DOCUMENT RESUME

ED 051 665	24	EM 009 043

AUTHOR	Clutterham, D. R.
TITLE	A Method for Evaluating Student Progress in Undergraduate Computer Science By Use of Automated Problem Sets. Final Report.
INSTITUTION	Florida Inst. of Tech., Melbourne.
SPONS AGENCY	Office of Education (DHEW), Washington, D.C. Bureau of Research.
BUREAU NO	BR-7-D-080
PUB DATE	31 Jan 70
GRANT	OEG-4-8-070080-0015-057
NOTE	121p.
EDRS PRICE	EDRS Price MF-$0.65 HC-$6.58
DESCRIPTORS	Automation, Computer Assisted Instruction, Computer Programs, *Computer Science Education, *Grading, *Instructional Aids, Programing, *Simulators, *Student Evaluation

ABSTRACT

As an instructional aid for beginning computer science courses, two systems are described which permit the automatic diagnosing and grading of student prepared problems. The first system, called SIM 610, is based on a simulator which performs by actually running student programs prepared for a simple tutorial computer used in the classroom. The simulator, which will run on any computer with even a limited FORTRAN IV capability, simulates a single address, six decimal machine with 15 basic instructions, nine index registers, and 1000 memory locations. It is capable of taking any problem and a solution prepared by the instructor and using that solution as a standard against which student problems and solutions are automatically compared and graded. The instructor can specify the weighting of factors he considers important in the grading. Diagnostic information is provided to the student on practice runs. A second system, called an Assembly Monitor, provides for the running of student machine language programs on any IBM 1130 computer. It provides a protection system against novice programers destroying resident programs and, in addition, supplies debugging aids and a grading system very much like that for SIM 610. (JY)

BR 7-D-080
PA 24
LM

A METHOD FOR EVALUATING STUDENT PROGRESS IN UNDERGRADUATE COMPUTER SCIENCE BY USE OF AUTOMATED PROBLEM SETS

Dr. D. R. Clutterham
Florida Institute of Technology
Melbourne, Florida 32901

FINAL REPORT
Project No. 7-D080
Grant No. OEG-4-8-070080-0015-057

A METHOD FOR EVALUATING STUDENT PROGRESS IN UNDERGRADUATE COMPUTER SCIENCE BY USE OF AUTOMATED PROBLEM SETS

Dr. D. R. Clutterham
Florida Institute of Technology
Melbourne, Florida 32901

January 31, 1970

U. S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE

Office of Education
Bureau of Research

FINAL REPORT

Project No. 7-D080
Grant No. OEG-4-8-070080-0015-057

A METHOD FOR EVALUATING STUDENT PROGRESS IN UNDERGRADUATE COMPUTER SCIENCE BY USE OF AUTOMATED PROBLEM SETS

Dr. D. R. Clutterham
Florida Institute of Technology
Melbourne, Florida 32901

January 31, 1970

The research reported herein was performed pursuant to a grant with the Office of Education, U.S. Department of Health, Education, and Welfare. Contractors undertaking such projects under Government sponsorship are encouraged to express freely their professional judgment in the conduct of the project. Points of view or opinions stated do not, therefore necessarily represent official Office of Education Position or policy.

U.S. DEPARTMENT OF
HEALTH, EDUCATION, AND WELFARE

Office of Education
Bureau of Research

CONTENTS

SUMMARY

As an instructional aid to beginning computer science courses, two systems are described which permit the automatic diagnosing and grading of student prepared problems. The first system is based on a simulator which performs by actually running programs prepared for a simple tutorial computer taught in the classroom. The simulator, which will run on any computer with even a limited FORTRAN IV capability, simulates a single address six decimal machine with 15 basic instructions, 9 index registers, and 1000 memory locations. Several problems which have been used in student classes are given; however, the strength in the system is that it is capable of taking any problem and it's solution, provided by the instructor, and using that solution as a standard against which student problems are automatically compared and graded. The instructor can also specify the weighting of factors he considers important in the grading. Diagnostic information is provided to the student on practice runs he makes. The system has been used over four quarters and over 600 students have run problems on the simulator.

A second system provides for the running of student machine language programs on any IBM 1130 computer. This system, called an Assembler Monitor, is programmed in IBM 1130 machine language itself, and will only run on that computer. It provides a protection system against novice programmers destroying resident programs in the machine and, in addition, supplies debugging aids and a grading system very much like that for the simulator.

INTRODUCTION

In the fall of 1965 Florida Institute of Technology introduced an undergraduate degree program in Computer Science. The year 1969 saw the first graduate of this program. In addition to the more than 150 sutdents majoring in Computer Science at Florida Institute of Technology all of the 500 freshmen each year are required to take an introductory course in Computer Science which includes programming. These students pursue degree programs in Electrical Engineering, Physics, Mathematics, and Space Technology.

The Computer Science curriculum at Florida Institute of Technology was designed to present the technology necessary for the undergraduate student to understand computers and their usage and to become a future specialist or generalist in the field. For the non Computer Science major the introductory course taken requires the student to learn programming through actual running of programs he has prepared. For some students this is the only formal training they will receive in programming, but it provides a sufficient basis for their own subsequent work. Others will take additional formal coursework.

Teaching of the quantities of persons taking the computer science introductory course has been a formidable problem for Florida Institute of Technology as well as at other schools in such an endeavor. Since qualified instructors are rather rare there is a natural tendency to load the good ones unmercifully in terms of the number of students they face. In such a situation the instructors find it difficult, if not impossible, to assign and evaluate a representative number of problems. Such is the motivation for a mechanized means of evaluating student problems. A mechanized system also provides for gathering and processing statistical data to assist the instructor in his subsequent problems assignments.

In the process of introducing the unititated to the use of electronic digital computers, and their programming in particular, a teacher or author is faced with an early decision on the specific computer he uses for illustration. He must either deal with an existing computer or develop an artificial one to demonstrate the characteristics he deems essential. Either approach has advantages and drawbacks.

If an existing computer is taken as the illustration, a dilemma is again faced; either to choose a large machine with an extensive and sophisticated instruction repertoire, or a smaller machine with non essential characteristics imposed on it by short word length. For either case, more complexity is required than is desired to present the rudimentary concepts. The advantage of being able to demonstrate those concepts discussed on an available computer is considerable, however.

Alternatively, if an artificial computer and its instruction repertoire are chosen as the illustrating medium, then a teaching tool can be developed exactly to the author's taste, and need only include essentials, or, it may be embellished as desired. However, the students or readers can never observe the joys of a successfully run program of their own design, or the realistic frustrations of trying to chase down a bug. The results may be like learning to drive an automobile by a correspondence course.

A compromise to the choice between a real and an artificial computer approach is to start from the idealized artificial machine and to simulate its behavior on a real computer. In this way, programs can actually be written for the artificial computer and run (via the real one).

Work done under this contract includes the development of artificial machine language and a simulator on which it runs, and an assembler monitor system which permits ready student access to the use of an actual machine language. The simulator computer is written in FORTRAN and can be used with any computer which has a FORTRAN compiler; the assembler monitor is for use on the IBM 1130 computer only with its machine language. The 1130 computer is in very common use in colleges and other schools and is the Florida Institute of Technology's computer.

The automated problem set undertaken for this contract employs an artificial machine language which is simulated in the universal FORTRAN language so that programs can be written and run to demonstrate the fundamentals of machine language programming. The simulator is designated the SIM 610 (for Simulator of six decimal digit machine). Six decimal digits permits reasonable length data and instruction words. Use of decimal numbers permit the learning of concepts without the added burden of unfamiliar binary numbers, and without numerical conversions which divorce input and output numbers from internal machine numbers and operations.

The machine language is represented in terms of an instruction set detailed in the report. The pseudo computer of the instruction set has a memory of exactly 1000 words, addressed from 000 to 999 decimal. It has nine index registers referenced by digits 1 to 9. It has a potential for 100 different instructions through instruction codes 00 to 99; however, only 15 of these are used. A computer word length of six decimal digits plus a sign bit (assumed + if not specified) accommodates both single address instruction and data. The 15 instructions fall into categories; data transfer, arithmetic, input/output and branching.

The Assembly Monitor System is designed to permit use of the actual machine language of the IBM-1130 by the student in a controlled environment. This environment permits evaluation of student problems and protects the system itself from being destroyed by student program faults. Since the actual IBM-1130 machine language is rather complicated to use by an apprentice this is considered a necessary feature when assigning students assembly or machine language programs. Such problems are not assigned in the first introductory course which employs the SIM 610 simulator.

METHODS

Initial Objectives

At the outset of the contract the intent with regard to an automated set of problems was the establishment of a continuously reusable set of machine language programming problems. These problems would be of graded sophistication and difficulty and span at least two successive quarters of student experience. An evaluation and grading program was to be developed concurrently which would permit "batch" running of student programs. This program was called the Florida Institute of Technology Student Program Operating Monitor (FITSPOM).

A second task described in the proposal was the preparation of a set of symbolic (assembly) language programs and a means of running these programs in batches and evaluating them also. The intent here was to modify the IBM 1130 Assembler operating under the IBM 1130 Disc Monitor Program, a system available at many schools and colleges.

Both the evaluation programs above were to have data collecting capability on the programs run and were to perform some statistical evaluations on the results. Also both would provide feedback to the student in the form of dumps of his program.

A set of more than 60 machine language problems were developed with optimally programmed solutions and a subset of about 20 of these were picked as a set to be used in the programming courses. The problems were actually used with some of the student classes during initial work on the evaluation program and before it was ready.

A number of unanticipated difficulties arose which necessitated some revisions in the initial objectives. There are described in the following paragraphs.

A major curriculum revision occurred at Florida Institute of Technology affecting all departments and going into effect with the September 1968 term which was in the middle of the period of this grant. In this revised curriculum the courses taken by all students during the first two years are identical and it is not until the Junior year that the differences in the degree programs appear. Such a curriculum has both advantages and disadvantages for both the school and the student. From the standpoint of this grant the advantage is that not only Computer Science students, but all students at the Institute take an introductory computer course. The disadvantage, from the grant standpoint, is that where the automated problem set was to cover a sequence of courses, it must now cover only a one quarter course and the quantity of problems which can be treated is necessarily fewer. This change did make the requirement for a mechanized handling of student programming problems mandatory for Florida Institute of Technology.

One difficulty which might have been anticipated, but was not originally, was that when the same problems are given to subsequent classes, the optimal solutions also pass along between the students. Thus, any finite set of problems will soon have a complete set of perfect solutions available within the student body so that any student who would rather copy a program than write his own finds no difficulty in doing this. This becomes particularly acute when the course is a mandatory one for all students and does not include just the voluntary Computer Science majors.

With the introductory programming course limited to one quarter its contents had to be very carefully evaluated so that it could best serve the needs of all students - both those Computer Science majors and the larger body some of which would not have any further formal programming. As a result it was deemed necessary to include a higher level language in the course and FORTRAN was chosen. The result is that only about half of the course is devoted to machine/symbolic language. Moreover, the machine language had to be a particularly simple one.

Student problems would really have to be prechecked before running on either the machine language or the symbolic language evaluator because they could fail to run to a finish or worse yet could destroy the evaluator or other resident programs in the computer.

New Direction to Program

As a consequence of the difficulties described, several changes occurred. A very simple machine/symbolic language was developed for an artificial but representative computer. Addressing was done in decimal rather than binary so that concepts could be taught without the additional burden of simulateneous familiarization with another number system. Memory was limited to 1000 words.

The SIM 610 program described in this report simulates this artificial six decimal digit computer in that programs in the artificial language are executed as if the computer was real.

Instead of a formalized set of fixed problems, the approach taken was that any problem (prepared by the instructor or an advanced student, for example) could be used as a master, and the students problems would be graded against that as a standard. Thus there is no final formal set of problems; the student problems are simply made up by each instructor for each course as he needs them. Moreover, it is not assumed that the instructor's program solution is , optimal, and it is quite possible for a student grade to be higher than that of the standard provided by the instructor. Flexibility is provided for the instructor to place weighting factors on the various points to be considered in grading, changing them from problem to problem or even at different times for the same problem, depending upon where he wishes emphasis placed. For example, if he is emphasizing program running speed, a high weight can be given in the grade for fast running time as actually measured in terms of actual operations used and their execution times.

The SIM-610 simulator has been used for four quarters and with over 600 students. Surveys of student, instructor, and machine operator observations are included in this report. The Assembler Monitor System has been in informal use and aids in the writing of assembly language programs. The grading portion of the Assembler Monitor System has not been completely debugged, but since it has not had to serve large numbers of students this has not proved a problem.

RESULTS

SIM 610 SIMULATOR SYSTEM
Philosophy for Automatic Grading of Student Programs

In order to grade a student's program, it is necessary to determine its operating characteristics, (i.e. what it does). It is not possible to determine what a program does except by going through it step by step, except in specialized cases. This means either running or simulating the student program. Although it would theoretically be possible to determine other factors about a students' program not determinable simply by running or simulating it, the process involved would be too complex and time consuming to be practical.

There is one major objection to this method, however. If the student programmer makes a minor but crucial mistake anywhere in his program, his grade could be reduced to zero, even though the major part of his program works. This can be handled, however, by giving the student programmer enough debugging aids to allow him to debug his program and re-run it for a better grade. It should be noted that in practice, a computer program, no matter what methods used or how skillfully written, is worth nothing if it does not work. (We will take up the question of partially finished programs again later).

It is, therefore, necessary in order to grade a student program, to actually run it either through simulation or by allowing the execution of the instructions of the program.

If the student program is to be graded, however, the grading program must eventually regain control from the student program. This is no problem if the student's program functions properly and exits normally when finished doing the job. However, if the student's program contains an infinite (unending) loop, the grading program must be able to abort the student program and tell the student the reason for aborting. This can be best done by aborting the student program after a certain amount of run time or after a certain number of instructions have been executed (whichever is more conviently available on the system). The maximum amount of time thus set, must be large enough to allow even the inefficient student's program to complete execution; yet not allow the computer to be tied up an excessive amount of time on programs containing infinite loops. As a backup to this, it is sometimes useful to allow the operator to tell the grading program to take control. The specified method or combination of methods must be matched to the computer being used.

It should also be noted that this same instruction count or runtime can be used later in grading the student program (see below).

It is necessary, therefore, to gain control after the student program is through executing, even if it has an infinite (endless) loop.

When the grading program has gained control, it must determine whether or not the student program has done the job assigned. In some manner the grading program must be told which problem the student is doing. It must also have been given before the student program was run, enough information to determine whether the student did the problem properly.

In order to prevent cheating, all problems should be designed so that the output is a function of the input. For example, a problem to sum the first 100 integers is not a function of an input parameter. Specificlly, the answer is a constant, 5050. The problem can be made suitable if the sum of the first "N" integers is required, where "N" is input to the student program. So long as the student does not know what value "N" will be when his program is finally graded, he must do the problem correctly in order to be assured of the correct answer.

In order to be sure that the student will not be able to cheat in this manner, the input data should be changed from practice runs before the final run of the student programs when the grades are recorded for the instructor.

In order to do the above functions, the grader must be able to feed input data to the student program. It must also have the proper answers to the problem based on this input data. The grader must also be told if some of the answers are more important than others.

What, then, should the grader do if the student programmer gets only part of the right answers? Partial credit can be given for some of the answers correct, the answers in the wrong order, or in the wrong places without too much difficulty. It should be remembered, however, that if the students are given sufficient opportunity to debug their programs, there will be little need for the grading routine to have these capabilities.

It is necessary, therefore, for the grading routine to calculate whether or not the student's program did the job required on the basis of his answers being correct for the given input.

Since most students will complete a program that does the job correctly, the studonts grade must be based upon other factors in addition to the amount of the job completed. The best factors are those actually used to judge practical programs in industry: Runtime (or number of instructions executed in the student program if more easily available), and program length (ie. amount of storage space used by the student program). In addition, if the student program ended for some reason other than normal exit (ie. invalid instruction executed, excesive runtime, or other reason), then credit should be taken off.

The following formula is implemented as a weighting function to calculate the student's raw grade.

$$G = J \times E \times (a / R + b / L + c)$$

where

G = Raw grade to be computed;

J = A factor whose value is zero if no indication was found of the job being done, and is maxium if the job was done completly correctly by the student program;

E = A factor whose value is maxium if the student program ended in normal exit;

R = Runtime (number of instructions executed);

L = Length of the program in core; and a, b, c are positive "weighting" constants for the given problem.

One method of establishing "a", "b", and "c" is to make "a" and "b" functions of the runtime and length (respectively) of a standard program, prepared by a proficient programmer that does the job correctly. This standard program can also be used to initially calculate the proper output from the given input for use by the grader. The constant "c" provides a basis for a non vanishing grade even in the event of vanishingly snall credit for runtime, R, and length, L.

Finally, this raw grade must be curved against that of the other students doing the same problem. It is our experience that the raw grade curve can vary widely from one problem to another. Therefore only if the student's raw grade is compared to that of thers doing the same problem can his grade be curved properly. All student programs must be run for a grade before any can be given a grade in familar letter (A,B,C,D, or F) or percent (100% to 0%) form. The raw grade (based only on the standard program for the problem) can be given each time the student program is run; even for debugging.

The grading program calculates the student's grade on the basis of whether or not he did the job, the number of instructions executed (or the runtime, if available), the length of the program (how much space it uses in core), and how well his program did relative to the other students doing the same job. Moreover, the grade can be weighted by the instructor depending upon where he has placed emphasis in the programming assignment.

Finally, it is necessary to output the information thus determined by the grader. The student is given as much information as necessary. This includes a program listing, reason for exit, runtime, length in core, and whether or not the program has completed the job successfully. In addition, debugging aids such as tracing all or part of the students program as it executes are included. When the programs are run for the final grade, information is supplied to the instructor so that the grades can be curved and recorded.

The SIM 610 Computer

The SIM 610 is an artificial machine, simulated in the FORTRAN language, which will permit the student to program in machine language, and run as if his program were performing on an actual machine. The simulated computer has a word length of 6 decimal digits plus sign. When words are used for instructions, they are broken into three fields. The first two digits are the operation code, the next digit refers to any one of nine index registers, and the final three digits permit addressing any one of 1,000 addresses. Registers and data flow in the SIM 610 computer, are shown in Figure 1. Following Figure 1, let us trace the operation required for the execution of a single instruction. The instruction address register will contain the address of the next instruction to be executed. Making the assumption that the tag register reads 0 (that is that none of the index registers are referenced) the address from the instruction register passes through the adder with nothing added to it and enters the memory address register. This results in the selected memory contents being placed in the memory data register, and from here it is transferred to the instruction register. While in the instruction register, the first two digits identifying the operation go to operation control to be decoded into the actual operation to be performed. The tag digit goes to the tag select switch. Here one of the index registers is identified if the tag digit is between one and nine. Finally the address is transmitted back to the memory address register through the three digit adder at which time the contents of one of the index registers may be added if it had been previously identified. The number now in the memory address register identifies the location of data in memory and this data is then brought into the memory data register. From the memory data register, the data may pass either to the input-output control, or to the transfer added and accumulator. If the operation is a print, the contents of the memory data register will actually be printed on the output print device of the real computer. If a data transfer operation is involved, such as a load acculululator, the data will pass through the transfer added into the accumulator. If an arithmetic operation is involved, such as subtract from accumulator, or add to index register, the transfer adder will pass the data in the proper direction. Arithmetic operations may cause either the sign latch or the overflow latch to be set. The subsequent use of these latch indicators is described in Appendix I where each of the commands is detailed.

