.ne.1)call box(i4x1,i4y1,i4x2,i4y2)

      ---exit
900   continue
      return
      end

      ------------------------------------------------------------

      subroutine spcchr
      implicit integer*2 (i-n)
      ---test character spacing of gitpase screen
$include: 'gitcom.for'
      dimension istr(80)

      ---title window
      nyy=lsmnt-3*kary
      nxx=lstir-14*karx+1
      str80='\activity nam:0\'
      call brkstr(str80,istr,lenstr)
      call wristr(numbr(1),nxx,nyy,numbr0,lenstr,istr)
      nyy=nyy-kary
      call wristr(numbr(2),nxx,nyy,numbr0,lenstr,istr)

      ---time scale
      nyy=lsmnt+kary+1
      str80='\01:01    01:02\'
      call brkstr(str80,istr,lenstr)
      call wristr(numbr(1),lsmnl,nyy,numbr0,lenstr,istr)

      ---main window
      nyy=lsmnt-4*kary
      nxx=lsmnl+1
      str80=
     *'\123456789+123456789+123456789+123456789+123456789+123456789+12\'
      call brkstr(str80,istr,lenstr)
      call wristr(numbr(1),nxx,nyy,numbr0,lenstr,istr)
      nyy=nyy-kary
      call wristr(numbr(2),nxx,nyy,numbr0,lenstr,istr)

      ---message window
      nxx=lsmsl+1
      nyy=lsmst-2*kary-1
      str80='\message of --15\'
      call brkstr(str80,istr,lenstr)
      call wristr(numbr(1),nxx,nyy,numbr0,lenstr,istr)

      ---triple size character
      call movtca(mxtmlx/2,mxtmly/2)
      call settex(3,3,0,1)
      call settcl(0,1)
      str80='\X\'
      call text(str80)
      call settcl(1,0)
      call text(str80)
      call settcl(0,1)
      call text(str80)
      call deltcu
```

```
---exit
900   continue
      return
      end

      ------------------------------------------------------

      subroutine wristr(kolor,nxx,nyy,numdir,numrev,istr)
      implicit integer*2 (i-n)
      ---write a string from integer storage at location nxx,nyy
      ---in color kolor with numdir characters normal and numrev
      ---reverse lettered

$include: 'gitcom.for'

      character*1 delim
      dimension istr(1)
      data delim,idelim/'}',8317/

      ---convert color and y-coordinate
      i4colr=kolor
      if(i4colr.gt.mxcol)i4colr=mxcol
      i4y1=konvy(nyy)

      ---adjust to halo cursor point in lower left
      i4y1=i4y1+ifonty
      i4x1=nxx

      ---first the normal lettered part
      if(numdir.lt.1)goto 400
         ---convert to character with delimiters
         str80=' '
         write(str80,200)idelim,(istr(iii),iii=1,numdir),
     $       idelim
200      format(80a1)
         ---set color
         call settcl(i4colr,0)
         call settex(1,1,0,0)
         ---set location
         call movtca(i4x1,i4y1)
         ---write
         call text(str80)
400    continue

      ---next send the reverse lettered part
      if(numrev.lt.1)goto 900
         ---convert to character with delimiters
         mini=numdir+1
         maxi=numdir+numrev
         str80=' '
         write(str80,200)idelim,(istr(iii),iii=mini,maxi),
     $       idelim
         ---set color
         call settcl(0,i4colr)
         call settex(1,1,0,1)
         ---set location
         i4x1=i4x1+numdir*karx
         call movtca(i4x1,i4y1)
         ---write
         call text(str80)