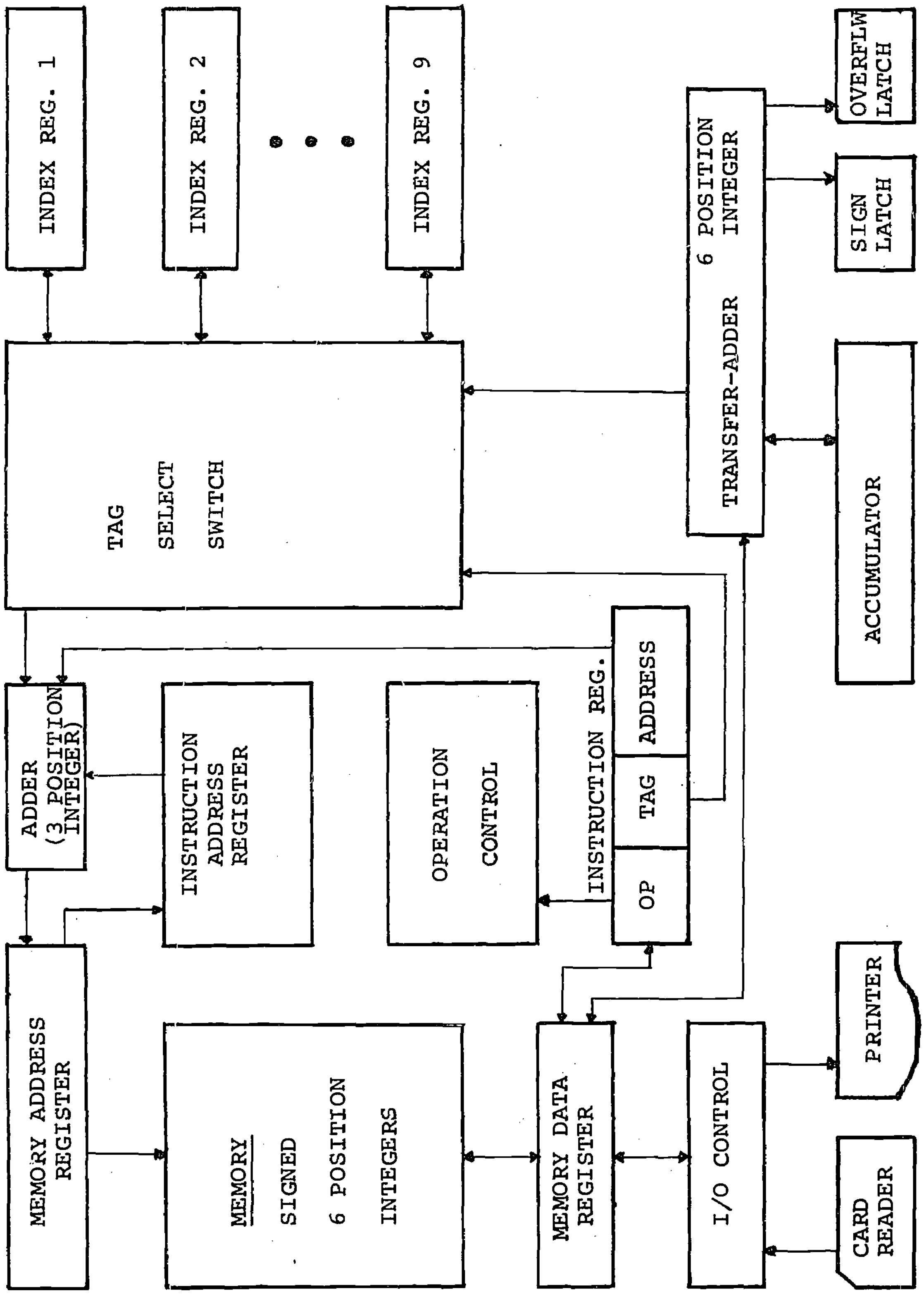
INDEX REG. 1
INDEX REG. 2
INDEX REG. 9
TAG
SELECT
SWITCH
6 POSITION
INTEGER
TRANSFER-ADDER
OVERFLW
LATCH
SIGN
LATCH
ACCUMULATOR
ADDER
(3 POSITION
INTEGER)
INSTRUCTION
ADDRESS
REGISTER
OPERATION
CONTROL
INSTRUCTION REG.
OP
TAG
ADDRESS
MEMORY ADDRESS
REGISTER
MEMORY
SIGNED
6 POSITION
INTEGERS
MEMORY DATA
REGISTER
I/O CONTROL
PRINTER
CARD
READER

Fig. 1

Problem Definition to the Student

Each problem included in the automated problem set which students must program, must be defined to the student and to the computer simulation program so that the desired automatic evaluation can be achieved. In addition to the fundamentals of the definition, a properly solved problem must be supplied to the computer. This solution must meet all of the specifications of the problem and should also be well programmed; that is, it should be optimum with respect to those characteristics where optimum is specified and should be near optimum in other respects. Thus, the solution should be prepared by the instructor or an otherwise well qualified programmer. This solution is called the "standard program" and all student programs are evaluated with respect to it. Nothing precludes a student bettering one or more of the parameters of the "standard program" and thus receiving a better relative score than the standard.

Characteristics which must be specified in each problem definition, provided they are appropriate are listed below:

Read: How much data must be called into the simulated computer by the program? Example: Read one card containing a number N which is the order of a polynomial whose coefficients are on subsequent cards. (A total of N+2 data cards are required: 1 containing the number N and N+1 containing the coefficients).

Store: Where are results or intermediate results to be located? Example: Calculate f(x), f'(x) and f"(x) and place them in locations 200, 201 and 500 respectively.

Output: What data is to be printed and in what order? Example: Print N (a problem parameter) and the contents of locations 100 and 101.

Statement: A statement of the problem to be solved. Examples: (1) Read in 50 items of data and add them. (2) Print out the squares of the integers from 6 to 20 inclusive. (3) Read in N numbers and sort them in increasing order of magnitude. Print out the sorted list.

Problem number: A two decimal digit number identifying the number of a problem set.

Appendix III contains some of the problems which have been assigned and solved by student classes.

Student Programs

Each student program is submitted as a deck of punched cards as follows: the first physical card in the deck is a beginning of program card, the next cards are the program proper. These are followed by an end of program card and finally by any data cards required. Format of the cards is as follows:

Beginning of program card

Column 1	* (asterisk)
Columns 2-7	000001
Column 8	1 if a deck listing is desired 0 if a deck listing is not desired
Columns 9-13	five digit student number
Column 14-15	two digit problem number
Column 16	(blank)
Columns 17-51	students name (LAST FIRST)

Program card

Column 1	+ or - (blank is treated as +)
Columns 2 and 3	operation code (see Appendix I)
Column 4	tag digit (0 if no index desired) (1-9 for index register)
Columns 5,6,7	three digit address (000-999)

End of program card

Column 1	* (asterisk)
Columns 2-7	999999

Data cards

Column 1	+ or - (blank is treated as +)
Columns 2-7	six digit integer (leading zeros if necessary).

When running programs for practice and debugging, the student should supply his own data deck following the end of data card and use an illegal problem number (e.g. 00). The data he supplies is strictly for his own use, and to satisfy himself that his program is working. If the student has supplied more data cards than required, and the program finishes before using all of them, SIM simply ignores the subsequent cards as it looks for the next students beginning of program card and starts on the next program. If the student has supplied fewer data cards than required and the attempt to read another card brings out the next students beginning of program card, then the present program is terminated and the next one begun. When a program is run for credit, data cards are not supplied by the student and instead "standard" or test data is supplied by the system from disk file storage just as if it were actual cards being read on command.

The first output command executed by a student program starts a new page of printing and prints one word of data from its effective address. Execution of each subsequent output command causes one item of data to be printed on a fresh line. If the trace program is in effect, the output will be intermixed with the trace, but still on a separate line.

Each run of a student program is provided with a trace of the first 25 instructions executed. Trace information (figure 2) includes on one line, the following information:

XEQNO - the number of the instruction just executed (1-25)
ADDR - the decimal address of the instruction just executed.
C(ADDR) - contents of the address above (i.e. the instruction just completed.
MNEMONIC - monemonic instruction including tag and decimal address.
C(XR) - contents of index register referenced (before)
EA - effective address in instruction
C(EA) - contents of effective address (before)
C(ACC) - contents of accumulator (before)
C(XR) - contents of index register referenced (after)
C(EA) - contents of effective address (after)
SIGN - sign latch setting
OVFL - overflow latch setting

Another helpful output from a student's program run is the memory dump. This dump consists of up to 100 lines of printout, each line containing ten words (sign plus six decimal digits). Each line is headed by a decimal identifier indicating the first word of the 10 word block it contains. No blocks (lines) are printed if at least one word in the line was not changed by either writing or executing the program. Unchanged words are left blank in a line. Thus, a few lines of printout may suffice to show everything that changed in a short program. In addition (in fact prior to) the memory dump, the contents of all index registers are printed sequentially on one line. Those which were unused are again left as blank in the printout.

Additional comments which may assist the student in debugging, are provided with the trace and dump and include one of the following;

EXECUTION COMPLETE
PROGRAM TERMINATED DUE TO EXCESSIVE RUN TIME
INVALID INSTRUCTION ENCOUNTERED AT ----
EXECUTION TERMINATED BY INSTRUCTION AT ---- ATTEMPTING TO READ 1ST CARD OF NEXT PROGRAM INTO ----.

Finally, scoring information is included with calculated scores. On a grading run, the standard program weighted score is shown, otherwise it is zero.

Figure 2 is a SIM 610 diagnostic printout for the student as described in this section.

STANDARD　　　　　　　　　　　　　　　　　　PROBLEM NO. 12

EXECUTION COMPLETE

RUNTIME		LENGTH OF DECK		NO OF CARDS READ		NO OF ANSWERS WRITTEN	
YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD
1246	1246	38	38	16	16	15	15

POINTS RECEIVED FOR---

READING DATA		ANS IN CORR LOCATIONS		WRITING ANSWERS		TOTAL		RAW
YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD	YOURS	STANDARD	GRADE
250	0	435	0	360	0	1045	0	32767

SIGN 0　OVERFLOW 0　ACCUMULATOR -999910

	I/RS	000000	000015	000000						
0	600040	401035	402040	601200	101200	111300	421036	422037	530003	411900
10	403900	103299	421037	520017	211299	510022	500011	423037	520029	413900
20	401900	500011	103299	110400	101299	113299	100400	111299	500011	401040
30	612300	422036	421037	530030	770000	000000	000001	-000001		
40	000015									
200	000456	499999	-999910	888889	-000001	000000	120450	000000	-112000	100001
210	000008	102250	-000005	000300	-000060					
300	-999910	-112000	-000060	-000005	-000001	000000	000000	000008	000300	000456
310	100001	102250	120450	499999	888889					
400	-999910									
900	000001									

Initialization of a SIM 610 Problem

The "initialize grader" program (INITG) accepts a set of ten (10) cards containing parameters of the problem to be run, and together with other systems programs "load program" (LOADP) and "dump grader" (DUMPG) and "auxiliary initialization program" (INI2G) provides the problem description to the simulator. These ten cards and their content and function are described in the following paragraphs.

Card 0: Character set card

Columns 1 - 16: The integers and operation symbols 0123456789-b+&*b where b designates a blank.

Column 17: Data Set Code (an integer from 1 to 6 inclusive).

The character set identifies the permissible character set and the data set designates a pair of records to be read from "Simulator data" (SIMDT) into DATA1 and DATA2 for use when the standard program executes a read card instruction.

Column 19: Final Grading Indicator. Set to 1 if the points and calculated grade of a student program are to be stored in SMSTU. Not used during initialization.

Cards 1 through 9 are the program description and all have the same format - 10 fields of six place integers, starting in column 1 and having two blanks between fields.

Card 1: Problem number
Field 1: Problem number. This is the record number in the "File of Standard Grades" (FSTDG).

Card 2: Read Groups
Consecutive Fields: Number of cards required in each group (NRDSR) for a number of groups up to 1 and including ten.

Card 3: Read Group Start
Consecutive Fields: The location of the first card in each read group corresponding to card 2 (LOCRD).

Card 4: Store Answers

Fields 1 - 5: Each field gives the first of a sequence of consecutive locations in which the student program is to store answers (LCANS).

Fields 6 - 10: The length (number of answers) of each of the sequences starting in the respective LCANS locations above (NANSR).

Card 5: Points Credit

Consecutive Fields: Each field stores the number of grade points credit to be given for correct answers (data matching the standard problem) for the read groups and their starting locations as given in the respective fields on cards 2 and 3. (PTCR)

Card 6: Proper answer location

Fields 1 - 5: Each field gives the number of points for placing computed answers in the proper locations (regardless of their correctness) as credit for satisfying this part of the problem specification. Proper locations are specified by the corresponding fields 1 - 5 and 6 - 10 on card 4. (PTCA)

Fields 6 - 10: not used.

Card 7: Correct Answers

Fields 1 - 5: Each field contains the number of points to be given for each correct answer found in the locations identified by card 4. (PTCC)

Fields 6 - 10: not used.

Card 8: Printed answer locations

Fields 1 - 5: Each field contains the number of points to be given if the correct answers are found stored in the appropriate group for printing (even if not printed in the correct sequence). (PTCW)

Card 9:

Field 1: Number of points credit if student program execute same number of card read instructions as standard program. Locations where the data read is placed is not considered here. (PTCKN)

Field 2: Number of points credit for obtaining each correct answer but storing it in an incorrect location (although within total area designated for answer storage). If an essential ingredient of the problem is intended to be sequencing or placement of results then credit points should be set to zero. (PTCO)

Field 3: Number of points credit for obtaining correct result for output but storing it in an incorrect location (although within the total area designated for output data storage). (PTWO)

Field 4: Number of answers written by standard problem. This number appears on student's dump but is not given any point value by the system. (NANS)

Field 5: The contents of this field gives the starting point within the data file for the problem under execution for the reading of simulated data cards as called for by the student (or standard) program. (FDATA)

Field 6: The number in this field establishes a maximum on the number of operations executed by a given student program. If this many steps are executed, it is assumed that the program is in a loop or is otherwise excessive in its running time and the program will be terminated. (MAXRT)

Field 7: Percent of grade for run time (steps executed for solution). (PCGRT)

Field 8: Percent of grade for program length (length of student deck). (PCGPL)

Fields 9 - 10: not used.

Figure 3 (a and b) illustrate an actual set of cards from a problem set. This may be correlated with problem 3 in Appendix II.

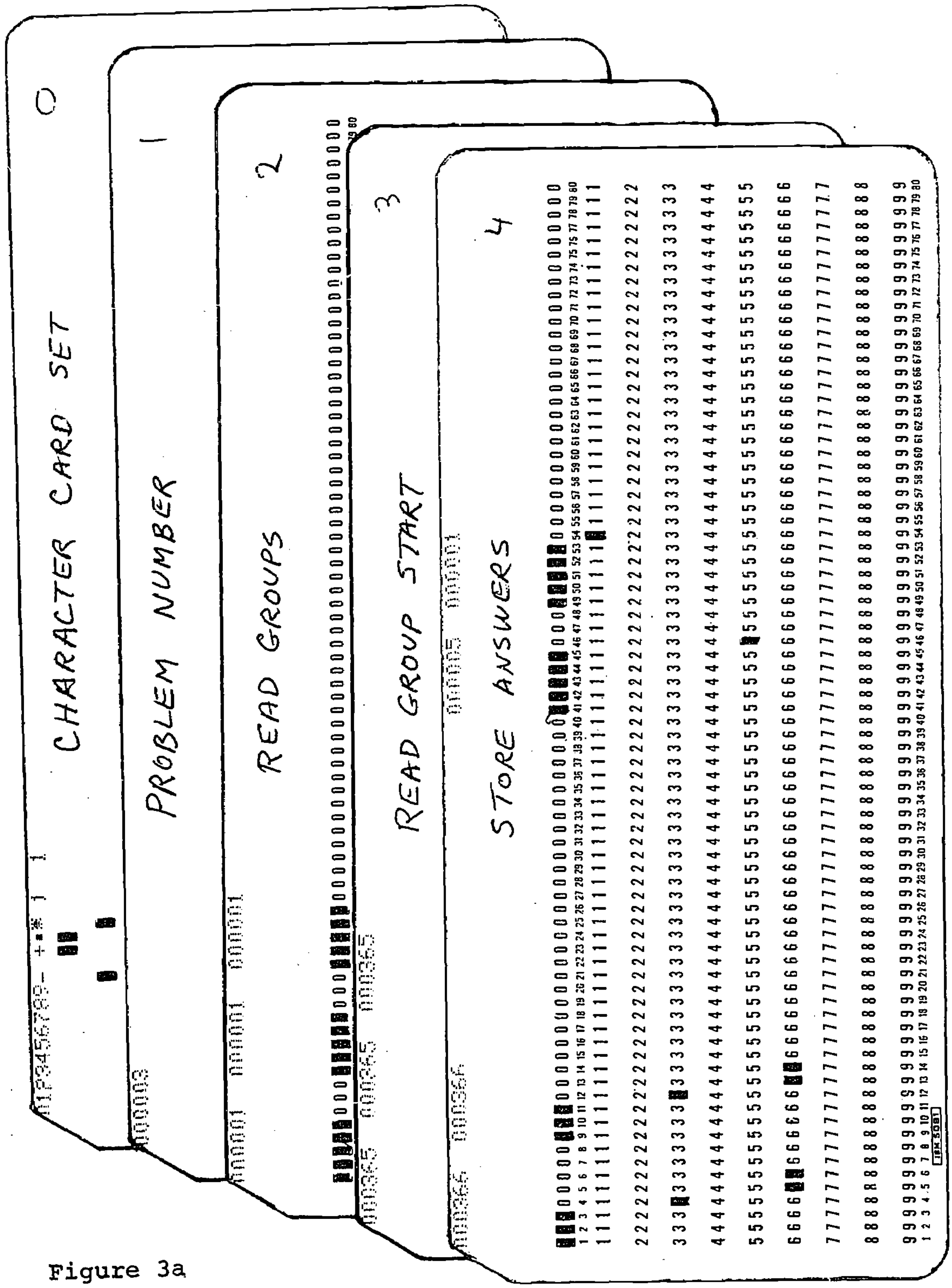
0
CHARACTER CARD SET
1
PROBLEM NUMBER
2
READ GROUPS
3
READ GROUP START
4
STORE ANSWERS

Figure 3a

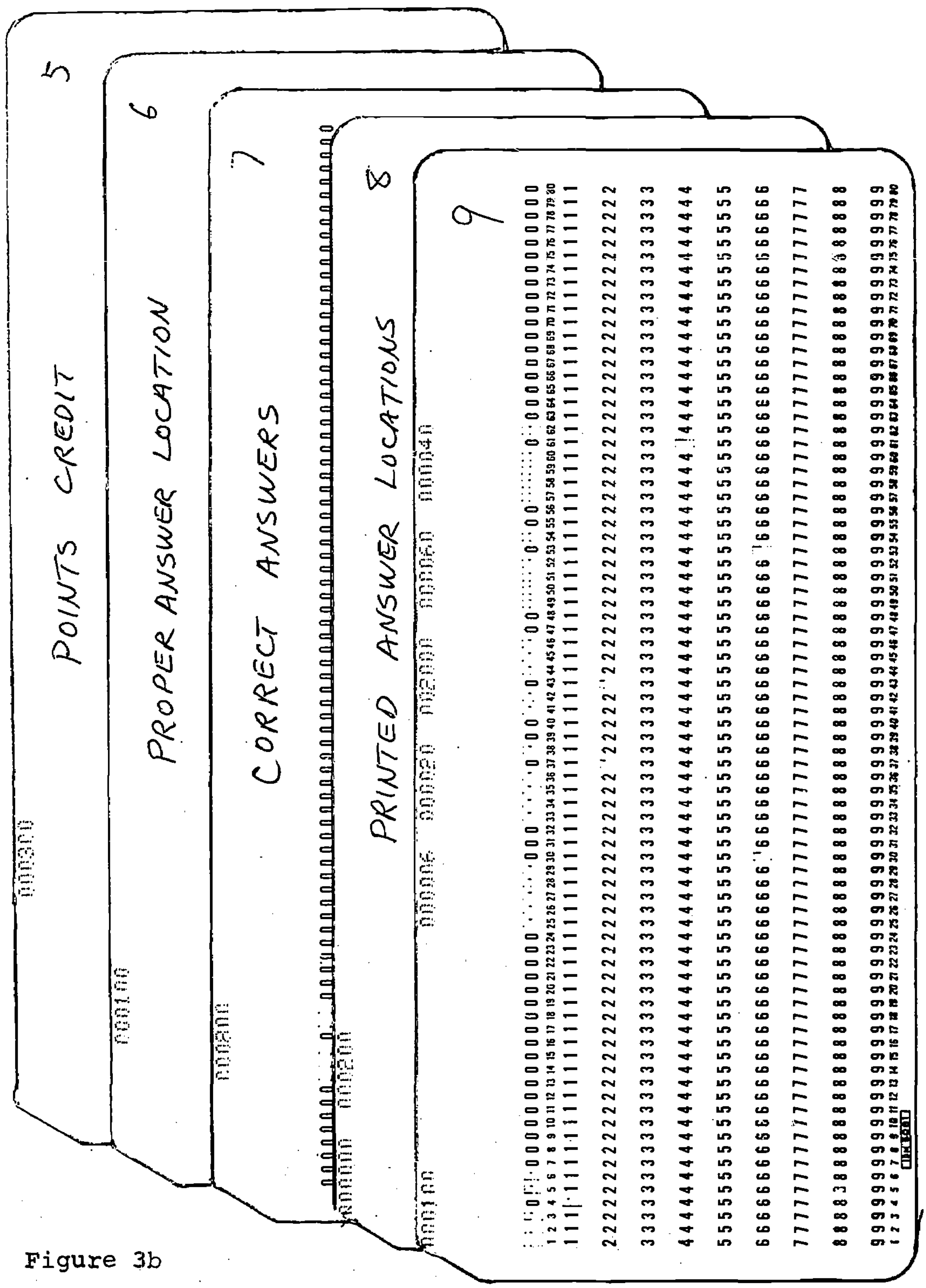
5
POINTS CREDIT
6
PROPER ANSWER LOCATION
7
CORRECT ANSWERS
8
PRINTED ANSWER LOCATIONS
9

Figure 3b

Operation of SIM 610

Three files must be defined in order to prepare the SIM 610 program to run a batch of student programs. These files are:

1) SIMDT This file consists of six pairs of records of 106 words each pair, and contains any simulated data required for problems to be run.
2) FSTGD This file consists of 24 records of 160 words each. Each record is associated with one problem; thus 24 different problems may be evaluated in one batch.
3) SMSTU This file consists of 800 records of 40 words each. Each record is associated with one student; thus 800 student's programs may be evaluated in one batch (assuming each student has only one program.)