      ---exit
900    continue
      call deltcu
      return
```

12 July 1984
E-24-669- Item A003
Milestone E Specifications
Project Director: Donovan B. Young

Milestone E: Complex Displays

By Ronald L. Rardin

The Phase III version of the GITPASE project management system employs a hybrid of a VAX computer running FORTRAN and a Chromatics color graphics microprocessor running BASIC. The Milestone C/D specification developed graphics and string primitives interfacing to HALO. This milestone specifies more complex routines calling those primitives (or HALO directly). It presumes the environment of Milestone C.1 and refers to the demonstration program attached to that Milestone.

E.1 Menus and Mode Screens

One fundamental element of GITPASE screens is menus. Twenty-five menu blocks are always present at the extreme left of the screen. An additional 14 appear sometimes in the Lower Window as the "auxiliary menu". Menu blocks or buttons are identified by "relative number" where 1-25 are the permanent ones (bottom to top), 26-32 are the first line of the auxiliary menu (left to right), and 33-39 are the second (left to right). Blocks 1-12 are a "color menu" with one button for each of the colors 1-13 except white.

E.1.1 Subroutine MNUBTN

Create replacement subroutine MNUBTN (NBTN1, NBTN2, ISFILL) to draw and label menu buttons NBTN1 through NBTN2. If ISFILL=0 the buttons are not filled and have labels in the color of their boundaries. If ISFILL=1, the buttons are filled and labels are in black. Since blink is unavailable on the PC, menu blocks will normally be unfilled. They will be redrawn filled when active.

All menu blocks for the PC are /CONTRL/ common variable WMM-6 pixels high by 3*KARY-2 pixels wide. Labels are vertically centered and over the left boundary. (This causes one pixel to "stick out" on the right, but is unavoidable with tight PC scaling). Lower left menu block corners are as follows (HALO coordinates):

Relative No. (n)	x	y
1-25	0	MXTMLY-IJMM*(n-1)
26-32	LSLOL+58+WMA*(n-26)	MXTMLY-LSLOT-17
33-39	LSLOL+58+WMA*(n-33)	MXTMLY-LSLOT-36

It is essential that menu writing proceed as rapidly as possible. Thus, MNUBTN should call HALO directly, using no GITPASE primitives.

The first processing step if ISFILL=0 is to 'black behind', i.e. draw one black filled rectangle for any parts of the NBTN1 to NBTN2 range in the interval 1 to 25, and/or one such rectangle for the interval 26-32 and/or one rectangle for the interval 33-39. Then, taking buttons NBTN1 to NBTN2 in order, draw and label.

Blocks 1 to 12 are the color blocks (in colors 1 to 6 and 8 to 13). If KURMOD=0 (Transition) or /NETWRK/ variable ICOLR(n)=0, button n is unlabeled. Otherwise the label of n is /NETSAV/ (FORMAT A2,A1) integer*2 values

IRNAM(1,ICOLR(n)), IRNAM(2,ICOLR(n)).

Blocks 13 to 39 have mode-dependent labels. Table E1 shows button colors and labels for all modes. Items in Table E1 should be loaded by MNUBTN DATA statements into CHARACTER*5 array MNULAB(n-12,mode). Each entry will be the 3 characters with delimiters so that HALO routine TEXT can be called without processing.

INTEGER*4 array MNUCLR (1 to 39) will be similarly loaded with menu block colors.

A few menu blocks, e.g. number 19 of activity mode, have 'cross out' lines through their labels. Graphically add these lines as the last MNUBTN step (in color if ISFILL=0 and in black otherwise).

E.1.3 Subroutine MODSCN

Replacement subroutine MODSCN (NEWMOD) creates initial screens for GITPASE modes. It first sets KURMOD=NEWMOD, then erases the whole screen with BLACK(NUMBR(1)), then draws the window boundaries of Milestone C figure C1 in white using OUTGR4 window 0. It then uses MNUBTN to draw unfilled menu blocks 1 to 25 (1 to 39 in Transition Mode). The 'GITPASE' logo is added in double high letters in the space above the Message Window.

Mode 2 = Resource has the title

'ßCD/BGßWT/AM' repeated 5 times above the Main Window.

Mode 4 = Status has the heading

'ßßßßFIRMßßßTEMPßßßßßßßSCHEDULEß

twice, with the last 10 characters reverse-lettered. Write these titles using subroutine OUTGRC.

E.2 Miscellaneous Displays

E.2.1 Error Messages

When GITPASE detects a user error, it displays a 15-character message in text line 5 of the Message Window. Table E2 shows a

Table E1. Menu Labels by Mode

Relative No.	Button Color	Transition Mode (0)	Activity Mode (1)	Resource Mode (2)	Schedule Mode (3)	Status Mode (4)	Calendar Mode (6) and Select Mode (5)
13	7	HLP	HLP	HLP	HLP	HLP	HLP
14	7	↑↓Ø	↑↓Ø	↑↓Ø	↑↓Ø	↑↓Ø	
15	7	NEU	NEU	NEU	NEU	NEU	NEU
16	7	BAK	BAK	BAK	BAK	BAK	BAK
17	7	PRT	PRT	PRT	PRT	PRT	PRT
18	7		Ø~Ø		RFM		
19	7		←—		ACC		
20	7		<??		SER	ADV	
21	7		<:0		~~DUR~~	SK+	
22	7	Ø↓Ø	STD		~~S/F~~	SK=	
23	7	Ø↑Ø	SEO		DUR	WI+	
24	7	SIM	DEL	DEL	S/F	WI=	
25	4		MNU	MNU	MNU	MNU	
26	2	ACT	ACT	ACT	ACT	ACT	ACT
27	2	RES	RES	RES	RES	RES	RES
28	2	SCH	SCH	SCH	SCH	SCH	SCH
29	2	STA	STA	STA	STA	STA	STA
30	2	SEL	SEL	SEL	SEL	SEL	SEL
31	2	CAL	CAL	CAL	CAL	CAL	CAL
32	2						
33	6	OLD	LOD	LOD	LOD	LOD	LOD
34	6	NEW	SAV	SAV	SAV	SAV	SAV
35	6	REN					
36	6	PUR					
37	6	DIR					
38	6	RST					
39	4	END	TRN	TRN	TRN	TRN	TRN

list. The message is colored in white (7). In the Chromatics version of GITPASE, it blinks.

Create replacement subroutine ERRMSG(NUM) to write error message NUM. It should call BLACK(10), then setup and call OUTGRC to write the message in white (7). Use DATA statements in ERRMSG to load CHARACTER*17 array IERMSG(26) with the message literals (and delimiters).

E.2.2 Message Window Schedule Report

Replacement subroutine NETMSG writes a 4-line schedule status report in the first 4 lines of schedule status report in the first 4 lines of the Message Window. It uses subroutine PAKPER(INPERD, IPACK) to convert interval period numbers INPERD to external period codes IPACK (3) in (A2, A1, A2) format (INTEGER*2).

First call BLACK(NUMBR(9)) to erase the area. Then display via internal writes and OUTGRC.

(i) Line 1 is green (2) and displays PAKPER of IDESST and ITRUST in format

('ds',A2,A1,A2,'ϸss',A2,A1,A2)

(ii) Line 2 is red (4) and displays PAKPER of IDESFN and ITRUFN in the format

('df',A2,A1,A2,'ϸsf',A2,A1,A2)

(iii) Line 3 is yellow (6) and displays

IFIX(TIMINF+.5) and RESINF in format

('ti=', I3,',ri=',F5.1)

(iv) Line 4 is magenta (5) and displays COST in format

('$$$=',F11.2)

E.2.3 Time Scale Ticks

Replacement subroutine TIMTIC draws time ticks for timescales. TIMTIC will begin by calling OUTGR4, window 0, to black an 11-dot-high area from x=LSMNL+1 to x=LSMNR-1 at y=LSMNB+11 in Resource Mode (KURMOD=2) and y-LSMNT+11 otherwise. Then draw white, 3-dot-high "ticks" across the bottom of this time scale region. Ticks are spaced to provide for periods (/CONTRL/ variables) MNTIM through MXTIM, with the tick for the left end of MNTIM at x=LSMNL and the one for the right end of MXTIM at x=LSMNR. Call HALO routines directly to improve speed.