An initialization is required at the outset of a batch run in order to: a) assure that grades associated with any problem numbers undefined for the current batch give a zero grade (if not initialized, a meaningless result could occur when an undefined problem number was attempted) and b) set the "pointer" in the first record of SMSTU to the first student record (as each student deck is processed, data on his program are placed in the SMSTU at the next available position. The pointer keeps track of the next available position so that batching of student problems for grading may actually take place over more than one time on the computer.)

Loading of data into one of the records of the SIMDT is done by the INDFG subprogram. A character set card, with the symbols

0123456789-b+&*b (b is a blank space)

in columns 1 through 16 and a digit 1 through 5 in column 17 to designate which pair of records is to be loaded, must precede the data cards. This is followed by up to 106 pairs of data cards which will be entered into the designated records.

Now the SIM system is ready to initialize or run problems. For initialization, INITG is executed, and reads for each problem a character set card, nine problem definition cards, a data set of 106 cards if column 17 of the character set card was zero (did not indicate one of the six prestored DATA sets in SIMDT), and a standard program including beginning and end of program cards. Further details are given under Initialization. For running a series of students programs, STRTG is executed, which requires only one character definition card, followed by a DATA set deck if the character card so indicates, and then the student programs stacked one after another. Normally students are given some time to debug their programs, and the results may not be desired to be recorded beyond the listing - dump which is given to the student. This will occur if anything but a 1 is in column 19 of the character set card. When the deadline for finished student programs has passed, STRTG is run using a 1 in column 19, and the student's student number, name, points received (3 categories) and raw grade are stored on a record of SMSTU for each program, except those with mispunched cards (such as a number in column 1), which are not executed or dumped.

Whereas initialization may be terminated at the end of the present program by turning off sense switch 2, no provision is presently made for exiting form SIM 610 in normal operation (under STRTG), since runs are generally of long duration and abnormally terminating a FORTRAN program is simple with most computers.

After a class or group of students programs have been run for grading, the file SMSTU should be dumped to cards for reduction to instructor-useable forms. The program DMPFG accomplished this, and also makes a listing. This gives the programs in the order run, and is useful for finding decks or listings (if not yet returned/given to instructor) or identifying mispunched programs, which are not run, and are in the deck but not on the list. The deck is used in conjunction with a simple listing program and a card sorter, as described below.

The cards may now be sorted in ascending raw grade order and separated by problem, giving a list useful for marking grade divisions; they may be resorted in alphabetical order or student number order for instructor's convenience. An advantage over on-disk sorts arises if correction is desired of cards which do not have last name first, or have other obtainable data missing. If more than one class is represented, the cards may be sorted on the field(s) chosen to distinguish classes, and each class deck listed in various sequences to the taste of the professor. In fact, the separate decks may be given to each instructor to cross index as he wills. Note that the original file is still available until INTFG is used to clear it. (Caution: if same problems are to be reused, references to the file FSTDG should be removed from INTFG, or else this will also be cleared; however, it is normally desired to change to a new data set both before and after grading, thus requiring reinitialization of the problems anyway).

ASSEMBLER MONITOR SYSTEM

The Assembler Monitor System differs from the SIM 610 system in a number of ways. First of all, the Assembler Monitor System uses an actual computer language -- that of the IBM 1130, a small general purpose computer, and thus can only be used on an IBM 1130. As was discussed before, the SIM 610 system can be used on any computer that has a FORTRAN compiler.

There are some advantages to the Assembler Monitor System (AMS), however, Unlike SIM, AMS can use subroutines, including all subroutines available for the system. This also means that AMS allows more flexibility in input/output and allows for problems of much greater complexity.

Two further uses for AMS were found during development. Like most small computers, the IBM 1130 has no memory protect hardware, and no available software to provide this feature. Therefore, we developed as part of AMS a software memory protect to prevent the student from accidentally destroying the Assembler Monitor itself, or the core-resident portion of the IBM supplied monitor-supervisor system. This feature of AMS has proved useful in itself as a debugging aid for the IBM 1130, especially for hard-to-debug assembler language programs and subroutines. Secondly, it was found that AMS could monitor FORTRAN programs on the IBM 1130 just as easily as assembler language programs, thus opening the way to additional uses for the system.

This memory protect software, a necessary part of the Assembler Monitor System, is an extremely complex system in itself. It comprises most of the AM program, which consists of more than 1000 cards . It is written in the assembler language of the IBM 1130.

The portion of this report on the Assembler Monitor System is presented in the form of descriptive handouts to those using the system, and has worked quite effectively. Each of the subsequent sections is such a handout.

Calling the Assembler Monitor System

In order to put your program under the control of the Assembler Monitor System, it is only necessary to call AM from your program, giving it the problem number and your student number. However, there are two pitfalls that must be avoided:

1. Your call to AM must be physically the first CALL or LIBF in your mainline program.

2. Your call to AM should be the first executed statement in your mainline program. Should any instructions be executed before the call to AM, they will not be under control of the Assembler Monitor System.

Your input and output are in COMMON, located at the very end of core. In order to set aside this space at the end of core, you must use a COMMON statement.

You should not attempt to call AM more than once in any given program. An attempt to do so will result in the Assembler Monitor System suppressing further execution.

Below is a sample program including calling sequence for the Assembler Monitor System. (Numbers in next line are card column positions.)

```
1                  21      27      35      42

//JOB
//ASM
*COMMON 48                      (Note: 48 is a sample number, only)
*LIST                           (Note: optional)

                                               Program, constants,
                                               etc.; not including
                                               CALL or LIBF
                                               statements.

                   START CALL    AM
                         DC      PRNO
                         DC      STNO
                                               Program, constants,
                                               calls, libfs, etc.

                   STNO  DC      417           Your student number
                   PRNO  DC      4             Problem number

                                               Program, constants,
                                               calls, libfs, etc.

                         END     START         Last card
```

Before each machine language instruction is executed, AM tests the instruction to determine if its execution would alter the core resident monitor, alter AM itself, make an invalid entry to a subroutine, or an invalid alteration of a subroutine. If its execution would have one of these undesireable effects, further execution of the student's program is suppressed and a link is made to DBUG as explained elsewhere. Further execution of the program is also suppressed if a valid exit is reached, or the run time becomes excessive.

If, on the other hand, AM decides that the instruction should be allowed to execute, the instruction counter is incremented and control is passed to the instruction that was tested.

Immediately after the execution of the instruction, control returns to AM by means of a hardware interrupt. This interrupt results from the machine being in interrupt run (also called trace) mode. AM then tests the next instruction, as before. This procedure of first testing each instruction and then allowing its execution is continued until further execution of the program is suppressed, as described above.

To cause the Assembler Monitor System to monitor your program, you need only call AM at the beginning of your program. When control is passed to AM, it reads the student's input data from the disk, initializes parameters to be used during execution to tell how core has been partitioned for the core load, forces the operator to place the machine in interrupt run mode, and gives control to the testing portion of AM so as to test the first instruction of the student's program.

Interpretation of Output

After the Assembler Monitor has decided that the student's program should not be allowed to execute further, control is passed to DBUG. DBUG moves the paper to the top of the page and prints on the right-hand side the student number, problem number, contents of the accumulator, extension, index, carry, and overflow registers and the floating accumulator. Student number and problem number are given as positive decimal numbers; the accumulator, extension, and index registers are given in hexadecimal; the carry and overflow are given as being "on" or "off"; and the floating accumulator is given in hexadecimal and decimal.

On the left-hand side is printed a core map which gives the starting addresses and lengths of eleven consecutive partitions that make up a core load. The lengths of these partitions vary according to the program(s) in the core load.

The first partition is the Index Register Area, which consists of the first four words of core (i.e., addresses 0, 1, 2, and 3). It is so called because it includes the three index registers, which are in words 1, 2, and 3 in core.

The second partition is the resident monitor, which includes the core resident monitor supplied by IBM (excluding the first four words of core) and the core image header which is located immediately thereafter.

The third partition is the mainline program, which includes everything from the end of the core image header to the beginning of the Assembler Monitor (AM).

The fourth partition is the AMS program, which consists of the program AM, and is the in-core part of the Assembler Monitor System.

The fifth partition is the subroutine area, which includes all subroutines, regardless of type, located between the AMS program and the interrupt level subroutine area.

The sixth partition is the interrupt level subroutine area, which includes all interrupt level subroutines except levels two and four.

The seventh partition is unused core. This partition of core is not used by the core load.

The eighth partition is the LIBF transfer vector, which consists of three words for each library function entry point in the core load.

The ninth partition is the floating accumulator, which consists of six words of core used as an accumulator for floating point arithmetic. There is no floating accumulator if there is no LIBF transfer vector.

The tenth partition of core is the CALL transfer vector, which consists of one word for each CALL entry point in the core load. The CALL transfer vector will sometimes include a dummy word in order to make the floating accumulator begin on an even core boundary.

The eleventh and last partition of core is COMMON, which is located at the very end of core. It is in this partition of core that the input and output occur. COMMON is saved between LINKs by the monitor system; i.e., it is still in core when DBUG and GROUT are loaded in turn.

On the left-hand edge the starting address and length of each partition are printed in hexadecimal. On the right-hand edge the word ADDR is printed beside that partition in which the effective address of the instruction causing the exit was located. If the exit was not caused by the effective address, the word PREA is printed beside the partition in which the last effective address formed was located.

DBUG then skips a space and prints the instruction causing the exit and the prior instruction in hexadecimal. To the left it prints the real address (the address of the instruction in core) and the loading address (the address of the instruction relative to the loading point of the mainline, which is the address found on a relocatable assembler mainline listing or a FORTRAN mainline listing).

If the program failed to clear location $IOCT (/0032 hexadecimal), a line is printed indicating this fact. This error would indicate that an interrupt service subroutine was not incrementing or decrementing $IOCT properly. Location $IOCT should be zero if and only if there are no I/O interrupts pending.

A line is then printed giving the reason why the student's program was prevented from further execution, i.e., the reason for exiting. This line is printed in the form:

AMS xx (message giving reason for exit) where xx is the error number. The error numbers are given in the following table:

00 Instruction is located in COMMON.
01 Instruction is located in CALL transfer vector.
02 Instruction is located in floating accumulator.
03 Instruction is located in LIBF transfer vector illegally.
04 Instruction is located in unused core.
05 Instruction is located in interrupt level subroutine.
06 Instruction is located in subroutine area illegally.
07 Instruction is located in AMS program.
09 Instruction is located in monitor illegally.
0A Instruction is located in index register area.
0C Attempt to alter CALL transfer vector.
0E Attempt to alter LIBF transfer vector.
10 Attempt to alter interrupt level subroutine.
11 Attempt to alter subroutine area from mainline.
12 Attempt to alter AMS program.
14 Attempt to alter resident monitor.
15 Attempt to alter word zero in core.
1A 64 instructions did irrelevent access of core.
1B Program terminated due to excessive run time.
1C Invalid instruction.
20 Valid exit.

Any other indicators indicate an error in the Assembler Monitor System, and should not occur.

Next, the message ADDRESSES OF LAST n INSTRUCTIONS EXECUTED is printed, where n is a decimal number with a maximum value of 64 giving the number of addresses listed thereafter. If the program ran for less than or equal to 64 program steps, all the addresses, in the order of execution, will be listed. If the program ran for more than 64 program steps, only the addresses of the last 64 are listed. Both the real and loading addresses are listed in hexadecimal.

If any instructions did an irrelevent access of core (i.e., they did no harm, but did no good, either), then the addresses of these instructions come out in a table titled ADDRESSES OF ____ INSTRUCTIONS LOADING IRRELEVENT DATA where ____ is a hexadecimal number. As above, each address is given both relative to the beginning of core ("REAL") and relative to the beginning of the core load ("LOAD").

In the event that the problem number is zero (or is not the number of a defined problem) no LINK is made to GROUT, the program is not graded, and the only other information printed is the program load length (both in hex and decimal) and the number of instructions executed (both in hex and decimal).

If the problem number is that of a defined problem, then a link is made to GROUT which outputs the student's grade and reasons behind it in three sections titled POINTS FOR CORRECT ANSWERS, ADDITIONAL POINTS FOR OUTPUT, and POINTS FOR PROGRAM EFFICIENCY. The total grade is the product of the total points for each of the three sections (divided by one million to scale it down). The total points for each section is printed after the word total at the bottom of each section and is equal to the sum of the points earned under that section as listed under the right-hand column. The points for each line are calculated from how well the student program did relative to the standard on this point. The total grade is printed beside the message TOTAL GRADE EQUALS at the bottom of the page. The total grade and each of the separate totals should range from zero to one thousand, although it is not impossible to make a grade greater than one thousand.

After printing the total grade, control is returned to the IBM supplied monitor supervisor, which begins looking for the next job.

Operator Procedure and Interpretation of Operator Console Displays

With student program decks in the card reader and the system initialized, the console typewriter will display the following message:

SET MODE SW TO INT RUN

At this time, the operator must set the mode switch (located on the right hand side of the display panel) to "interrupt run" and press the "program start" switch. If "program start" is pressed without first setting the "interrupt run" condition, the above message will be printed again. If the machine is already in the interrupt run mode, the message will not be printed. While in interrupt run mode, the "stop" button will have no effect.

The Assembler Monitor System has a provision for terminating a student due to excessive run time (based on a count of operations executed) and this is done automatically. However, an operator may abort a student program by momentarily placing the bit 11 switch on the console in the up position. In case this does not abort the program and cause an appropriate error message to be printed, then the program is not under Assembler Monitor System control.

If an abort is desired while the machine is in the interrupt run mode and not under control of the Assembler Monitor System, the operator must first take the machine out of interrupt run mode and then press "interrupt request." Alternately, he can first press "interrupt request" which will stop the computer, then change to the run mode and press "program start."

If bit switch 0 is up, the program will stop after each machine language instruction is executed under control of the Assembler Monitor System and display the contents of the Accumulator, Extension and Carry and Overflow status.

Bit switches 14 and 15 are used to control student core dumps and displays to the operator during student program execution under control of the Assembler Monitor System. If bit switch 14 is up and 15 is down, all relevant student core content will be dumped on the printer and the system will pass to the next student program. If bit switch 15 is up, the computer will pause and display a coded error number in the storage buffer register, the address of the instruction causing the exit in the accumulator, and the effective address of the last instruction employing an effective address in the extension register. Upon restarting, if bit switch 14 is also up, then the relevant student core data will be dumped on the printer. With neither switch 14 or 15 up, no pause or dump occurs.

An override feature is provided which may be used with caution: if bit switch 13 is up after a pause caused by a program exit and switch 15 being up then the Assembler Monitor System will return to the student program.

Programs, Subroutines and Files

Running of the student programs is done under the control of the Assembler Monitor System. This system consists of seven main computer programs, several standard subroutines and four data files described briefly below.

The Assembler Monitor Program (AM) serves as a direct monitor over the running of the student's program, with each instruction performed under monitor control. A debugging aid generator program (DBUGT) prints out a trace and other diagnostic aids to the student from information provided by the AM. The raw grade is calculated by a grading program (GROUT) which calculates the students grade, prints it and records it for the instructor. Program GRINT generates information on which the grade is based from the standard problem supplied. Program INITD initializes data for the grading of each student's problem. For the start of a grading run or for each new problem set, the system is reinitialized with program RINIT which clears the data and grade files. A message input program (MSGIN) loads file a message file with the appropriate messages to be used by the DBUGT program.

Subroutines used in the system include the IBM-supplied Commercial Subroutine Package-Version III, and assembler subroutine for floating binary to decimal (FBTD) and the following special subroutines: FORMT and SHIFT are used by DBUGT to decipher assembler instructions HEXIN converts four alphanumeric characters representing a hexadecimal number into the integer equivalent. HEX and HXOUT convert an integer back to hexadecimal. DCOUT converts an integer into five alpha characters representing a number in decimal. OUT prints a line of alphanumeric characters and clears the output buffer to blanks. DSCTR dumps a 320 word core sector (length of one disk sector) in hexadecimal to the printer.

SAVGR contains three records of 320 words per record. Since each disk sector contains 320 words, this file uses three sectors. The initial contents of SAVGR are unimportant because AM loads the file with new data with each new student program. The actual instructions, variables, and constants of AM are stored by that program in three blocks. The three records are the 320 words following respectively the three DSA statements labeled IOAR1, IOAR2, and IOAR3. It is the task of program DBUGT to extract the pertinent data from irrelevant coding. SAVGR is referred to in all programs by symbolic file number 1.

MSGBF also contains three records of 320 words per record, giving three disk sectors. It is used by program DBUGT to print all words interpreting the output of AM including all headings and in converting all numbers from integer format to alpha characters. To initialize MSGBF, program MSGIN is executed, reading data from twelve cards in FORMAT (80A1), and storing the contents on disk. Refer to program listing of MSGIN for contents of data cards. MSGBF is referred to in all programs by symbolic file number 2.

The records of GFILE each contain 16 words with one record generated per student program run under the system for grading. The length of GFILE can therefore be varied with the needs of the user by simply changing the number of sectors specified when the file is set up and by changing the number of records in the DEFINE FILE statement in program GROUT. For example, if the user desired 400 records at 20 words per sector, this would require 20 sectors of disk. The contents of each record of GFILE will be listed and explained later. The contents of GFILE is initially set to zeroes by program RINIT. GFILE is referred to by symbolic file number 3.

DATFT contains ten records of 320 words apiece, giving 10 disk sectors. Each record contains information used by the system in grading a problem of the standard data set. The system can therefore handle a problem set of 10 problems. The corresponding record of DATFT must be reset to zeroes before entering a new standard problem in the problem set. To reset DATFT and/or GFILE, execute program RINIT, following it by one data card of FORMAT (10I2,10x,I2). The first 10 fields indicate which records of DATFT are to be reinitialized. If GFILE is also to be 32 is to be left blank. DATFT will be referred to by symbolic file number 4.