Table E2. GITPASE Error Messages

Error Number	Message
1	Program␣Error␣␣
2	File␣Not␣Found␣
3	Data␣Is␣Cyclic␣
4	No␣Active␣File␣
5	Time␣Out␣Range␣
6	Can't␣Scal␣Mode
7	Incomplete␣Data
8	Invald␣Hit␣Locn
9	Value␣Out␣Range
10	Invalid␣Format␣
11	Can't␣Add␣More␣
12	Duplicate␣Code␣
13	Duplicate␣Time␣
14	Resource␣Unused
15	Invalid␣Characs
16	Invalid␣Length␣
17	Commucn␣Failure
18	Unimplemented␣␣
19	Invalid␣For␣DDA
20	Before␣Consumpn
21	Already␣Started
22	Status␣Conflict
23	Invalid␣DDA␣␣␣␣
24	Time␣Perd␣Confl
25	Invalid␣Code␣␣␣
26	Too␣Much␣Input␣

E.2.4 File/Network/Save Time Message

Subroutine OUTINF sets up three-line messages about file, network, and save times the Lower Window. Replace this routine with a new routine OUTINF that directly writes the information.

The messages required form two columns at x=LSLOL+6 and x=LSLOL+294 by y=LSLOB+40, y-LSLOB+30, and y=LSLOB+20 of the Lower Window. Begin with a call to OUTGR4 to black LSLOL+1 to LSLOR-1 and LSLDB+1 to LSLOB+40. Only line 1 appears in Transition Mode (KURMOD=0).

(i) Line 1 is in magenta (5)

(a) Column 1 displays KURFIL in format

('File=',A20)

(b) Column 2 displays IFILTM in format

(Saved␣AT␣',4A2)

(ii) Line 2 is in green (2)

(a) Column 1 displays KURNAM in format

('Netw=',6A2)

(b) Column 2 displays INETTM in format

('Saved␣AT␣,4A2)

(iii) Line 3 is in cyan (3) and depends on /NETSAV/ variable ISPLAN and /CONTRL/ variable IKPLN

(a) Column 1 writes 'Available␣Plans=' followed by

'min␣' if ISPLAN(1)≠0

'nom␣' if ISPLAN(2)≠0

'max␣' if ISPLAN(3)≠0

'4␣␣␣' if ISPLAN(4)≠0

'5␣␣␣' if ISPLAN(5) ≠0

If some ISPLAN(i)=0, later strings are shifted left to avoid gaps.

(b) Column 2 writes

'Active␣Plan=' followed by 'none',

'min$', 'nom$', 'max$', '4' or '5',

depending on whether IKPLN is 0, 1, 2, 3, 4 or 5.

Display via internal writes and OUTGRC.

E.2.6 Lower Window Duration Header

Replacement subroutine LOTBHD(NODT) writes the heading for the Lower Window consumption table.

LOTBHD begins by blacking. If NOPT=0, call OUTGR4 window 0 to black the top 11 dots of the Lower Window. If NOPT≠0 black the whole window (do not use subroutine BLACK). Call OUTGR4 again to draw a white line across that window at 11 dots below LSLOT. Compute duration KDUR as NAFN(IKACT)-NAST(IKACT)+1 unless that is zero in which case IANMD(2,NACT) is used. The four durations IANMD(1 to 3, IKACT), and KDUR are then written in white (7) under format ('DUR=',I5,317,'$ACTIVITY='). Use OUTGRC with the last duration reverse lettered. Finally IANAM (1 to 6, IKACT) is written in red. Use OUTGRS.

E.2.5 Status Mode Time Line

Subroutine STATIM (NACT1, NACT2) writes Status Mode time lines in the Main Window for activities NACT1 through NACT2.

Replacement routine STATIM should take each activity NACT in turn. It first blacks window 99 x=LNMNL+1 to LNMNR-1 by y=ISYY(1,NACT) to ISYY(1,NACT)-KARY+1. It then writes six columns of information, three in green (2), three in red (4), and the last of each color reverse lettered. All columns are 10 characters wide, except that the third and sixth are 11 characters wide.

(a) Column 1 relates to /NETSAV/ variable ISST(NACT,1). It contains (3X'(null)'1X) if ISST(NACT,1)=0, the PAKPER of ISST(NACT,1) in format ('acs=',A2,A1,A2,1X) if 0<ISST(NACT,1) ISTATM(1), and the PAKPER of ISST(NACT,1) in format ('pjs=',A2,A1,A2,1X) otherwise.

(b) Column 2 is identical except it refers to ISST(NACT,2) and ISTATM(2).

(c) Column 3 shows the PAKPER of /NETWRK/ variable ICURST(NACT) in format ('cs=',A2,A1,A2,2X).

(d) Column 4 relates to /NETSAV/ variable ISFN(NACT,1). It contains (4X'(null)',1X) if ISFN(NACT,1)=0, the PAKPER of ISFN(NACT,1) in format ('acf=',A2,A1,A2,1X) if 0<ISFN(NACT,1) ≤ ISTATM(1), and the PAKPER of ISFN(NACT,1) in format ('pjf=',A2,A1,A2,1X) otherwise.

(e) Column 5 is identical to 4 except it refers to ISFN(NACT,2) and ISTATM(2).

(f) Column 6 shows the PAKPER of /NETWRK/ variable ICURFN(NACT) in format ('scf=',A2,A1,A2,2X).

Use internal writes and OUTGRC to write this display after blacking. The y-address in Window 99 is ISYY(1,NACT).

E.2.6 Status Mode Message Window

Subroutine STAMSG presently writes the Message Window of Status Mode. Substitute a new STAMSG to perform this function.

The routine should begin with a BLACK(NUMBR(9)) call to black the first four text lines of the Message Window. Then write on the first two lines.

(i) Line 1 is in yellow (6) and displays the PAKPER of ISTATM(1), in format

('firmbthrub',A2,A1,A2)

(ii) Line 2 is in magenta (5) and displays the PAKPER of ISTATM(2) in format

('tempbthrub',A2,A1,A2)

E.2.7 Cost Parameters

Subroutine RESPAR presently writes a four line set of cost calculation parameters in the Title Window. Create new subroutine CSTPAR to perform this write.

The new subroutine writes four lines (KARY apart) in white (7) with upper left corner at x=LNTIL, y=IRYY(KOSRES)-30 of the Title Window (79) (KOSRES and IRYY are in /NETWRK/). It begins with an OUTGR4 call to black the write area. Then use internal writes and OUTGRC to display the 4 lines.

(i) Line 1 displays the PAKPER of IPENTM (/CONTRL/) in format ('pnltybaftb',A2,A1,A2)

(ii) Line 2 displays PENAMT in format ('penlty',F9.0)

(iii) Line 3 displays VALFIN in format ('value',F10.0)

(iv) Line 4 displays DSCRAT in format ('dscount%',F7.3)

E.2.8 HELP Messages

Subroutine

HLPLIN(NXXX,NYYY,ISTRA1,LEN1)

presently writes a line of help message to the graphics screen at window 0 coordinates (NXXX,NYYY). The message is of length LEN1 characters stored in INTEGER*2 array ISTRA1, one character per word (i.e. format A1).

Internal to the string ISTRA1 are certain instructions for special typing. Each is '{' followed by a two digit number. Table E3 details the meaning of the codes.

Since two monitors are available in the IBM version, HELP messages will be diverted to the text screen (IBM monitor). Write a new subroutine HLPLIN that prepares a line for that screen and writes it with an ordinary formatted FORTRAN write to unit 0. Before sending the line remove any '{nn' instructions and insert or correct as shown at Table E3 right.

Note: GITPASE sometimes has very short help lines because of the window to which they were bound. Such lines will automatically be packed together before the HLPLIN call and need not be considered here.

E.3 Test Program

As with earlier Milestone a test calling program should be generated that allows testers to call for any of the above routines at any appropriate parameter setting. The program should call INITLZ as in Milestone C/D, set dummy network data as shown below, then cycle through modes and displays.

E.3.1 Download Routines

To test the routines of this Milestone subroutine BLACK will need to be downloaded from module 'PH3IO1.FOR' and subroutine PAKPER from module 'PH3CAL.FOR'.

E.3.2 Dummy Data

Dummy network data values will be needed by various routines. Use the following to test.