To define these four files on disk, the computer should be given instructions corresponding to these:

```
// JOB
// DUP
*STOREDATA  WS FX SAVGR0003

*STOREDATA  WS FX MSGBF0003

*STOREDATA  WS FX GFILE0020

*STOREDATA  WS FX DATFT0010
```

Since programs DBUGT, GRINP, and GROUT are executed by links and have quite lengthy core-loads, the running of a student program under the Assembler Monitor System can be quite time-consuming. If the user has sufficient area on disk, it is suggested that these programs be stored Core-Image. This will considerably speed the operation of the system. All four data files must therefore be stored in Fixed Area on disk.

The next step is to execute program MSGIN which will read 12 cards of alphanumeric data and initialize file MSGBF (see program listing). This file will be used to generate headers and output information by program DBUGT.

Assembler Monitor Use

The Assembler Monitor System has provision for up to 120 words of input data read by the student program determining the grade on up to 120 words of output. The input is loaded by AM into COMMON, beginning with the last word of core. AM will not load input data beyond the end of a student's specified COMMON. Any COMMON beyond the number of words of input is filled with zero or some other easily recognizable "garbage word" specified by the instructor. This is done as a debugging aid so that the student can determine by examining a core dump what, if anything, his program has changed. The output must also be in COMMON and within the last 320 words of core. The 120 words of output can be divided into as many as 10 blocks of consecutive core locations and these blocks can be located anywhere within COMMON. This permits freedom to:

1. Give more important answers more credit for grade.

2. Count part of the grade on intermediate answers arrived at in the process of generating the final answers.

3. Remove points for destroying the input in the process of obtaining an answer. A further option is provided to give points for partially correct answers, that is answers either in the correct blocks but in incorrect order, or answers found anywhere within COMMON. This option can be used as a debugging aid by pointing out to the student that he has made only a small logic error in addressing and not written a program that does nothing.

Program efficiency is determined on the basis of five parameters: mainline program length, subroutine length, length of COMMON, number of instructions executed, and a standard curve or bias. The curve is based on the theory that with the high speed of this computer, the length of most programs run under the system, the difficulty of writing in assembler, and inexperience in programming of most students using the system, that a program that works should not receive a failing grade no matter how inefficient it is.

In order to initialize DATFT with the standard input data, output buffer locations, and grading factors the instructor must perform the following operations: First Store subroutines INITD, HEX and HEXIN on disk. HEXIN is used to translate core addresses entered in hexadecimal (four characters) into integer constants. HEX is used to translate DATFT to hexadecimal characters for dump to printer. INITD takes parameters problem number and standard input and data cards for output locations and grading points and puts them on disk. Since INITD is a subroutine, it cannot initialize its own IO. This must be done by a short calling program (written in FORTRAN). This program must initialize ISS routines for disk, card reader, and line printer and must tell INITD where to find DATFT on disk. For Example;

```
//JØB
//FØR
*ØNE WØRD INTEGERS
*EXTENDED PRECISIØN
*IØCS(DISK,CARD, 1403 PRINTER)
      .
      .
      .
   DEFINE FILE  4(10,320,U,K)
      .
      .
      .
      program (see below)
      .
      .
      .
   CALL INITD(...            )
   CALL EXIT
   END
//XEQ        01
*FILES(4,DATFT)
      .
      .
      .
      5 Data cards.
```

The following four <u>integer</u> calling arguments should be passed to INITD if called by FØRTRAN:

1. Problem Number(PROBN).

2. Standard Input (STDIP), the first element of an array up to 120 words long.

3. Standard Input Length (STDIL), the number of words of input.

4. "Garbage" Word (GBGWD), filler for remaining student COMMON; e.g., CALL INITD(PROBN,STDIP, STDIL,GBGWD).

The array STDIP can be initilized by data statements, arithmetic assignment statements, or read statements in integer or Al format. (Do not use the commercial CALL READ.) If it is desired to place real numbers into STDIP it must be remembered that one extended precision real number fills three words of core and that the first element of a real array should be equivalenced to the <u>third</u> element of the corresponding integer array. This is because FORTRAN arrays are stored in reverse order in core. For the same reason, the first element of STDIP will be placed by AM into the last word in core, and following elements will be stored into descending core locations.

If greater versatility of input is desired, the FØRTRAN program can call an assembler subroutine which generates STDIP and in turn calls INITD. In this way, the student can be provided with input in the format of actual instructions, characters in card-code, paper tape, etc. These changes in the calling sequence must be noted: All calling arguments must be <u>addresses</u> of the parameters, not the parameters themselves. Also, STDIP is the address of the <u>last</u> location of input. For example:

```
      ENT        DATA
DATA  ...        ....
      CALL       INITD
      DC         PROBN
      DC         STDIP
      DC         STDIL
      DC         GBGWD
      EXIT
PROBN DC         1
STDIP BES   E    120
STDIL DC         120
GBGWD DC         /EEEE
      END
```

In this way, STDIP can be filled by such assembler pseudo-ops as:

DC /...	hex constant
DEC	2-word decimal integer or real constant
XFLC	entended precision real constant
EBC	entended BCD interchange code characters
DMES	printer hex (console, 1132 or 1403)
DN	name code constant.

The instructor can provide, by an LIBF to ZIPCO, paper tape or card-code characters.

Output locations and grading parameters are entered as data on five cards after the //XEQ and *FILES cards (and also after any data cards read by the mainline). The first two cards contain respectively the beginning and ending addresses of up to ten output buffers GROUT is to search for answers. The addresses are to be expressed in four digit hexadecimal, absolute, with two spaces between address, up to ten addresses per card.

FØRMAT (10(4A1,2X))

Card three contains five numbers which are the percentage points to be assigned for program efficiency. The first number is for mainline program length, the second for subroutine length, the third for length of COMMON, the fourth for number of instructions executed, and the fifth is the curve. The sum of all five parameters should equal 100. Each number should be expresses as three digits with two spaces between each.

FØRMAT (5(I3,2X))

Card four contains up to ten percentage points for answers in correct locations, one corresponding to each answer buffer defined in cards one and two. Card five contains three percentage points determining value of partially correct answers. The first parameter is percentage for completely correct answers, the second for answers within the correct buffers but not necessarily in correct order, and the third is for answers anywhere within COMMON. The sum of cards four and five must each equal 100. The formats are the same as for card three. For example (for a machine with 8K core):

```
1FF0  1FD0  1FE0  1FE8
1FFF  1FDF  1FE7  1FE8

010  010  010  040  030
015  015  040  030
070  020  010
```

At the end of execution, INITD will give a hex dump of DATFT to the printer. The standard input buffer is stored in DATFT in reverse order to that in which it is loaded into core. The first element of DATFT (last element in the FØRTRAN dump) is loaded into the last location of core and so forth.

Initilization of Standard Programs

The final step in preparing the system for grading student programs is to run the standard programs. These are to be run in the same manner as student programs, with the following changes in operating procedure:

1. Parameters to be passed to AM are the address of problem number and a student number of -1 (FFFF in hexadecimal).
2. All data switches on the console must be placed in the up position (FFFF hexadecimal).
3. The program will stop after the first instruction with an exit code of 301C hexadecimal in the SBR. All switches except 13 should be placed in the down position and the program started. The program should now stop with 3020 in the SBR (normal Exit). (If a core-dump is desired, put switch 14 up.) Restart the machine. AM will now store the information it has compiled on SAVGR to DBUGT, which will read SAVGR, MSGBF, and DATFT. DBUGT will determine that the program is a standard and will link to GRINP. GRINP will complete the initilization of DATFT with standard output and standard program efficiency. COMMON and DATFT will be dumped to the printer in hexadecimal. A link will be performed back to DBUGT, which will then handle the standard as if it were a normal student program (as a cross-check on the standard.) The standard program will receive a grade of 1000 points. All student programs will be graded in comparison to this standard grade. Student programs can now be run and graded on the system for all problems on which the standard has been initialized.

Computation of Grade

The computation of the student's grade is based on these factors:

- I. Answers
  - A. completely correct
  - B. partially correct
- II. Program efficiency
  - A. Mainline length
  - B. Subroutine length
  - C. Length of COMMON
  - D. Number of instructions executed
  - E. Standard curve
- III. Correct termination of program (EXIT)

To compute I,A, GROUT compares the contents of the output data blocks in the students COMMON to the corresponding standard output block, and computes the ratio of the number of correct answers the student finds to the length of the block (standard number of correct answers). This ratio is multiplied by the corresponding grading parameter for correct answers (entered into DATFT by INITD, data card #4). The sum of these 10 products is then multiplied by the grading factor for totally correct answers (INITD, first number, data card #5). GROUT then searches the student's output buffers, counting the number of correct answers placed anywhere within the correct data block. The ratio of the number of answers so found to the total number of possible answers, is multiplied by the grading parameter for answers within the correct data blocks (INITD, second number, card #5). All of COMMON is then searched for the correct answers found in any locations, the ratio to total answers is computed and multiplied by the parameter for answers within COMMON. The total points for answers is the sum of points for correct answers, answers within the correct buffers, and answers anywhere in COMMON.

Points for program efficiency are computed as the sum of points for program length, subroutine length, length of COMMON, number of instructions executed and standard curve. Points for program length are computed as the ratio of Standard program length to student program length, times the grading parameter for program length (INITD, first number, card #3). If the student did not receive a perfect score on answers and his program length was less than that of the standard, points for program length is computed as if his program length was the same as that of the standard. Points for subroutine, COMMON, and number of instructions are computed in a like manner.

Total grade is computed by multiplying points for answers by points for program efficiency. 25% of the grade is lost if the program is terminated by anything but a standard exit (AMS 20). A message to this effect is printed. The final grade is then scaled on a factor of 1000. It is important to note that the grade given by the system is based upon a comparison between the student program and a "standard" program, and not between the student and other student programs. For this reason, the final scaling of grades must be left to the instructor. The system does, however, give a fair grade in that the grade is proportional to the worth of the program (if the grading parameters are assigned properly), and that the instructor can easily tell from the output supplied to him, where to scale the grades.

Output of GROUT to GFILE

GROUT supplies certain pertinent information about the student's grade to the instructor by entering a 16 word record on GFILE for each program graded, unless the student passes a negative student number to AM. The contents of GFILE is as follows:

1. Record number (first record has total number of records saved).
2. Student number.
3. Problem number.
4. AMS exit code.
5. Total grade.
6. Points for completely correct answers.
7. Points for all answers.
8. Points for program efficiency.
9. Program length.
10. Subroutine length.
11. Length of COMMON.
12. and 13. Number of instructions executed. Since a program can possibly execute more than 32,767 instructions, (the greatest possible integer the machine can hold), AM divides the instruction count into two words. The first is the number of instructions divided by 10000, and the second is the remainder of the instruction count. In other words, 13 is the low order four decimal digits and 12 is the upper decimal digits.
14. Number of answers in correct locations.
15. Number of answers within correct data blocks.
16. Number of answers anywhere within COMMON.

CONCLUSION AND RECOMMENDATIONS

Difficulty with a fixed problem set to be used repeatedly, led to the approach employed which permits new problems to be introduced as frequently as necessary. This has been effective over several quarters. Experience has shown that a first program for the student should be extremely simple - something like reading a number into the computer and printing it out. This divorces the mechanics of basic input and output from other programming complexities and gives the student the satisfaction of having been on the computer very early in the course.

Additional instructions have been considered for the repertoire of the simulator. These might include arithmetic and cyclic shifts, multiplication and perhaps even division. Although these would permit the solution of more sophisticated problems and may make the simulated computer more like an actual one, they would not make a major advance to the learning obtained via the current basic machine commands.

Provision is made in the present systems for accommodating the five decimal digit student identification number at Florida Institute of Technology. This is inadequate for some schools and will ultimately be inadequate at F.I.T. when a change to Social Security numbers as identification occurs, as it most surely will.

The Assembly Monitor system is only serving a small quantity of people - those computer science majors who use it in machine language programming. However, they are not required to use it. Moreover, nearly all problems at the machine language level, have been individually designed and must result in a working program. Further work on this program is not recommended at this time.

APPENDIX I

10 T AAA LOAD ACCUMULATOR - LDA

The contents of the Accumulator are replaced by the contents of the effective address. The contents of the effective address are not changed. The Sign latch is set equal to the sign of the contents of effective address. The Overflow latch is not affected.

EA = AAA + contents of T (if T=0, EA=AAA)

Example: 10 4 625 EA=625+213=838

Before execution:		After execution:	
Accumulator	???????	Accumulator	+991246
I/R 4	+000213	I/R 4	+000213
Location 838	+991246	Location 838	+991246
Sign Latch	?	Sign Latch	Positive

11 T AAA STORE ACCUMULATOR - STA

The contents of the effective address are replaced by the contents of the Accumulator. The contents of the Accumulator are not changed. The Sign latch is set equal to the contents of the Accumulator. The Overflow latch is not affected.

EA = AAA + contents of T (if T=0, EA=AAA)

Example: 11 0 001 EA=001

Before execution:		After execution:	
Accumulator	-999999	Accumulator	-999999
Location 001	???????	Location 001	-999999
Sign Latch	?	Sign Latch	Negative

40 T AAA LOAD INDEX REGISTER - LDX

The contents of the specified Index Register T are replaced by the contents of the effective address AAA. The contents of the effective address are not affected. The Sign latch is set equal to the sign of the contents of the effective address. The Overflow latch is not affected.

EA = AAA (Note: T cannot be 0; this instruction must specify an Index Register.)

Example: 40 9 123 EA=123

Before execution:		After execution:	
I/R 9	???????	I/R 9	-999995
Location 123	-999995	Location 123	-999995
Sign Latch	?	Sign Latch	Negative

20 T AAA ADD TO ACCUMULATOR - ADD

The contents of the effective address are algebraically added to the contents of the Accumulator. The contents of the effective address are not changed. The sign latch is set equal to the sign of the result in the Accumulator. The Overflow latch is set on if sum exceeds +999999 or is less than -999999. When Overflow occurs, high-order digits are truncated. The Overflow latch is set OFF if overflow did not occur.

EA = AAA + contents of T (if T=0, EA=AAA)

Examples:

Accumulator Before	EA	Accumulator After	Overflow	Sign
-999999	-000001	000000	ON	0
-001001	+000001	-001000	OFF	Neg.
-999999	+999999	000000	OFF	0
+010010	000000	+010010	OFF	+
+999999	+000001	000000	ON	0
+999999	+999999	+999998	ON	+

21 T AAA SUBTRACT FROM ACCUMULATOR - SUB

The contents of the effective address are algebraically subtracted from the contents of the Accumulator. The contents of the effective address are not changed. The sign latch is set equal to the sign of the result in the Accumulator. The Overflow latch is set on if the result is greater than +999999 or less than -999999. When overflow occurs, high-order digits are truncated. The Overflow latch is set off if overflow did not occur.

EA = AAA + contents of T (if T=0, EA=AAA)

Examples:

Accumulator Before	EA	Accumulator After	Overflow	Sign
-999999	-999999	000000	OFF	0
-999999	+000001	000000	ON	0
+000001	+999000	-998999	OFF	Neg.
+999998	+000001	+999997	OFF	+
+999999	-999999	+999998	ON	+

42 T AAA ADD TO INDEX REGISTER - MDX

The contents of the effective address AAA are algebraically added to the contents of the specified Index Register T. The contents of the effective address are not changed. The Sign latch is set equal to the sign of the result in the Index Register. The Overflow latch is set on if sum exceeds +999999 or is less than -999999. When overflow occurs, high-order digits are truncated. The Overflow latch is set off if overflow did not occur.

EA = AAA (Note: T cannot be 0; this instruction must specify an Index Register.)

Example: 42 5 002

Before execution:		After execution:	
I/R 5	-999999	IR/5	000000
Location 002	-000001	Location 002	-000001
Sign latch	?	Sign latch	0
Overflow latch	?	Overflow latch	ON

41 T AAA STORE INDEX REGISTER - STX

The contents of the effective address AAA are replaced by the contents of the specified Index Register T. The contents of Index Register T are not affected. The Sign latch is set equal to the sign of the contents of Index Register T. The Overflow latch is not affected.

EA = AAA (Note: T cannot be 0; this instruction must specify an Index Register.)

Example: 41 1 402 EA=402

Before execution:		After execution:	
I/R 1	000000	I/R 1	000000
Location 402	??????	Location 402	000000
Sign Latch	?	Sign Latch	0

60 T AAA READ A CARD - IN

Data is read in from a card and temporarily held in a buffer area. The data in the buffer is then checked for validity. If the first column contains an asterisk, the current program is terminated. If not, the first column must be a blank, plus sign, or minus sign. Blank is treated as a plus sign. Columns 2 through 7 must contain digits from 0 to 9 --- blanks are not allowed. Columns 8 - 80 may contain comments.

If the validity checking does not detect an error, the data is loaded into the core location specified by the effective address. If the data is invalid, the contents of the effective address are not altered. The Overflow and Sign latches are not affected in any case.

EA = AAA + contents of T (if T=0, EA=AAA)

Example: 60 1 427 (data in card, +426351) EA=427+111=538

Before execution:		After execution:	
I/R 1	+000111	I/R 1	+000111
Location 538	??????	Location 538	+426351

61 T AAA WRITE - OUT

The contents of the effective address is printed on the printer, and the paper is advanced one space. The Sign and Overflow latches are not affected.

77 0 000 STOP - HLT

Execution is terminated. The Sign and Overflow latches are not affected. Core is dumped onto the printer, ten locations per line for any line containing a word in which any change has been made in storage during execution.

50 T AAA BRANCH (Unconditional) - B

Control is transferred to the instruction at the effective address. The Sign and Overflow latches are not affected.

EA = AAA + contents of T (if T=0, EA=AAA)

Example:	Core location	Contents
	042	500862
	043	??????
	...	......
	862	210044

Execution of the Branch instruction at location 042 will cause the next instruction executed to be the subtract instruction at location 862.

51 T AAA BRANCH NEGATIVE - BN

This instruction causes a branch to the effective address if the Sign latch is Negative. If the Sign latch is not negative, control goes to the next sequential address. The sign and Overflow latches are not altered.

EA = AAA + contents of T (if T=0, EA=AAA)

52 T AAA BRANCH ZERO - BZ

This instruction causes a branch to the effective address if the Sign latch is zero. Otherwise, control goes to the next sequential address. The Sign and Overflow latches are not altered.

EA = AAA + contents of T (if T=0, EA=AAA)

53 T AAA BRANCH POSITIVE - BP

This instruction causes a branch to the effective address if the Sign latch is positive. Otherwise, control goes to the next sequential address. The Sign and Overflow latches are not altered.

EA = AAA + contents of T (if T=0, EA=AAA)

54 T AAA BRANCH OVERFLOW - BØ

This instruction causes a branch to the effective address if the Overflow latch is ON. Otherwise, control goes to the next sequential address. If branch occurs, then the Overflow latch is reset to OFF. The Sign latch is not affected.

EA = AAA + contents of T (if T=0, EA=AAA)

APPENDIX II

Problems 1, 3, 4, 5, and 6 are from the winter quarter 1969. Problems 11, 12, 13, 14 are from the spring quarter 1969.

PROBLEM NO. 1

Given: A set of 100 data cards containing values X_i such that:

$i = 1, 2, 3, \ldots, 100$
$-999999 \leq X_i \leq +999999$

Write a machine language problem beginning in location 0 (zero) to solve the following equation:

$$\sum_{i=1}^{100} X_i \text{ where } 0 \leq X_i \leq 1000$$

i.e.; omit values of X_i outside of the above range from the sum.

Be as efficient as possible.
Write out the answer on the printer.
Store your answer in location 900.
Read the given input data into locations 500-599.
Use index register(s) and conditional instruction(s).

PROBLEM NO. 3

Given: A set of 100 data cards containing values X_i such that:

$i=1, 2, 3, \ldots 100$
$-999999 \le X_i \le +999999$

Write a machi-e language program beginning in location 0 (zero) to solve the following equations:

$$\text{Sum } 1 = \sum_{i=1}^{99} X_i$$ (Sum the contents of only the odd numbered locations: i=1,3,5,...99)

$$\text{Sum } 2 = \sum_{i=2}^{100} X_i$$ (Sum the contents of only the even numbered locations: i=2,4,6,...100)

Write out both answers on the printer.
Store the answers: Sum 1 in location 900
Sum 2 in location 901

Read the given input data into locations 500-599.
Assume no overflow will occur.
Use any instructions you think necessary.
Be as efficient as possible.