```
MNTIM = 10
MXTIM = 43
ISST(1,1) = 14
    (2,1) =  0
    (3,1) = 19
    (1,2) = 22
    (2,2) = 14
    (3,2) =  0
ISFN(1,1) = 15
    (2,1) =  0
    (3,1) = 20
    (1,2) = 22
    (2,2) = 14
    (3,2) =  0
```

Table E3. Internal HELP '{nn' Instructions

number n	meaning	PC Version implementation
00 to 07	switch to normal printing in color n	none
08 to 15	switch to reverse printing in color (n-8)	none
16	print compound tag	insert '<>ƀ'
17	print fixed start tag	insert '|__ƀ'
18	print fixed finish tag	insert '__|ƀ'
19	print fixed duration tag	insert ',__,ƀ'
20	print fixed start/duration tag	insert '|__,ƀ'
21	print fixed finish/duration tag	insert ',__|ƀ'
22	print member tag	insert '~ƀ'
23	cross through previous 3 characters	insert '(crossed)ƀ'
24	down arrow	insert '(upƀarrow)ƀ'
25	up arrow	insert '(downƀarrow)ƀ'

```
ICURST(1) = 12
      (2) = 14
      (3) = 16
ICURFN(1) = 18
      (2) = 20
      (3) = 22
ITRUST = 12
ITRUFN = 40
IDESST = 11
IDESFN = 35
ISTATM(1) = 15
      (2) = 20
  ISYY(1,1) = 8180
      (1,2) = 8150
      (1,3) = 8120
  IRYY(1) = 8191
      (2) = 8105
      (3) = 8000
KOSRES = 2
RESINF = 123.8
TIMINF = 12.6
COST = 111.111
KURFIL 'TEST0'
KURNAM(1) = 'AB'
      (2) = 'CD'
      (3) = 'EF'
      (4) = 'GH'
      (5) = 'IJ'
      (6) = 'KL'
ISPLAN(1) = 0
      (2) = 1
      (3) = 0
      (4) = 1
      (5) = 0
IKPLAN = 2
IPENTM = 18
PENAMT = 320.65
VALFIN = 100.329
DSCRAT = 11.29
INETTM(1) = 'DA'
INETTM(2) = 'TE'
INETTM(3) = 'Þ#'
INETTM(4) = 'Þ1'
IFILTM(1) = 'DA'
IFILTM(2) = 'TE'
IFILTM(3) = 'Þ#'
IFILTM(4) = 'Þ2'
```

11 July 1984
E-24-669
Milestone F Specifications
Project Director: Donovan B. Young

Milestone F: Input Completion

By Ronald L. Rardin

The Phase III version of the GITPASE project management system employs a hybrid of a VAX computer running FORTRAN and a Chromatics color graphics microprocessor running BASIC. Milestone B took first steps toward replacing input phases of the BASIC code with new FORTRAN and Milestone C/D and E replaced outputs. This Milestone completes inputs. It preserves the environment of Milestone C.1 and references the demonstration program attached to Milestone C.

F.1 Prompt Messages

One function now performed by the BASIC code is to write prompt strings in double-wide letters centered on the last line of the Message Window. Create new subroutine PRMPT(NUM) to display such prompts. (See the similar routine in the demonstration program.)

Table F1 shows the messages that should be supported and the colors in which to write them. Store values of that table in CHARACTER*9 array MSGPRM and KLRPRM set by DATA statements in PRMPT. (MSGPRM includes the delimiters).

Subroutine PRMPT should write the appropriate message by direct HALO call, then call BEEP to ring the terminal bell. (BEEP is part of object module SBEEP.OB5 supplied with the demonstration program).

F.2 Enhanced Keyed Input

Milestone B specified processing for input response codes 2 to 9 and 11 to 15 (see Table B1) through OUTKEY and OUTQA. Complete these two routines by adding prompts and arranging questions and echos of answers to pass to the text (IBM) monitor. Specifically,

(i) Write the 'question' of OUTQA to the IBM screen without a '?'

(ii) Read input in OUTKEY via ordinary FORTRAN read (see demonstration program routine KEYIN).

(iii) Prompt for input in OUTKEY by calling PRMPT (4) for response codes except 15 or PRMPT (5) for code 15. Also write a '?' to the IBM screen.

(iv) Just before exit of OUTQA revise the prompt by a call to PRMPT (1)

Table F1. Prompt Messages

Number	Message	Color
1	'ƀWAITƀƀ'	4
2	'ƀTOUCHƀ'	2
3	'CONFIRM'	2
4	'ƀTYPEƀƀ'	2
5	'ƀ<CR>ƀƀ'	2

F.3 Touch Input

In the present GITPASE, screen touch input is obtained via light pen. The revised IBMPC version will obtain touch input via the Microsoft Mouse.

Two important features of the present design must be retained. First, the system should provide for a queue of several touch locations to be entered at once, i.e., before the screen responds. Many touch inputs in GITPASE are logically paired in user's minds. If both hits cannot be entered at once, the user's thought process is interrupted.

The second important concept is confirming touches. Normal (KOMRSP = 1) touches merely call out new operations. Confirming (KOMRSP = 10) touches force users to think carefully about potentially disastrous operations. If, for example, the user calls for deleting an activity, GITPASE will highlight screen data relating to that activity and seek a confirming light pen touch. Only if the next touch is also on the "delete" menu item will delete processing continue. Otherwise, highlighting is removed and the delete aborted.

Confirming touches cannot be queued. Thus, a call for confirmation deletes any remaining touch queue hits.

F.3.1 Touch Queue Variables

To provide for a touch queue, add to common /IBMPC/ new variables

KUEXXX(25) = the x-coordinate of the queued touch

KUEYYY(25) = the y-coordinate of the queued touch

LIMKUE = the dimension of KUEXXX and KUEYYY (set to 25 in INITDI).

Existing /CONTRL/ common variable KUENUM already records the number of active items in the touch queue.

F.3.2 Replacement Subroutine OUTPEN

Subroutine OUTPEN(IOPN) returns single touches. If IOPN = 0, the touch is a normal (KOMRSP=1) one. If IOPN > 0, the touch is a confirming (KOMRSP=10) type.

Processing of the IBMPC replacement OUTPEN should begin by setting KOMRSP according to IOPN. If KOMRSP = 10, cancel the touch queue by KUENUM = 0.

Main OUTPEN processing consists of either extracting the next touch from the queue if KUENUM > 0, or collecting a new queue and extracting the first entry when KUENUM = 0. To extract an entry from a queue set /CONTRL/ variables KOMXXX to KUEXXX(1), KOMYYY to

KUEYYY(1), move all remaining KUEXXX and KUEYYY values up one position and set KUENUM to KUENUM - 1.

To solicit a new pen queue follow very closely demonstration routine MOUSIN. Begin by calling PRMPT (2 or 3). Then loop to mark the present position with HALO's crosshair cursor. Process responses acording to mouse buttons. If the left button is touched, save the touch as the next (KUEXXX, KUEYYY), and exit queue processing. If the right button is touched, store as the next KUEXXX, KUEYYY the scaled coordinates of the touch, and (if KUENUM < LIMKUE) loop back for another touch.

Once queue processing is complete and (KOMXXX, KOMYYY) set, call PRMPT(1) and then call subroutine NALHIT to classify the hit. (Note: That routine should be dummied for testing to merely set KOMBTN = 12 and return). If NALHIT returns with /CONTRL/ variable KOMBTN = 4096 an error has been detected. Call ERROR(8), clear IERFLG to 0, and loop back for a new (KOMXXX, KOMYYY) from the queue. If no error is detected OUTPEN returns.

F.3.3 Replacement Subroutine CONFRM

Subroutine CONFRM(NBTN, NUMHLT, KOORDS, NWIN) processes confirming touches. A replacement routine should first call HLIT a total of NUMHLT times to highlight NUMHLT rectangles in window NWIN. Coordinates of the rectangles are contained in vector KOORDS (4 per rectangle).

The subroutine then calls OUTPEN(10) to solicit a confirming touch. If, upon return, KOMBTN = NBTN, confirmation has been received; merely return. Otherwise, confirmation was denied; cancel highlighting by calling DEHLIT on the same rectangles as above and return.

F.3.4 Program-Directed Purges

At a few points in the GITPASE code the program detects the need for a special touch queue cancel. This is accomplished by a call to OUTGR1, option 39. Modify Milestone B code to set KUENUM = 0 on this OUTGR1 call.

F.4 Test Program

To test this Milestone enhance the test program of Milestone B. The revised version should solicit on tester demand all forms of keyed input. It should also be capable of calling OUTPEN asnd CONFRM. In the case of OUTPEN, print to the IBM screen the coordinates (KOMXXX, KOMYYY) so that mouse processing can be verified.