PROBLEM NO. 4

Given: A set of 100 data cards containing values X_i such that:

$i=1, 2, 3, \ldots 100$
$-999999 \le X_i \le +999999$

Write a machine language program beginning in location 0 (zero) to perform the following:

(a) Find ANS. 1 = total number of negative items in the list
(b) Find ANS. 2 = total number of zero items in the list.
(c) Find ANS. 3 = total number of positive items in the list.

Problem No. 4 (cont'd)

Read the given data into locations 500-599
Store the answers: Ans. 1 in loc 900
Ans. 2 in loc 901
Ans. 3 in loc 903

Use any instructions you think necessary.
Be as efficient as possible.

PROBLEM NO. 5

Given: Two sets of 50 data cards containing values

$$\left.\begin{matrix} X_i \\ Y_i \end{matrix}\right\} \quad i=1, 2, 3, \ldots 50$$

such that $-5000 \leq X_i \leq +5000$
$-5000 \leq Y_i \leq +5000$

Find the sum of the differences $(X_i - Y_i)$ by the following formula:

$$\sum_{i=1}^{50} (X_i - Y_i)$$

Read the first set of fifty cards into locations 500-549.
Read the second set of fifty cards into locations 550-599.
Write out the answer on the printer.
Store the answer in location 900.
Use any instructions you think necessary.
Be as efficient as possible.

PROBLEM NO. 6

Determine and print the first N numbers of the "FIBBONACCI" series. In the "FIBBONACCI" series each number is the sum of the previous two numbers with the first two numbers of the series being 0 and 1.

Example of the "FIBBONACCI" series:
0, 1, 1, 2, 3, 5, 8....(to N terms of the series)

Read the value of N into location 500. Store the terms of the series starting in location 900. Print the terms of the series.

PROBLEM NO. 11

Write a program which will evaluate

$$f(x) = 3x^2+2x+7$$

for x an integer ($0<x<100$) to be read in from a data card. Test x after reading to make sure it is correct. Print out the value of x and f(x). Store f(x) in 900. If the value of x is out of the allowable range, print out the actual value of x, 000000 for f(x), and stop.

PROBLEM NO. 12

Write a program which will read (1) a card with the integer $0<N<100$. (2) N data cards into N successive locations, then sort the N numbers into ascending order and print them out. Read the data cards into locations 200ff and sort into locations 300ff.

PROBLEM NO. 13

Write a program to read in 25 numbers. These are to be stored in consecutive locations starting at 200. The numbers represent consecutive elements in consecutive rows of a matrix. Perform the transpose of the matrix so that rows and columns are interchanged. Print out the transposed matrix. Store transpose in locations 300ff.

PROBLEM NO. 14

Given three sets of data cards of $N\leq30$ cards each: Read the first set of N cards into locations 100, 103, 106,...

Read the second set of N cards into locations 101, 104, 107,...

Read the third set of N cards into locations 102, 105, 108,...

Print out in order locations 100, 102, etc.

N is on first card. (A total of 3N+1 cards will be read.)

APPENDIX III

This appendix contains summaries of the results of three surveys conducted after the automated problem sets had been used by several classes.

First is the student response to a questionaire which followed the course.

Second is the concensus of the instructor who taught the course.

Third is observations of the IBM-1130 operator who actually accepted the students programs and batch processed them.

Student Survey on Automated Problem Sets

A questionnaire (Table I) was prepared to ascertain the effectivity of the automated problem sets from the standpoint of the students. This questionnaire and the summarized responses from 134 students are shown. The questions were designed to determine the extent of ease or difficulty which the new (to the students) concept of machine language was assimilated. Results were obtained after the student had subsequently been exposed to, and had written programs in, a compiler language, namely, FORTRAN.

The final question requesting comments on improvement of the course elicited response from approximately fifty percent of the questionnaires. It opened a Pandora's box with a great diversity of opinions expressed. At the extremes, these ranged from the ideas that machine language was a complete waste of time and all programming training should be concentrated on FORTRAN to the desire to have the full quarter devoted to binary machine language with more emphasis on arithmetic and control unit organization. Specific comments also dealt with insufficient demonstration on keypunch, need to have first programs examined in detail by instructor before attempting to run, need for monitors to be better versed in the simulation language and in the problems assigned that quarter. A majority of the opinions expressed reflected the students' personal desires in results of such a course and in their success or frustrations in achieving these desires.

The following numbered observations correspond to the questions of the same number shown in Table I.

1. Less than two percent of the students had any prior experience with machine language.

2. Eighty percent believed that the instruction set was about the right complexity with the rest equally divided between too simple and too complex.

3. Responses were equally divided between those accepting the set as adequate and those desiring a multiply instruction. The fact that a negligible number thought shifting should be included probably indicates that its use was not pointed out to the students.

4. A negligible number of responses felt that the number of branch instructions was excessive and about a third wanted even more variations.

5. Opinion was about 7-5 in favor of a less restrictive I/O set.

6. Opinion was about equally divided for and against inclusion of logical instructions.

7. The decimal coding was almost universally accepted as suitable for grasping the essentials of machine language. A few dissidents identified a desire for binary.

8. Less than twenty percent considered the brief study of machine language a waste of time for the ultimate user.

9. All debugging aids provided proved helpful but the greatest aid was discussion with other students.

10. Difficulties with getting ultimately successful runs were most impeded by the actual closed shop mechanism of the Computer Center (probably underqualified monitors, bugs still in the program, and general lack of understanding of procedures). Failure to understand the function of the simulated computer operations and errors in card punching were also substantial contributors.

11. A large majority (over ninety-eight percent) considered the problem set reasonably difficult with the rest equally divided between too hard and too easy.

12.}
15.} Problem difficulty was rated roughly equal.

13.}
14.} The most difficult problems took three quarters of the students less than four hours of homework and less than five computer runs.

16.}
17.} The easiest problem took three quarters of the students less than two hours of homework and less than three computer runs.

18.}
19.} Results of this question appear to belie the preceding two results. For if the program were indeed tested and ready for the run for record it should succeed on the first, or at worst, second run. The statistics indicate that many used four or more of these runs on their more difficult problem.

20. A majority felt that there was a sufficient diversity in the problem set although several felt the problems were too similar.

21.} 22.} Analysis, coding and debugging difficulty varied much between individuals and no one stood out as uniformly particularly hard or particularly easy.

TABLE I

TO: Students who took CS162 during Winter Term 1969

FROM: D. R. Clutterham, Head of Mathematical Sciences Dept.

We need to obtain some information regarding the use of the simulated computer used to teach machine language in the CS162 course. Please complete the following questionnaire as accurately as possible and return to the Mathematical Sciences Department in person or by campus mail. If desired you may delete the portion above the double line to preserve anonymity. Please complete and return immediately.

Underline answer which fits your case.

1. Had you ever worked with machine language before?
   (a) yes (b) no

2. The instruction set provided was
   (a) too complex, (b) about right, (c) too elementary

3. The arithmetic instructions
   (a) were adequate, (b) should have included shifting,
   (c) should have included multiplication.

4. The branch instruction set
   (a) was adequate, (b) could be improved with some additional types, (c) had too many alternatives.

5. The input/output set of instructions was
   (a) too restrictive, (b) adequate, (c) should permit formatting

6. Logic instructions should be included
   (a) no, (b) such as "AND", "OR", "COMPLEMENT."

7. Greater understanding of machine language would have been obtained if numbers and codes had been
   (a) in octal, (b) in hexadecimal, (c) in binary,
   (d) the decimal used was adequate.

8. The study of machine language
   (a) is a waste of time for an ultimate user
   (b) gave me a much better understanding of computers
   (c) contributed to my appreciation of FORTRAN

9. The most helpful debugging aid was
(a) the program trace, (b) the memory and status dump, (c) discussion with monitor, (d) discussion with classmates

10. The greatest difficulty in completing a program successfully was
(a) incomplete understanding of instructions
(b) getting results from a run on the computer
(c) punching an accurate set of cards

11. The problem set to be solved
(a) was adequate, (b) was too difficult, (c) was too easy

12. The problem which was most difficult for me was
(a) 1 (b) 2 (c) 3 (d) 4 (e) 5

13. The problem which was most difficult for me required
(a) less than 2 hours of homework
(b) two to 4 hours of homework
(c) four to 10 hours of homework
(d) over 10 hours of homework

14. The problem which was most difficult for me required
(a) less than 3 computer runs, (b) 3 to 5 computer runs
(c) 6 to 9 computer runs, (d) more than 9 computer runs

15. The problem which was easiest for me was
(a) 1 (b) 2 (c) 3 (d) 4 (e) 5
(f) don't remember

16. The problem which was easiest for me required
(a) less than 2 hours of homework, (b) 2 to 4 hours of homework, (c) 4 to 10 hours of homework
(d) over 10 hours of homework

17. The problem which was easiest for me required
(a) less than 3 computer runs, (b) 3 or 4 computer runs
(c) 5 to 7 computer runs, (d) 8 or more computer runs

18. My easiest problem ran correctly on my run for record number
(a) 1 (b) 2 (c) 3 (d) 4 or greater

19. My hardest problem ran correctly on my run for record number
(a) 1 (b) 2 (c) 3 (d) 4 or greater

20. The problems in our problem set
(a) were about right, (b) were too similar,
(c) were too different

21. The part of these problems I found easiest was
(a) analysis, (b) coding, (c) debugging

22. The part of these problems I found hardest was
(a) analysis, (b) coding, (c) debugging

23. Include any comments for improving this part of the course.

Survey of Instructors Using the Automated Problem Sets

Seven instructors have been introduced to the automated problem sets and five have taught the introductory computer course at Florida Institute of Technology using the sets. Their observations are summarized here.

When a class is given a common problem, there is a tendency to either copy the solution of one of the better students or to work collectively on a program so that the net result is several groups of identical solutions. This problem is not peculiar to this course or even this subject, but usually students vary their own solutions from the one they copy and this is not done with the automated problem sets. One solution may be to have the students turn in their handwritten coding sheet before they begin their actual machine debugging; then their final programs should be modifications to the handwritten ones. Another solution is to develop a very large set of similar problems so that students have essentially an individual problem.

The instruction code set seems generally suitable to the instructors. More experienced instructors found the set quite suitable or else desired only a shift operation. Newly indoctrinated instructors desired a multiply and perhaps also a divide instruction. Somewhat more capability in the input-output format appears desirable, although exactly what form it should take was not agreed upon. A set of left and right shifts with and without a circular capability have been designed for the program but are not now incorporated.

One anticipated problem - that of teaching the use of the keypunch in classroom - has not arisen; learning the use of the keypunch seems to be passed very readily between the students, and a minimum of words from the instructor is sufficient.

Survey of Machine Operators Using the Simulator

An initial complaint of the operators was that instructors did not sufficiently define the problem to the students and further definition had to be supplied in detail. This is recognized as a continuing problem and the instructors are putting more care and detail into the definition.

A second difficulty is that assignments are relatively few, but everyone's problem comes due at the same time. Even if the assignments are given well in advance, normal student procrastination causes a heavy run on both the card punching equipment and on the computer in the last couple of days before a grading run is due. A solution to this problem, as yet untried, is to stager problem due dates giving easier problems to the students whose problems are due first. In addition to this, simpler problems could be given much earlier in the quarter so that the students can first learn some of the mechanics of preparing a problem for the machine and getting basic input-output mastered.

APPENDIX IV

Program Listing

```
// JOB
// *
// *PROGRAM TO READ IN DATA FILE,INITIALIZE AND START SIMULATION
// * OF A DECK OF SIM610 PROGRAMS.
// *
// FOR
*NAME STRTG
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*LIST SOURCE PROGRAM
*ONE WORD INTEGERS
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
      INTEGER A(2205),INPUT(160),NREM(77),DATA(212),PRSET(15)
      INTEGER TABLE(16)
      INTEGER ERR,EA
      INTEGER DATA1(106),DATA2(106)
      COMMON A,INPUT,NREM,DATA,PRSET
      EQUIVALENCE (NPROB,A(2140)),(TABLE(1),A(2116))
      EQUIVALENCE (INIT,A(2138))
      EQUIVALENCE (EA,A(2025)),(ERR,A(2109))
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
      EQUIVALENCE(LOC11,A(1)),(LOC12,A(1001))
      EQUIVALENCE (NI,A(2114)),(NO,A(2115))
      DEFINE FILE 5(12,106,U,NXRDC)
    1 NI=2
      NO=5
      READ(NI,11) TABLE,NDTST,INIT,PRSET
11    FORMAT(16A1,I1,1X,I1,1X,15I2)
      IF(INIT-1) 10,12,10
10    INIT = -1
      GO TO 13
12    INIT = 0
13    IF(NDTST) 16,16,14
14    IF(NDTST-6) 15,15,16
15    READ(5'2*NDTST-1) DATA
      GO TO 19
16    EA=1
      DO 4 I=1,106
      CALL RDR60
      DATA1(I)=LOC11
      DATA2(I)=LOC12
      IF(ERR) 3,2,3
3     PAUSE 7009
      I = I - 1
2     LOC12 = IABS(LOC12)
4     WRITE(NO,17) LOC11,LOC12
17    FORMAT(1H ,I4,I3)
19    CALL RDR60
      CALL LINK(LOADP)
      END
// DUP
*DELETE              STRTG
*STORECI    WS  UA  STRTG 0001
*FILES(5,SIMDT)
```

```
// JOB
// *
// * PROGRAM TO LOAD EACH STUDENT PROGRAM INTO PSEUDO CORE, AND
// * BRING IN THE FILE OF STANDARD DATA ON THE PROBLEM FOR GRADING.
// *
// DUP
*DELETE             LOADP
// FOR
*NAME LOADP
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
      INTEGER ERRS
      INTEGER ERROR
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMO(2)
      INTEGER IOBUF(48),NAME(32),ERRCT(5)
      INTEGER TABLE(16)
      INTEGER RNTIM(2),PROGL
      INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
      INTEGER NSAV1(30),NSAV2(30)
      INTEGER STORT,STOPL
      INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
      INTEGER NRDSR(10),LOCRD(10)
      INTEGER PTSR,PTSA,PTSW,PTS
      INTEGER FDATA,POSPT(3)
      INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCO,PTCW(10),PTWO
      INTEGER PCGRT,PCGPL
      INTEGER RDATA(14)
      INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMO,IOBUF,NAME
      COMMON ERRCT
      COMMON NI,NO,TABLE,JERR
      COMMON I,J,K,L,M
      COMMON INIT
      COMMON NSTUD,NPROB
      COMMON RNTIM,PROGL,NOCDS
      COMMON NANSW,NSAV1,NSAV2
      COMMON IDUMY,STORT,STOPL,ANS1,ANS2,NORDS,NRGPS,POSPT
      COMMON NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
      COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
      COMMON PTSR,PTSA,PTSW,PTS
      COMMON FILNO,PC,IOVFL,LINE,DATA
      EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
      EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
      EQUIVALENCE (POSPT(1),NPPTR),(POSPT(2),NPPTA),(POSPT(3),NPPTW)
      EQUIVALENCE (NI,NOCDS),(NWTR,NANSW)
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
      EQUIVALENCE (LOC11,LOC1(1)),(LOC12,LOC2(1))
      EQUIVALENCE (ERRCT(1) ERROR)
      DEFINE FILE 1(24,160,U,NXREC)
C------TEST FOR MONITOR CARD
    1 IF(ERROR) 2,10,10
    2 IF(IOBUF(2)-TABLE(1)) 10,20,10
   10 EA=1
      CALL RDR60
      GO TO 1
C------SKIP TO NEW PAGE, PRINT MONITOR CARD
   20 WRITE(NO,22) IOBUF
   22 FORMAT(1H1,16A1,32A2,//)
      DO 27 I=2,15
      DO 26 J=1,10
```

```
      IF(IOBUF(I)-TABLE(J)) 26,27,26                                    LOADP065
   26 CONTINUE                                                          LOADP066
C---ERROR  IOBUF(I) SET TO ZERO.                                        LOADP067
      J = 1                                                             LOADP068
   27 IOBUF(I) = J-1                                                    LOADP069
C------TEST FOR MONITOR START CARD                                      LOADP070
C     NCARD = (((((IOBUF(2)*10+IOBUF(3))*10+IOBUF(4))*10+IOBUF(5))*10+  LOADP071
C    1IOBUF(6))*10+IOBUF(7)                                             LOADP072
      NCARD = ((IOBUF(4)*10+IOBUF(5))*10+IOBUF(6))*10+IOBUF(7)          LOADP073
C     NO LIST OF SOURCE PROGRAM IF NCARD EQUALS ZERO.                   LOADP074
      IF(NCARD-1) 28,29,10                                              LOADP075
   28 NCARD = 2                                                         LOADP076
   29 NOCDS = 1                                                         LOADP077
      NSTUD = ((((IOBUF(9)*10+IOBUF(10))*10+IOBUF(11))*10+IOBUF(12))*10+ILOADP078
     1OBUF(13)                                                          LOADP079
      NPROB = 10*IOBUF(14) + IOBUF(15)                                  LOADP080
      DO 45 I=1,32                                                      LOADP081
   45 NAME(I) = IOBUF(I+16)                                             LOADP082
C                                                                       LOADP083
C-----------------------------------------------------------------------LOADP084
C     ROUTINE TO LOAD STUDENT PROGRAM INTO 1000 WORD PSUEDO-CORE.       LOADP085
C     A LISTING IS PRINTED OF ALL NON-MONITOR CARDS.                    LOADP086
C     MONITOR CARDS ARE IDENTIFIED BY AN ASTERISK IN COLUMN 1.          LOADP087
C     ROUTINE RETURNS ON READING A MONITOR CARD, OR WHEN CORE LOAD      LOADP088
C     EXCEEDS PSUEDO-CORE.                                              LOADP089
C     ON RETURN--IAR CONTAINS COUNT OF CORE LOCATIONS USED.             LOADP090
C                IOBUF CONTAINS LAST RECORD READ.                       LOADP091
C                ERRS CONTAINS COUNT OF ERROR FLAGS.                    LOADP092
C     ERRORS ARE FLAGGED WITH AN ASTERISK ON LISTING.                   LOADP093
C     ERRORS ARE ALWAYS LISTED.                                         LOADP094
C     LOADING STARTS IN CORE LOCATION ZERO.                             LOADP095
C                                                                       LOADP096
C     CLEAR PSUEDO-CORE.                                                LOADP097
      AREG(1)=20000                                                     LOADP098
      AREG(2)=20000                                                     LOADP099
      DO 3 IAR=1,18                                                     LOADP100
    3 XR(IAR) = 25000                                                   LOADP101
      DO 106 IAR=1,2000                                                 LOADP102
  106 LOC(IAR)=30000                                                    LOADP103
C     INITIALIZE IAR AND ERRS.                                          LOADP104
      IAR=0                                                             LOADP105
      ERRS=0                                                            LOADP106
  110 EA = IAR + 1                                                      LOADP107
      CALL RDR60                                                        LOADP108
      IF(ERROR) 120,140,130                                             LOADP109
  130 ERRS=ERRS+1                                                       LOADP110
C     PUT ASTERISK IN ERROR FLAG.                                       LOADP111
      ERROR=TABLE(15)                                                   LOADP112
      GO TO 150                                                         LOADP113
C     BLANK OUT ERROR FLAG                                              LOADP114
  140 ERROR=TABLE(12)                                                   LOADP115
C     NO LIST OF SOURCE PROGRAM IF NCARD EQUALS TWO                     LOADP116
      GO TO(150,160),NCARD                                              LOADP117
  150 WRITE(NO,51) ERROR,IAR,IOBUF                                      LOADP118
   51 FORMAT(1H ,A1,1X,I4,4X,7A1,4X,9A1,32A2)                           LOADP119
  160 IAR=IAR+1                                                         LOADP120
C     TEST FOR END OF PSUEDO-CORE.                                      LOADP121
      IF(IAR-999) 110,110,120                                           LOADP122
C-----* CARD OR END OF CORE ENCOUNTERED                                 LOADP123
  120 WRITE(NO,51) TABLE(12),IAR,IOBUF                                  LOADP124
      PROGL=IAR                                                         LOADP125
      ERROR = 0                                                         LOADP126
C                                                                       LOADP127
C-----ABORT IF MISPUNCHED CARD IN DECK.                                 LOADP128
      IF(ERRS) 30,30,10                                                 LOADP129
   30 IF(IOBUF(3)-TABLE(1)) 31,2,31                                     LOADP130
```

```
// JOB
// *
// * PROGRAM TO LOAD EACH STUDENT PROGRAM INTO PSEUDO CORE, AND
// * BRING IN THE FILE OF STANDARD DATA ON THE PROBLEM FOR GRADING.
// *
// DUP
*DELETE             LOADP
// FOR
*NAME LOADP
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
      INTEGER ERRS
      INTEGER ERROR
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMO(2)
      INTEGER IOBUF(48),NAME(32),ERRCT(5)
      INTEGER TABLE(16)
      INTEGER RNTIM(2),PROGL
      INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
      INTEGER NSAV1(30),NSAV2(30)
      INTEGER STORT,STOPL
      INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
      INTEGER NROSR(10),LOCRO(10)
      INTEGER PTSR,PTSA,PTSW,PTS
      INTEGER FDATA,POSPT(3)
      INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCO,PTCW(10),PTWO
      INTEGER PCGRT,PCGPL
      INTEGER RDATA(14)
      INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMO,IOBUF,NAME
      COMMON ERRCT
      COMMON NI,NO,TABLE,JERR
      COMMON I,J,K,L,M
      COMMON INIT
      COMMON NSTUD,NPROB
      COMMON RNTIM,PROGL,NOCOS
      COMMON NANSW,NSAV1,NSAV2
      COMMON IDUMY,STORT,STOPL,ANS1,ANS2,NOROS,NRGPS,POSPT
      COMMON NROSR,LOCRO,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
      COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
      COMMON PTSR,PTSA,PTSW,PTS
      COMMON FILNO,PC,IOVFL,LINE,DATA
      EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
      EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
      EQUIVALENCE (POSPT(1),NPPTR),(POSPT(2),NPPTA),(POSPT(3),NPPTW)
      EQUIVALENCE (N1,NOCOS),(NWTR,NANSW)
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
      EQUIVALENCE (LOC11,LOC1(1)),(LOC12,LOC2(1))
      EQUIVALENCE (ERRCT(1) ERROR)
      DEFINE FILE 1(24,160,U,NXREC)
C------TEST FOR MONITOR CARD
    1 IF(ERROR) 2,10,10
    2 IF(IOBUF(2)-TABLE(1)) 10,20,10
   10 EA=1
      CALL RDR60
      GO TO 1
C------SKIP TO NEW PAGE, PRINT MONITOR CARD
   20 WRITE(NO,22) IOBUF
   22 FORMAT(1H1,16A1,32A2,//)
      DO 27 I=2,15
      DO 26 J=1,10
```

```
      IF(IOBUF(I)-TABLE(J)) 26,27,26
   26 CONTINUE
C---ERROR  IOBUF(I) SET TO ZERO.
      J = 1
   27 IOBUF(I) = J-1
C------TEST FOR MONITOR START CARD
C     NCARD = ((((IOBUF(2)*10+IOBUF(3))*10+IOBUF(4))*10+IOBUF(5))*10+
C    1IOBUF(6))*10+IOBUF(7)
      NCARD = ((IOBUF(4)*10+IOBUF(5))*10+IOBUF(6))*10+IOBUF(7)
C     NO LIST OF SOURCE PROGRAM IF NCARD EQUALS ZERO.
      IF(NCARD-1) 28,29,10
   28 NCARD = 2
   29 NOCDS = 1
      NSTUD = ((((IOBUF(9)*10+IOBUF(10))*10+IOBUF(11))*10+IOBUF(12))*10+I
     1OBUF(13)
      NPROB = 10*IOBUF(14) + IOBUF(15)
      DO 45 I=1,32
   45 NAME(I) = IOBUF(I+16)
C
C-----------------------------------------------------------------------
C     ROUTINE TO LOAD STUDENT PROGRAM INTO 1000 WORD PSUEDO-CORE.
C     A LISTING IS PRINTED OF ALL NON-MONITOR CARDS.
C     MONITOR CARDS ARE IDENTIFIED BY AN ASTERISK IN COLUMN 1.
C     ROUTINE RETURNS ON READING A MONITOR CARD, OR WHEN CORE LOAD
C     EXCEEDS PSUEDO-CORE.
C     ON RETURN--IAR CONTAINS COUNT OF CORE LOCATIONS USED.
C                IOBUF CONTAINS LAST RECORD READ.
C                ERRS CONTAINS COUNT OF ERROR FLAGS.
C     ERRORS ARE FLAGGED WITH AN ASTERISK ON LISTING.
C     ERRORS ARE ALWAYS LISTED.
C     LOADING STARTS IN CORE LOCATION ZERO.
C
C     CLEAR PSUEDO-CORE.
      AREG(1)=20000
      AREG(2)=20000
      DO 3 IAR=1,18
    3 XR(IAR) = 25000
      DO 106 IAR=1,2000
  106 LOC(IAR)=30000
C     INITIALIZE IAR AND ERRS.
      IAR=0
      ERRS=0
  110 EA = IAR + 1
      CALL RDR60
      IF(ERROR) 120,140,130
  130 ERRS=ERRS+1
C     PUT ASTERISK IN ERROR FLAG.
      ERROR=TABLE(15)
      GO TO 150
C     BLANK OUT ERROR FLAG
  140 ERROR=TABLE(12)
C     NO LIST OF SOURCE PROGRAM IF NCARD EQUALS TWO
      GO TO(150,160),NCARD
  150 WRITE(NO,51) ERROR,IAR,IOBUF
   51 FORMAT(1H ,A1,1X,I4,4X,7A1,4X,9A1,32A2)
  160 IAR=IAR+1
C     TEST FOR END OF PSUEDO-CORE.
      IF(IAR-999) 110,110,120
C-----* CARD OR END OF CORE ENCOUNTERED
  120 WRITE(NO,51) TABLE(12),IAR,IOBUF
      PROGL=IAR
      ERROR = 0
C
C-----ABORT IF MISPUNCHED CARD IN DECK.
      IF(ERRS) 30,30,10
   30 IF(IOBUF(3)-TABLE(1)) 31,2,31
```

```
C       SKIP TO NEW PAGE IF LISTING MADE.
   31 GO TC(32,35),NCARD
   32 WRITE(NC,33)
   33 FORMAT(1H1)
C      READ(1'FILNO) ICUMY,STDRT,STDPL,ANS1,ANS2,NCRDS,NRGPS,PCSPT,
C     1 NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO,
C     2 PTWC,NANS,FDATA,MAXRT,PCGRT,PCGPL(,RDATA)
   35 CALL RDSTD
C     SIMULATE RUN.
      CALL SIMRN
C     DUMP GRADING INFORMATION.
      CALL LINK(DUMPG)
      END
// DUP
*STORECI    WS  UA  LOADP 0002
*LOCAL,RDR6C,DWADC,DECEB
*FILES(1,FSTDG)
```

```
// JOB
// *
// *ROUTINE WHICH ACTUALLY SIMULATES EXECUTION OF THE PROGRAM
// * IN PSEUDO-CORE.
// *
// FOR
*LIST ALL
*EXTENDED PRECISION
*ONE WORD INTEGERS
      SUBROUTINE SIMRN
      INTEGER TCNTR
      INTEGER SHFTC,CARRY,CARY2
      INTEGER OPTBL(44),NUTBL(44)
      INTEGER AREG1,AREG2
      INTEGER MREG(2)
      INTEGER CXR(2),CXR1,CXR2
      INTEGER            NXREG(2),CEAR(2) ,                NNREG(2)
      INTEGER    IIBUF(7),JJBUF(7),KKBUF(7),LLBUF(7),MMBUF(7),NNBUF(7)
      INTEGER CEAR1,CEAR2
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMC(2)
      INTEGER IOBUF(80),ERRCT(5)
      INTEGER TABLE(16)
      INTEGER RNTIM(2),PROGL
      INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
      INTEGER NSAV1(30),NSAV2(30)
      INTEGER STDRT,STDPL
      INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
      INTEGER NRDSR(10),LOCRD(10)
      INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCC,PTCW(10),PTWO
      INTEGER FDATA,POSPT(3)
      INTEGER PCGRT,PCGPL
      INTEGER RDATA(14)
      INTEGER PTSR,PTSA,PTSW,PTS
      INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMC,IOBUF,ERRCT
      COMMON NI,NO,TABLE,JERR
      COMMON I,J,K,L,M
      COMMON INIT
      COMMON NSTUD,NPROB
      COMMON RNTIM,PROGL,NOCDS
      COMMON NANSW,NSAV1,NSAV2
      COMMON IDUMY,STDRT,STDPL,ANS1,ANS2,NORDS,NRGPS,PCSPT
      COMMON NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
      COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
      COMMON PTSR,PTSA,PTSW,PTS
      COMMON FILNO,PC,IOVFL,LINE,DATA
      EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
      EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
      EQUIVALENCE (POSPT(1),NPPTR),(POSPT(2),NPPTA),(POSPT(3),NPPTW)
C     PUT INTEGERS USED ONLY HERE IN LINE TO SAVE CORE.
      EQUIVALENCE (LINE(1),IIBUF(1)),(LINE(8),JJBUF(1)),(LINE(15),
     CKKBUF(1)),(LINE(22),LLBUF(1)),(LINE(29),MMBUF(1)),(LINE(36),
     CNNBUF(1)),(NXREG(1),NXRG1),(NXREG(2),NXRG2)
      EQUIVALENCE (LINE(43),CXR(1)),(LINE(45),MREG(1))
     1,(LINE(47),NXREG(1)),(LINE(49),CEAR(1)),(LINE(51),NNREG(1))
      EQUIVALENCE (LINE(53),TCNTR),(LINE(54),SHFTC)
      EQUIVALENCE (LINE(55),CARRY),(LINE(56),CARY2)
      EQUIVALENCE (LINE(57),MSW  ),(LINE(58),LCTR2)
      EQUIVALENCE (LINE(59),IAR  ),(LINE(60),IFLAG)
      EQUIVALENCE (CEAR(1),CEAR1),(CEAR(2),CEAR2)
      EQUIVALENCE (CXR(1),CXR1),(CXR(2),CXR2)
      EQUIVALENCE (MREG1,MREG(1)),(MREG2,MREG(2))
      EQUIVALENCE (AREG(1),AREG1),(AREG(2),AREG2)
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
```

```
      EQUIVALENCE (NWTR,NANSW)
      DATA OPTBL/0,10,0,11,0,20,C,21,0,30,0,31,0,32,0,33,1,40,1,41,1,42,
     C          0,50,0,51,0,52,0,53,0,54,0,60,0,61,0,77,-1,100/
      DATA NUTBL/'LD','A ', 'ST','A ', 'AD','C ', 'SU','B ',
     C          'SL','A ', 'SR','A ', 'RL','A ', 'RR','A ',
     C          'LD','X ', 'ST','X ', 'MD','X ', 'B ','  ',
     C          'BN','  ', 'BZ','  ', 'BP','  ', 'BC','  ',
     C          'IN','  ', 'OU','T ', 'HL','T ',' U','  '/
C
C
C     JERR RETURNS...
C        =1  SUCCESSFUL EXECUTION
C        =2  INVALID INSTRUCTION CAUSED ABORT
C        =3  TIME EXHAUSTED CAUSED ABORT
C        =4  MONITOR CARD READ BY PROGRAM. CARD IS IN ICBUF
C
C
C-----INITIALIZE SIMULATOR.
    1 PC = 0
      TCNTR=0
      MSW=2
      RNTIM(1)=0
      RNTIM(2)=0
      JERR=0
      ISIGN=0
      IOVFL=0
      NOCDS = 0
      NWTR=0
      LCTR = 0
C
C-----BUMP IAR
 1000 IAR = PC+1
C-----LOAD C(PC) INTO MREG.
      MREG1=LOC1(IAR)
      MREG2=LOC2(IAR)
C-----
      RNTIM(2)=RNTIM(2)+1
C
C-----STATICIZE INSTRUCTION INTO OPCOD,TAG,ADDR
C     THIS ROUTINE STATICIZES A PSEUDO-MACHINE INSTRUCTION
C     CONTAINED IN THE DOUBLE-WORD REGISTER REG.
C     EXAMPLES...
C    REG(1)   REG(2)    INSTR      TAG      ADDR
C     315      208       31         5       208
C     403      772       40         3       772
C    -315     -208      -31         5       208
      OPCOD = MREG1/10
      TAG = MREG1-OPCOD*10
      ADDR = MREG2
C
C-----COMPUTE INSTR,EA
C     THIS ROUTINE CONVERTS A PSEUDO-LANGUAGE OP-CODE IN OPCOD INTO
C     AN INTEGER FOR USE IN A COMPUTED GO TO STATEMENT FOR
C     INSTRUCTION SIMULATION.  THE EFFECTIVE ADDRESS IS ALSO COMPUTED.
C     THE INSTRUCTION MNEUMONIC IS RETURNED IN NEUMO AS 2A2.
C
C     CONVERSION REQUIRES A TABLE OF VALID OP-CODES AND CONDITIONS.
C     NOINS IS THE LENGTH OF THE TABLE.
C     OPTBL CONTAINS POSITIVE THREE DECIMAL DIGIT INTEGERS.
C     THE FIRST TWO DIGITS ARE THE OP-CODE FOR THE INSTRUCTION.  THE
C     LAST DIGIT IS A CONDITION FLAG.
C        =0 FOR NO CONDITION
C        =1 IF INDEX TAG IS REQUIRED
C     INSTR WILL BE SET TO THE SUBSCRIPT NUMBER OF OPTBL IF A MATCH
C     IS FOUND.  IF THERE IS NO MATCH, INSTR RETURNS =0.
C
```

```
      EA=ADDR + 1
C-----SEARCH OP-CODE TABLE FOR MATCH
      IF(OPCODE-100) 7,30,30
    7 INSTR = 0
    8 INSTR = INSTR + 2
      IF(OPCODE-OPTBL(INSTR)) 30,35,8
C-----INVALID OP-CODE
   30 INSTR = 20
      GO TO 160
C
C-----SET CONDITION FLAG
   35 IFLAG = OPTBL(INSTR-1)
C-----MAKE INSTR EQUAL TO SEQUENCE NO. OF INSTRUCTION.
      INSTR = INSTR/2
C-----COMPUTE EFFECTIVE ADDRESS
      CXR1 = XR1(TAG)
      CXR2 = XR2(TAG)
      IF(IFLAG-1) 45,40,30
C-----REQUIRED TAG MISSING
   40 IF(TAG) 160,160,200
C-----IS INSTRUCTION (ADDRESS) INDEXED
   45 IF(TAG) 200,200,70
C-----ANY ERRORS...
  160 JERR = 2
      GO TO 570
C
C-----COMPUTE EFFECTIVE ADDRESS FOR INDEXED INSTRUCTIONS.
  70  EA=EA+ CXR(2)+1000
      EA=EA-(EA/1000)*1000
C
C
C-----SAVE CONTENTS OF EA
 200  CEAR1=LOC1(EA)
      CEAR2=LOC2(EA)
C-----EXECUTE INSTRUCTION
      GO TO (1100,1110,1200,1210,1300,1300,1320,1320,1401,1411,1421,
     C    1500,1510,1520,1530,1540,1600,1610,1770), INSTR
C
C
C
C----------------------------------------------------------------------
C
C-----LOAD ACCUMULATOR. SET SIGN LATCH.
C
 1100 AREG1=CEAR1
      AREG2=CEAR2
 1105 CALL LATCH(AREG)
      GO TO 500
C
C----------------------------------------------------------------------
C
C-----STORE ACCUMULATOR. SET SIGN LATCH.
 1110 LOC1(EA)=AREG1
      LOC2(EA)=AREG2
      GO TO 1105
C
C----------------------------------------------------------------------
C
C-----ADD TO ACCUMULATOR. SET SIGN AND OVFL LATCHES.
 1200 CALL DWADD(AREG,CEAR,AREG,ICVFL)
      GO TO 500
C
C----------------------------------------------------------------------
C
C-----SUBTRACT FROM ACCUMULATOR. SET SIGN AND OVFL LATCHES.
 1210 NNREG(1)=-CEAR1
```

```
      NNREG(2)=-CEAR2
      CALL DWADD(AREG,NNREG,AREG,IOVFL)
      GO TO 500
C
C
C-----SHIFT LEFT ACCUMULATOR.
C-----SHIFT RIGHT ACCUMULATOR
C------------------------------------------------------------------
C-----NEGATIVE SHIFT COUNT GIVES INVALID INSTRUCTION.
 1300 IF(CEAR1) 160,1301,1329
C-----ZERO SHIFT COUNT SETS SIGN LATCH ONLY.
 1301 IF(CEAR2) 160,1373,1302
 1302 IF(CEAR2-6) 1303,1329,1329
 1303 SHFTC = CEAR2
      GO TC 1340
C------------------------------------------------------------------
C
C-----ROTATE LEFT ACCUMULATOR.
C-----ROTATE RIGHT ACCUMULATOR.
C----            1000 MCD 6 EQUALS 4.
C1320 SHFTC = 1000*(CEAR1 - 6*(CEAR1/6)) + CEAR2
 1320 SHFTC = 4*(CEAR1 - 6*(CEAR1/6)) + CEAR2
      SHFTC = SHFTC - 6*(SHFTC/6)
      IF(INSTR-8) 1340,1330,1340
 1330 SHFTC = 6 - SHFTC
C
C------------------------------------------------------------------
C-----ALL SHIFTS
 1340 K = 0
C-----TO AVOID FORTRAN DIVISION CF NEGATIVE NUMBERSIN SHIFTS.
      IF(AREG2)1346,1345,1350
 1345 IF(AREG1)1347,1350,1350
 1346 AREG2 = -AREG2
 1347 AREG1 = -AREG1
C-----SAVE FACT THAT SIGN IS NEGATIVE.
      K = 1
C
 1350 IF(INSTR-6) 1351,1361,1351
C
C
C-----ROTATE INSTRUCTIONS
C-----SHIFT LEFT ACCUMULATOR.
 1351 DO 1359 I=1,SHFTC
      CARRY = AREG2/100
      AREG2 = (AREG2-100*CARRY)*10
      CARY2 = AREG1/100
      AREG1 = (AREG1-100*CARY2)*10 + CARRY
      IF(INSTR-7) 1356,1358,1358
C-----SHIFT LEFT ONLY - SET OVERFLOW IF NONZERO DIGIT SHIFTED OUT.
 1356 IF(CARY2) 1357,1359,1357
 1357 IOVFL = 1
C-----ROTATE INSTRUCTIONS ONLY
 1358 AREG2 = AREG2 + CARY2
 1359 CONTINUE
      GO TO 1371
C
C-----SHIFT RIGHT ACCUMULATOR
 1361 DO 1369 I=1,SHFTC
      CARY2 = AREG1/10
      CARRY = AREG1 - 10*CARY2
      AREG1 = CARY2
 1369 AREG2 = AREG2/10 + 100*CARRY
C RESTORE SIGN OF ACCUMULATOR. SET SIGN LATCH.
 1371 IF(K) 1373,1373,1372
 1372 AREG2 = -AREG2
      AREG1 = -AREG1
```

```
 1373 CALL LATCH(AREG)
      GO TO 5C0
C-----SHIFT COUNT GREATER THAN SIX
 1329 AREG1 = 0
      AREG2 = 0
      ISIGN = 0
      GO TC 500
C
C-----------------------------------------------------------------------
C
C-----------------------------------------------------------------------
C
C-----LOAD XR. SET SIGN LATCH.
 1401 XR1(TAG)=CEAR1
      XR2(TAG)=CEAR2
      CALL LATCH(CEAR)
      GO TO 5C0
C
C-----------------------------------------------------------------------
C
C-----STORE XR. SET SIGN LATCH.
 1411 LOC1(EA)=CXR1
      LOC2(EA)=CXR2
      CALL LATCH(CXR)
      GO TO 5C0
C
C-----------------------------------------------------------------------
C
C-----ADD TO XR. SET SIGN AND OVFL LATCHES.
 1421 CALL DWADD(CXR,CEAR,NXREG,IOVFL)
      XR1(TAG) = NXRG1
      XR2(TAG) = NXRG2
      GO TO 5C0
C
C-----------------------------------------------------------------------
C
C-----UNCONDITIONAL BRANCH.
 1500 IAR = EA - 1
      GO TO 500
C
C-----------------------------------------------------------------------
C
C-----BRANCH ON NEGATIVE.
 1510 IF(ISIGN) 1500,500,500
C
C-----------------------------------------------------------------------
C-----BRANCH ON ZERC.
 1520 IF(ISIGN) 5C0,1500,500
C
C-----------------------------------------------------------------------
C
C-----BRANCH ON POSITIVE.
 1530 IF(ISIGN) 500,500,1500
C
C-----------------------------------------------------------------------
C
C-----BRANCH ON OVERFLOW. RESET OVFL LATCH.
 1540 IF(IOVFL) 500,500,1541
 1541 IOVFL=0
      GO TO 1500
C
C-----------------------------------------------------------------------
C
C-----READ FROM INPUT DEVICE INTO (EA).
 1600 NOCDS = NOCOS + 1
      IF(NOCOS-NORDS) 1601,1601,1605
```

```
 1601 K = NCCDS + FDATA - 1
      LOC1(EA) = DATA1(K)
      LOC2(EA) = DATA2(K)
      GO TO 500
 1605 CALL RDR60
      IF(ERRCT(1)) 385,500,385
  385 JERR=4
      GO TO 500
C
C------------------------------------------------------------------------
C
C-----WRITE (EA) ONTO OUTPUT DEVICE.
 1610 CALL DECEB(CEAR,KKBUF)
      WRITE(NC,1615)KKBUF
 1615 FORMAT(1H ,7A1)
      NWTR=NWTR+1
      IF(NWTR-30) 1617,1617,500
 1617 NSAV1(NWTR)=CEAR1
      NSAV2(NWTR)=CEAR2
      GO TO 500
C
C------------------------------------------------------------------------
C
C-----STOP.
 1770 IAR = PC
      JERR=1
      GO TO 500
C
C------------------------------------------------------------------------
C
C
C-----TRACE IF SSW 1 ON
500   TCNTR=TCNTR+1
      IF(TCNTR-25) 510,510,501
501   CALL DATSW(1,J)
      GO TO (510,520),J
  510 MSW=1
C
C
      IF(LCTR) 570,560,570
  560 LCTR=1
      WRITE(NC,561)
 561  FORMAT(' XEQNC  ADDR    C(ADDR)     MNEMONIC      C(XR)        EA',
     C       '      C(EA)       C(ACC)        C(XR)         C(EA)',
     C       '     SIGN   OVFL',/)
C-----GET C(ADDR)
  570 CALL DECEB(MREG,IIBUF)
      NEUMO(1) = NUTBL(2*INSTR-1)
      NEUMO(2) = NUTBL(2*INSTR)
C-----GET C(XR)
      IF(TAG) 580,580,585
 580  DO 582 I=1,7
      JJBUF(I) = TABLE(12)
      MMBUF(I) = TABLE(12)
 582  CONTINUE
      GO TO 590
  585 CALL DECEB(CXR,JJBUF)
      NXRG1 = XR1(TAG)
      NXRG2 = XR2(TAG)
      CALL DECEB(NXREG,MMBUF)
C-----GET C(EA)
  590 CALL DECEB(CEAR,KKBUF)
      NNREG(1)=LOC1(EA)
      NNREG(2)=LOC2(EA)
      CALL DECEB(NNREG,NNBUF)
C-----GET C(ACC)
```

```
      CALL DECEB(AREG,LLBUF)                                            SIMRN395
C                                                                       SIMRN396
      EA=EA-1                                                           SIMRN397
      WRITE(NO,596) RNTIM(2),PC,IIBUF,NEUMO,TAG,ADDR,JJBUF,EA,KKBUF,    SIMRN398
     C               LLBUF,MMBUF,NNBUF,ISIGN,IOVFL                      SIMRN399
 596  FORMAT(1H ,I5,2X,I4,4X,7A1,4X,2A2,I1,1X,I3,4X,7A1,4X,I4,3X,7A1,5X,SIMRN400
     C      7A1,4X,7A1,5X,7A1,6X,I2,5X,I2 )                             SIMRN401
C                                                                       SIMRN402
      GO TO 521                                                         SIMRN403
C                                                                       SIMRN404
C                                                                       SIMRN405
C-----SKIP LINE WHEN DATSW TURNED OFF                                   SIMRN406
520   GO TO(512,521),MSW                                                SIMRN407
512   MSW=2                                                             SIMRN408
      WRITE(NO,555)                                                     SIMRN409
555   FORMAT(1H )                                                       SIMRN410
  521 IF(JERR) B00,523,800                                              SIMRN411
C-----FLUSH TO NEXT JOB (SIM610) IF SSW 11 ON                           SIMRN412
C      ( OPERATOR JUDGES TIME EXCESSIVE -IF PRINTING IN LOOP            SIMRN413
C       WILL NOT BE STOPPED IN REASONABLE TIME BY COUNTER. )            SIMRN414
 523  CALL DATSW(11,J)                                                  SIMRN415
      GO TO (530,600),J                                                 SIMRN416
C                                                                       SIMRN417
C-----BEGIN NEXT MACHINE CYCLE                                          SIMRN41B
  600 PC = IAR                                                          SIMRN419
C-----FLUSH TO NEXT PROGRAM  (AFTER DUMP) IF RUN TIME EXCESSIVE.        SIMRN420
      IF(RNTIM(2)-MAXRT) 1000,1000,530                                  SIMRN421
 530  JERR=3                                                            SIMRN422
  800 RETURN                                                            SIMRN423
      END                                                               SIMRN424
// DUP                                                                  SIMRN425
*DELETE             SIMRN                                               SIMRN426
*STORE      WS  UA  SIMRN                                               SIMRN427
```

```
// JOB
// *
// * PROGRAM TO COMPUTE AND PRINT GRADING INFORMATION AND DUMP CORE.
// *
// FOR
*NAME DUMPG
*IOCS(CARD,1403PRINTER,DISK)
*LIST SYMBOL TABLE
*EXTENDED PRECISION
*ONE WORD INTEGERS
C-----SINCE INTEGER SIZE NOT ADEQUATE,
      REAL RWGTM,RWGPL,PPTT
      INTEGER REG(2),REG1,REG2
      INTEGER KBUFF(7)
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMO(2)
      INTEGER IOBUF(48),NAME(32),ERRCT(5)
      INTEGER TABLE(16)
      INTEGER RNTIM(2),PROGL
      INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
      INTEGER NSAV1(30),NSAV2(30)
      INTEGER STORT,STOPL
      INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
      INTEGER NRDSR(10),LOCRD(10)
      INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCC,PTCW(10),PTWD
      INTEGER PCGRT,PCGPL
      INTEGER FDATA,PDSPT(3)
      INTEGER RDATA(14)
      INTEGER PTSR,PTSA,PTSW,PTS
      INTEGER FILNO,PC,LINE(70),DATA(212),DATA1(106),DATA2(106)
      INTEGER NAM(31),RAWGR
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMO,IOBUF,NAME
      COMMON ERRCT
      COMMON NI,NC,TABLE,JERR
      COMMON I,J,K,L,M
      COMMON INIT
      COMMON NSTUD,NPROB
      COMMON RNTIM,PROGL,NOCDS
      COMMON NANSW,NSAV1,NSAV2
      COMMON IDUMY,STORT,STOPL,ANS1,ANS2,NORDS,NRGPS,PDSPT
      COMMON NRDSR,LOCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCD
      COMMON PTWD,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
      COMMON PTSR,PTSA,PTSW,PTS
      COMMON FILNO,PC,IOVFL,LINE,DATA
      EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
      EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
      EQUIVALENCE (PDSPT(1),NPPTR),(PDSPT(2),NPPTA),(PDSPT(3),NPPTW)
      EQUIVALENCE (KBUFF(1),LINE(1))
      EQUIVALENCE (REG(1),REG1),(REG(2),REG2)
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
      EQUIVALENCE (NAM(1),NAME(1))
      DEFINE FILE 2(800,40,U,NXRCC)
C
C-----EXECUTION COMPLETE
C
C--
C------PTSR       = POINTS RECIEVED FOR READING.
    4 PTSR = 0
C------PTSA       = POINTS RECIEVED FOR ANSWERS + ANSWER LOCATIONS.
      PTSA = 0
C------PTSW       = POINTS RECIEVED FOR WRITING ANSWERS.
      PTSW = 0
C------FDATA   = FIRST LOCATION IN 'DATA' FROM WHICH INPUT DATA WAS
C       'READ'  (FOLLOWING READS WERE FROM SUCCESIVE LOCATIONS) .
C       DATA CARD 9, WORD 5.
```

```
      ID=FDATA-1
C------NRGPS    = NO OF GROUPS OF READ AREAS
      DO 704 I=1,NRGPS
C------LOCRD(I) = FIRST LOCATION OF ITH GROUP TO BE READ INTO.
C       DATA CARD 3
      IAR = LOCRD(I) + 1
C------NRDSR(I) = NO OF READS REQUIRED IN ITH GROUP.
C       DATA CARD 2
      K=NRDSR(I)
      DO 704 J=1,K
      ID=ID+1
      IF(LOC1(IAR)-DATA1(ID)) 704,702,704
  702 IF(LOC2(IAR)-DATA2(ID)) 704,703,704
C------PTCR(I) = NO OF POINTS FOR READING EACH CARD IN ITH GROUP
C       DATA CARD 5
  703 PTSR=PTSR+PTCR(I)
C------ITH GROUP CONSISTS OF CONSECUTIVE LOCATIONS.
  704 IAR=IAR+1
C------NORDS    = NO OF READS REQUIRED
      IF(NORDS-NOCDS) 706,705,706
C------PTCRN    = NO OF POINTS FOR CORRECT NO OF READS.
C       DATA CARD 9, WORD 1
  705 PTSR=PTSR + PTCRN
  706 CONTINUE
      IF(PTSR+PTCRN-NPPTR) 710,709,709
  709 PTSR = PTSR + PTCRN
  710 I = 0
      DO 730 K = 1,5
      L = 1
C------NANSR(K) = NO OF ANSWERS IN K'TH ANSWER GROUP.
C       DATA CARD 4, WORDS 6 TO 10
 7101 IF(L-NANSR(K)) 7105,7105,730
C------LCANS(I) = LOCATIONS IN WHICH ANSWERS ARE TO BE PUT
C       DATA CARD 4, WORDS 1 TO 5
 7105 IAR = LCANS(K) + L
      IF(LOC1(IAR)-30000) 712,711,712
  711 IF(LOC2(IAR)-30000) 712,713,712
C------PTCA(I) = NO OF POINTS FOR AFFECTING ANSWER LOCATIONS
C       DATA CARD 6
  712 PTSA= PTSA+ PTCA(K)
  713 I = I + 1
      L = L + 1
      IF(I-30) 7135,7135,728
 7135 IF(LOC1(IAR)-ANS1(I)) 716,714,716
  714 IF(LOC2(IAR)-ANS2(I)) 716,715,716
C------PTCC(I) = NO OF POINTS FOR CORRECT ANSWERS
C       DATA CARD 7
  715 PTSA= PTSA+ PTCC(K)
  716 IF(I-NANSW) 7165,7165,724
 7165 IF(NSAV1(I)-ANS1(I)) 719,717,719
  717 IF(NSAV2(I)-ANS2(I)) 719,718,719
C------PTCW(I) = NO OF POINTS FOR PRINTING CORRECT ANS. IN CORR.ORDER
C       DATA CARD 8
  718 PTSW= PTSW+ PTCW(K)
C------NANSW    = NO OF ANSWERS WRITTEN
  719 DO 722 J= 1,NANSW
C------NSAV1,2(I) = ANSWERS WRITTEN BY PROGRAM ( FIRST 10 )
C------ANS1,2(I) = CORRECT ANSWERS
      IF(NSAV1(J)-ANS1(I)) 721,720,721
  720 IF(NSAV2(J)-ANS2(I)) 721,723,721
  721 IF(J-30) 722,724,724
  722 CONTINUE
      GO TO 724
C------PTWO     = NO OF POINTS FOR PRINTING CORRECT ANS. IN ANY ORDER
C       DATA CARD 9, WORD 3
  723 PTSW= PTSW+ PTWO
```

```
C------NANS     = NO CF ANSWERS RECUIRED
C       DATA CARD 9, WORD 4.
  724 DO 7265 K1=1,5
      L1 = 1
 7241 IF(L1-NANSR(K1)) 7242,7242,7265
 7242 IAR = LCANS(K1) + L1
      L1 = L1 + 1
      IF(LOC1(IAR)-ANS1(I)) 726,725,726
  725 IF(LOC2(IAR)-ANS2(I)) 726,727,726
  726 GO TO 7241
 7265 CONTINUE
      GO TO 728
C------PTCO     = NO OF POINTS FOR CORRECT ANS IN CORR LCCS IN ANY CRDER.
C       DATA CARD 9, WORD 2
  727 PTSA= PTSA+ PTCO
  728 GO TO 7101
  730 CONTINUE
C
C------PTS       = TOTAL NO. CF POINTS RECIEVEC.
      PTS = PTSR + PTSA + PTSW
C
C
C     NPPTT = NO. OF POSSIBLE PCINTS - TCTAL.
      NPPTT = NPPTR + NPPTA + NPPTW
      PPTT = NPPTT
      RWGTM = 1.
      RWGPL = 1.
      IF(PTS-NPPTT) 742,750,750
C------DO NCT COUNT TIME OR LENGTH BETTER THAN STANDARD IF FULL
C       POINTS WERE NOT EARNED.
  742 IF(RNTIM(2)-STDRT) 744,744,743
  743 RWGTM = RWGTM*STDRT/RNTIM(2)
  744 IF(PROGL-STDPL) 760,760,745
  745 RWGPL = RWGPL*STDPL/PROGL
      GO TO 760
  750 RWGTM = RWGTM*STDRT/RNTIM(2)
      RWGPL = RWGPL*STDPL/PROGL
  760 RAWGR= ( 100-PCGRT-PCGPL+PCGRT*RWGTM+PCGPL*RWGPL)/PPTT*10.*PTS
      GO TO(780,770,770,780),JERR
  770 RAWGR = 3*RAWGR/4
  780 CONTINUE
C
C
C-----ROUTINE TO DUMP PSUEDO-CORE TO PRINTER.
C
C     LOC IS 1000 WORD PSUEDO-CORE.
C     DUMP IS TEN 7I1 INTEGERS PER LINE.
C     ALL OF CORE IS DUMPED.
C
      WRITE(NO,799) NAME,NPROB
  799 FORMAT(1H0,08X,32A2,12X,'PROBLEM NO.',I4)
      IF(NPROB-4)7995,8205,7995
8205  NANS=0
7995  GO TO(801,803,805,807),JERR
  801 WRITE(NO,802)
  802 FORMAT(1H0,'EXECUTION COMPLETE')
      GO TO 820
  803 WRITE(NO,804) PC
  804 FORMAT(1H0,'EXECUTION TERMINATED BY INVALID INSTRUCTION AT ',I3)
      GO TO 820
  805 WRITE(NO,806)
  806 FORMAT(1H0,'EXECUTION TERMINATED DUE TO EXCESSIVE RUN TIME')
      GO TO 820
  807 WRITE(NO,808) PC,EA
  808 FORMAT(1H0,'EXECUTION TERMINATED BY INSTR. AT ',I3,' ATTEMPTING TO
     1 READ 1ST CARD OF NEXT PROG. INTO ',I3)
```

```
820 WRITE(NC,811) RNTIM(2),STLRT,PRCGL,STOPL,NCOCS,NCRDS,NANSW,NANS
811 FORMAT(1H0,10X,'RUNTIME',14X,'LENGTH OF DECK',C8X,'NO OF CARDS '
   1,'READ',06X,'NC OF ANSWERS WRITTEN'/4(05X,'YOURS',C6X,'STANDARD'),
   2/,3X,8(I6,C6X)/)
    WRITE(NC,815) PTSR,NPPTR,PTSA,NPPTA,PTSW,NPPTW,PTS,NPPTT,RAWGR
815 FORMAT(1H0,C3X,'POINTS RECEIVED FOR---'/05X,'READING DATA',11X,
  1 'ANS IN CORR LOCATIONS',C4X,
  2'WRITING ANSWERS',C9X,'TOTAL',19X,'RAW',/
  34(05X,'YOURS',C6X,'STANDARD'),4X,'GRADE',/,3X,9(I6,6X)/)
    CALL DECE8(AREG,KBUFF)
    WRITE(NC,813) ISIGN,IOVFL,K8UFF
813 FORMAT(1H ,'SIGN ',I2,3X,'OVERFLOW ',I2,3X,'ACCUMULATOR',2X,7A1)
C
C-----PRINT INDEX REGISTERS.
    IAR=C
    J = 8
    DO 860 K=1,9
    IAR =IAR + 1
    REG1=XR1(IAR)
    REG2=XR2(IAR)
C-----CLEAR UNUSED INDEX REGISTERS
    IF(REG1-25000) 831,832,831
832 IF(REG2-25000) 831,833,831
833 DO 834 L=1,7
    LINE(J)=TABLE(12)
834 J=J+1
    GO TO 860
831 CALL DECE8(REG,LINE(J))
    J = J + 7
860 CONTINUE
    WRITE(NC,843) (LINE(J),J=8,70)
843 FORMAT(1H ,10X,5HI/RS ,9(3X,7A1),/)
C-----DUMP PSEUDO - CORE.
    IAR=0
    DO 83C I=1,10C
    J=1
    M=0
    DO 88C K=1,10
    IAR =IAR + 1
    REG1=LOC1(IAR)
    REG2=LOC2(IAR)
    IF(REG1-300C0) 851,852,851
852 IF(REG2-300C0) 851,853,851
853 DO 854 L=1,7
    LINE(J)=TABLE(12)
854 J=J+1
    M=M+1
    GO TO 880
851 CALL DECE8(REG,LINE(J))
    J = J + 7
880 CONTINUE
C     DO NOT PRINT LINE IF ALL LOCATIONS IN IT UNAFFECTED BY PROGRAM.
    IF(M-9) 821,821,830
821 J=IAR-1C
    WRITE(NC,822) J,LINE
822 FORMAT(1H ,I3,2X,10(3X,7A1))
830 CONTINUE
C-----ERROR TRAP
C
C------IF FINAL GRADE RUN, WRITE GRADE INFO ON FILE.
C-----IF INITIALIZATION,GOTO  INI2G, IF STUD. PROG. GO TO LOAD NEXT.
    IF(INIT) 885,881,890
881 READ(2'1) NFILE
    NFILE = NFILE + 1
    WRITE(2'1) NFILE
    WRITE(2'NFILE) NPROB,NSTUD,JERR,RNTIM(2),PROGL,PTSR,PTSA,PTSW,NAM.
```

```
     1RAWGR
  885 CALL LINK(LOADP)
  890 CALL LINK(INI2G)
      END
// DUP
*DELETE             DUMPG
*STORECI    WS  UA  DUMPG 0001
*FILES(2,SMSTU)
```

```
// *
// *PROGRAM TC INITIALIZE GRADER.
// *
// FOR
*NAME INITG
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
      INTEGER A(2205),INPUT(160),CRDIN(78),NREM(77),CATA(212)
      INTEGER NRDSR(1C),TABLE(16)
      INTEGER ERR,EA
      INTEGER DATA1(106),DATA2(106)
      INTEGER FDATA
      COMMON A,INPUT,NREM,DATA
      EQUIVALENCE (NPROB,A(2140)),(CRCIN(1),INPUT(69)),(TABLE(1),A(2116)
     1),(INIT,A(2138)),(NRDSR(1),CRDIN(1)),(NCRCS,INPUT(64)),(NRGPS,INPU
     2T(65))
      EQUIVALENCE (EA,A(2C25))
      EQUIVALENCE(LGC11,A(1)),(LDC12,A(1001))
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),CATA(107))
      EQUIVALENCE (ERR,A(2109))
      EQUIVALENCE (NI,A(2114)),(NC,A(2115))
      EQUIVALENCE(FDATA,CRDIN(75))
      DEFINE FILE 1(24,16C,U,NXREC)
      DEFINE FILE 5(12,106,U,NXRDC)
    1 INIT = 1
      DO 8 I=1,160
    8 INPUT(I) = 0
      NI=2
      NO=5
      READ(NI,13) TABLE,NDTST
   13 FORMAT(16A1,I1)
      CALL DATSW(3,J)
      GO TO(5C0,1C),J
  500 READ(NI,11) NPROB,NRDSR(1),FDATA
      GO TC 600
   10 READ(NI,11) NPROB
C     READ(NI,11) NRDSR,LGCRD,LCANS,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCC,
C    1PTWC,NANS,FDATA
      READ(NI,11) CRDIN
  600 CONTINUE
   11 FORMAT(10(I6,2X))
      DO 2C I = 1,1C
      NRGPS=I-1
      K = NRDSR(I)
      IF(K) 2C,21,2C
   20 NORCS = NORDS + K
      NRGPS = 10
C     WRITE(1'NPROB) IDUMY,STDRT,STOPL,ANS1,ANS2,NORCS,NRGPS,NRDSR,
C    1LOCRD,LCANS,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCC,PTWC,NANS,FDATA,PCSPT
   21 WRITE(1'NPROB) INPUT
      IF(NCTST) 16,16,14
   14 IF(NDTST-6) 15,15,16
   15 READ(5'2*NDTST-1) CATA
      GO TC 19
   16 EA=1
      DO 4 I=1,106
      CALL RDR6C
      CATA1(I)=LOC11
      CATA2(I)=LOC12
      IF(ERR) 3,2,3
```

```
    3 PAUSE 7009
      I = I - 1
    2 LOC12 = IABS(LCC12)
    4 WRITE(NC,17) LOC11,LOC12
   17 FORMAT(1H ,I4,I3)
   19 CALL RDR60
      CALL LINK(LOADP)
      END
// DUP
*DELETE              INITG
*STORECI     WS  UA  INITG 0001
*FILES(1,FSTDG),(5,SIMDT)
```

```
// JOB
// *
// * PROGRAM TO FINISH PROBLEM INITIALIZATION PROCEDURE.
// *
// FOR
*NAME INI2G
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
      INTEGER ERROR
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCCC,NEUMO(2)
      INTEGER IOBUF(48),NAME(32),ERRCT(5)
      INTEGER TABLE(16)
      INTEGER RNTIM(2),PROGL
      INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
      INTEGER NSAV1(30),NSAV2(30)
      INTEGER STORT,STOPL
      INTEGER NRDSR(10),LCCRD(10)
      INTEGER ANS1(30),ANS2(30),NANS,LCANS(5),NANSR(5)
      INTEGER RDATA(14)
      INTEGER PTSR,PTSA,PTSW,PTS
      INTEGER FDATA,PCSPT(3)
      INTEGER PCGRT,PCGPL
      INTEGER PTCR(10),PTCRN,PTCA(10),PTCC(10),PTCC,PTCW(10),PTWO
      INTEGER FILNO,PC,LINE(70),DATA(212)
      INTEGER INPUT(160),         IDUMY(1)
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCCC,NEUMO,IOBUF,NAME
      COMMON ERRCT
      COMMON NI,NO,TABLE,JERR
      COMMON I,J,K,L,M
      COMMON INIT
      COMMON NSTUD,NPROB
      COMMON RNTIM,PROGL,NOCOS
      COMMON NANSW,NSAV1,NSAV2
      COMMON IDUMY,STORT,STOPL,ANS1,ANS2,NORCS,NRGPS,PCSPT
      COMMON NRDSR,LCCRD,LCANS,NANSR,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCO
      COMMON PTWO,NANS,FDATA,MAXRT,PCGRT,PCGPL,RDATA
      COMMON PTSR,PTSA,PTSW,PTS
      COMMON FILNO,PC,IOVFL,LINE,DATA
      EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
      EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
      EQUIVALENCE (POSPT(1),NPPTR),(PCSPT(2),NPPTA),(PCSPT(3),NPPTW)
      EQUIVALENCE (NI,NOCOS),(NWTR,NANSW)
      EQUIVALENCE (LOC11,LOC1(1)),(LOC12,LOC2(1))
      EQUIVALENCE (ERRCT(1),ERROR)
      EQUIVALENCE (INPUT(1),IDUMY(1))
      DEFINE FILE 1(24,160,U,NXREC)
    1 IF(INIT-2) 2,101,101
C----- PUT RESULTS OF RUN OF STANDARD INTO 'STANDARD' VARIABLES.
    2 INIT = 2
      I = 0
      DO 730 K =1,5
      J = 1
  710 IF(J-NANSR(K)) 720,720,730
  720 IAR = LCANS(K) + J
      I = I + 1
      IF(I-30) 725,725,730
  725 ANS1(I) = LOC1(IAR)
      ANS2(I) = LOC2(IAR)
      J = J + 1
      GO TO 710
```

```
  730 CONTINUE
      NANS = NANSW
      IF (NANS) 770,770,740
  740 DO 760 I=1,NANS
      ANS1(I) = NSAV1(I)
      ANS2(I) = NSAV2(I)
      IF(I-30) 760,770,770
  760 CONTINUE
  770 STDRT = RNTIM(2)
      STDPL = PRDGL
      CALL LINK(DUMPG)
C------PUT RESULTS OF SECOND PASS THRU DUMPG INTO STD VARIABLES
C        AND PUT STANDARD DATA ON FILE.
  101 NPPTR = PTSR
      NPPTA = PTSA
      NPPTW = PTSW
C     WRITE(1'NPROB) IDUMY,STDRT,STDPL,ANS1,ANS2,NCRDS,NRGPS,NRDSR,
C    1LOCRD,LCANS,PTCR,PTCA,PTCC,PTCW,PTCRN,PTCC,PTWC,NANS,FDATA,PCSPT
      WRITE(1'NPROB) INPUT
      PAUSE 3333
C     RETURN TO INITIALIZE ANOTHER PROBLEM IF SENSE SWITCH 2 ON.
      CALL DATSW(2,J)
      GO TO(200,777),J
  200 CALL LINK(INITG)
  777 STOP 7777
      END
// DUP
*DELETE             INI2G
*STORECI    WS  UA  INI2G 0001
*FILES(1,FSTDG)
```

```
// JOB
// FOR
*LIST ALL
*ONE WORD INTEGERS
*EXTENDED PRECISION
      SUBROUTINE RDSTD
      INTEGER A(2205),INPUT(160),NREM(75)
      COMMON A,INPUT,NREM
      EQUIVALENCE(NPROB,A(2140))
      EQUIVALENCE (MAXRT,INPUT(144))
C
C-----THIS ROUTINE READS THE FILE MADE FROM THE STANDARD FOR THE
C     PROBLEM THE STUDENT IS ATTEMPTING.
C
1     MAXRT = 5000
      IF(NPROB) 5,5,2
    2 IF(NPROB-24) 10,10,5
    5 DO 8 I = 1 , 103
    8 INPUT(I) = 0
      NPROB = 0
      RETURN
   10 READ(1'NPROB) INPUT
      RETURN
      END
// DUP
*DELETE             RDSTD
*STORE      WS  UA  RDSTD
```

```
// JOB
// DUP
// FOR
* LIST ALL
*EXTENDED PRECISION
*ONE WORD INTEGERS
      SUBROUTINE RDR60
      INTEGER KBUFF(7)
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMC(2)
      INTEGER IOBUF(80),ERRCT(5)
      INTEGER TABLE(16)
      INTEGER LOC1(1000),LOC2(1000),XR1(9),XR2(9)
      INTEGER ERROR
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMC,IOBUF,ERRCT
      COMMON NI,NO,TABLE
      EQUIVALENCE (LOC(1),LOC1(1)),(LOC(1001),LOC2(1))
      EQUIVALENCE (XR(1),XR1(1)),(XR(10),XR2(1))
      EQUIVALENCE (ERRCT(1),ERROR)
C
C
C     ROUTINE TO SIMULATE READ INSTRUCTION
C     ON RETURN--ERROR IS SET  -1 IF ASTERISK CARD READ
C                               0  IF NO ERROR
C                              +1  IF INVALID DATA
C     PSUEDO-CORE LOCATION IS NOT ALTERED IF ASTERISK CARD IS READ,
C     OR IF INVALID DATA IS READ.
C

      ERROR=0
      READ(NI,11) (IOBUF(I),I=1,48)
   11 FORMAT(16A1,32A2)
      NOCDS=NOCDS+1
C     RETURN IF MONITOR(ASTERISK) CARD.
      IF(IOBUF(1)-TABLE(15)) 30,20,30
   20 ERROR=(-1)
      RETURN
C     CONVERT 7A1 TO 7I1.
C     TEST FOR CONVERSION ERROR.
   30 DO 21 N=12,14
      IF(TABLE(N)-IOBUF(1)) 21,25,21
   21 CONTINUE
      IF(TABLE(11)-IOBUF(1)) 22,23,22
   22 ERROR=1
      GO TO 50
   23 KBUFF(1)=-1
      GO TO 26
   25 KBUFF(1)=1
   26 DO 29 N=2,7
      DO 28 J=1,10
      IF(TABLE(J)-IOBUF(N)) 28,29,28
   28 CONTINUE
      GO TO 22
   29 KBUFF(N) = J-1
C     PACK 7I1 INTO 2I3 AND STORE INTO PSUEDO-CORE.
      LOC1(EA) = ((KBUFF(2)*10+KBUFF(3))*10+KBUFF(4))*KBUFF(1)
      LOC2(EA) = ((KBUFF(5)*10+KBUFF(6))*10+KBUFF(7))*KBUFF(1)
   50 RETURN
      END
// DUP
*DELETE             RDR60
*STORE      WS  UA  RDR60
```

```
// JOB
// FOR
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST ALL
      SUBROUTINE DECEB(REG,VECTR)
      INTEGER REG(2),VECTR(7),DATA,BUF
      INTEGER CORE(2020),RCORE(94),TABLE(16)
      COMMON CORE,ISIGN,RCORE,TABLE
C----
    1 VECTR(1) = TABLE(12)
      DATA = REG(1)
      I = 2
      DO 60 J=1,2
      IF(DATA) 10,25,25
   10 VECTR(1) = TABLE(11)
      DATA = -DATA
   25 L = 100
      DO 50 K=1,3
      BUF = DATA/L
      IF(BUF-9) 30,30,100
   30 VECTR(I) = TABLE(BUF+1)
      DATA = DATA - BUF*L
      L = L/10
   50 I = I + 1
   60 DATA = REG(2)
      RETURN
  100 DO 110 I=2,7
  110 VECTR(I) = TABLE(15)
      RETURN
      END
// DUP
*DELETE             DECEB
*STORE      WS  UA  DECEB
```

```
// JOB
// DUP
*DELETE             DWADD
// FOR
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST ALL
      SUBROUTINE DWADD(A,B,C,IOVFL)
      INTEGER A(2),B(2),C(2),CARRY
      INTEGER LOC(2000),XR(18),AREG(2),TAG,ADDR,EA,OPCOD,NEUMO(2)
      INTEGER IOBUF(80),ERRCT(5)
      COMMON LOC,XR,AREG,ISIGN,INSTR,TAG,ADDR,EA,OPCOD,NEUMO,IOBUF,ERRCT
C       THIS ROUTINE PERFORMS DOUBLE-WORD DECIMAL ADDITION
C       SUCH THAT C = A + B
    1 IOVFL = 0
   10 C(2) = A(2) + B(2)
      CARRY = C(2)/1000
      C(2) = C(2) - CARRY*1000
      C(1) = A(1) + B(1) + CARRY
      CARRY = C(1)/1000
      C(1) = C(1) - CARRY*1000
      IF(CARRY) 25,30,25
   25 IOVFL = 1
      ISIGN = CARRY
      RETURN
C       IF NO CARRY CHECK WHETHER SIGNS OF UPPER + LOWER HALF DISAGREE.
C       (IF CARRY A + B MUST HAVE HAD SAME SIGN.)
   30 M = 1
      I = (C(1)/IABS(C(1)))*(C(2)/IABS(C(2)))
      IF(I) 32,40,40
   32 IF(C(1)) 33,40,35
   33 M = -1
   35 C(1) = C(1)-M
      C(2) = C(2) + M*1000
   40 CALL LATCH(C)
      RETURN
      END
// DUP
*STORE      WS  UA  DWADD
```

```
// JOB     0015
// FOR
*LIST ALL
*EXTENDED PRECISION
*ONE WORD INTEGERS
      SUBROUTINE LATCH(REG)
      INTEGER REG(2)
      INTEGER CORE(2020)
      COMMON CORE,ISIGN
C
C     THIS ROUTINE SETS THE SIGN INDICATOR, ISIGN, TO -1,0,+1
C     ACCORDING TO THE SIGN OF THE DATA IN A.
C
C     EXAMPLES...
C    REG(1)   REG(2)  ISIGN
C     -999     000     -1      DATA IS NEGATIVE
C      000    -999     -1      DATA IS NEGATIVE
C      000     000      0      DATA IS ZERO
C      000     999      1      DATA IS POSITIVE
C      999     000      1      DATA IS POSITIVE
C
      IF ( REG(1) )  30,20,50
   20 IF ( REG(2) )  30,40,50
   30 ISIGN=-1
      RETURN
   40 ISIGN=0
      RETURN
   50 ISIGN=1
      RETURN
      END
// DUP
*DELETE             LATCH
*STORE      WS  UA  LATCH
```

```
// JOB                                                                    INTFG001
// *                                                                      INTFG002
// * PROGRAM TO INITIALIZE STUDENT GRADE FILE AND CLEAR STANDARD FILE.    INTFG003
// *                                                                      INTFG004
// DUP                                                                    INTFG005
*DELETE             INTFG                                                 INTFG006
// FOR                                                                    INTFG007
*NAME INTFG                                                               INTFG008
*ONE WORD INTEGERS                                                        INTFG009
*EXTENDED PRECISION                                                       INTFG010
*LIST ALL                                                                 INTFG011
*IOCS(DISK)                                                               INTFG012
      INTEGER ONE(160),TWO(40)                                            INTFG013
      DEFINE FILE 1(24,160,U,NXREC)                                       INTFG014
      DEFINE FILE 2(800,40,U,NXRCC)                                       INTFG015
    1 DO 10 I=1,160                                                       INTFG016
   10 ONE(I) = 0                                                          INTFG017
      DO 20 I=1,40                                                        INTFG018
   20 TWO(I) = 0                                                          INTFG019
      NXREC = 1                                                           INTFG020
      NXRCC = 1                                                           INTFG021
      TWO(1) = 1                                                          INTFG022
      WRITE(2'NXRCC) TWO                                                  INTFG023
      TWO(1) = 0                                                          INTFG024
      DO 30 I=1,24                                                        INTFG025
   30 WRITE(1'I) ONE                                                      INTFG026
      DO 40 I=2,800                                                       INTFG027
   40 WRITE(2'NXRCC) TWO                                                  INTFG028
      CALL EXIT                                                           INTFG029
      END                                                                 INTFG030
// XEQ       L 01                                                         INTFG031
*FILES(1,FSTDG),(2,SMSTU)                                                 INTFG032
```

```
// JOB
// *
// * PROGRAM TO READ A SET OF DATA FOR THE STUDENT PROGRAMS TO 'READ'
// * INTO A FILE ( ONE OF 12.)
// *
// FOR
*NAME INDFG
*IOCS(CARD,DISK,1403 PRINTER)
*EXTENDED PRECISION
*ONE WORD INTEGERS
*LIST SOURCE PROGRAM
*LIST SUBPROGRAM NAMES
*LIST SYMBOL TABLE
      INTEGER BUFF(2),OBUFF(7)
      INTEGER A(2165),INPUT(160),NREM(77),DATA(212)
      INTEGER DATA1(106),DATA2(106)
      INTEGER TABLE(16)
      INTEGER ERR,EA
      COMMON A,INPUT,NREM,DATA
      EQUIVALENCE (TABLE(1),A(2116))
      EQUIVALENCE(LOC11,A(1)),(LOC12,A(1001))
      EQUIVALENCE (NI,A(2114)),(NO,A(2115))
      EQUIVALENCE (ERR,A(2109)),(EA,A(2025))
      EQUIVALENCE (DATA1(1),DATA(1)),(DATA2(1),DATA(107))
      DEFINE FILE 5(12,106,U,NXRDC)
    1 NI=2
      NO=5
      READ(NI,13) TABLE,NOTST
   13 FORMAT(16A1,I1)
      DO 8 I=1,212
    8 DATA (I) = 0
      EA=1
      DO 5 I=1,106
    2 CALL RDR60
      IF(ERR) 3,4,3
    3 PAUSE 7009
      GO TO 2
    4 DATA1(I)=LOC11
      DATA2(I)=LOC12
      BUFF(1) = LOC11
      BUFF(2) = LOC12
      CALL DECEB(BUFF,OBUFF)
      WRITE(NO,11) I,OBUFF
   11 FORMAT(1H ,I3,3X,7A1)
    5 CONTINUE
      WRITE(5'2*NOTST-1) DATA
      STOP 7777
      END
// XEQ        L 01
*FILES(5,SIMDT,0015)
0123456789- ++* 2
```

+000023	DATA2001
-003547	DATA2002
+354501	DATA2003
-000006	DATA2004
+002346	DATA2005
000000	DATA2006
+012345	DATA2007
-001278	DATA2008
+024035	DATA2009
-000023	DATA2010
+850043	DATA2011
+012005	DATA2012
+100000	DATA2013
+233245	DATA2014
-000156	DATA2015
-751000	DATA2016
-000245	DATA2017
+120345	DATA2018
-003486	DATA2019
000000	DATA2020
-001597	DATA2021
+043189	DATA2022
-000005	DATA2023
-100035	DATA2024
+145508	DATA2025
000000	DATA2026
-000135	DATA2027
+000020	DATA2028
-000009	DATA2029
+000045	DATA2030
-000054	DATA2031
+003498	DATA2032
+000009	DATA2033
-120005	DATA2034
-000010	DATA2035
+000005	DATA2036
000000	DATA2037
+000501	DATA2038
+000001	DATA2039
+010101	DATA2040
-100045	DATA2041
-000753	DATA2042
-000005	DATA2043
+000348	DATA2044
+000008	DATA2045
-156247	DATA2046
-036475	DATA2047
-102020	DATA2048
+012045	DATA2049
+000125	DATA2050
-010000	DATA2051
-500134	DATA2052
+000010	DATA2053
+000045	DATA2054
-245365	DATA2055
+360000	DATA2056
+000453	DATA2057
+000125	DATA2058
-000063	DATA2059
+003941	DATA2060
+987654	DATA2061
-853240	DATA2062
+500000	DATA2063
+000283	DATA2064

-004319	DATA2065
-001800	DATA2066
-004400	DATA2067
-000003	DATA2068
-000051	DATA2069
-000051	DATA2070
+000123	DATA2071
+000045	DATA2072
+000123	DATA2073
-001276	DATA2074
-001357	DATA2075
+000252	DATA2076
-000234	DATA2077
-000005	DATA2078
-000453	DATA2079
+000230	DATA2080
+000015	DATA2081
+000456	DATA2082
+499999	DATA2083
-999910	DATA2084
+888889	DATA2085
-000001	DATA2086
000000	DATA2087
+120450	DATA2088
000000	DATA2089
-112000	DATA2090
+100001	DATA2091
+000008	DATA2092
+102250	DATA2093
-000005	DATA2094
+000300	DATA2095
-000060	DATA2096
000000	DATA2097
-000245	DATA2098
+000035	DATA2099
+000202	DATA2100
+000005	DATA2101
+000023	DATA2102
+000008	DATA2103
+000025	DATA2104
+000010	DATA2105
+000014	DATA2106

```
PAGE   1

// JOB    0026 0015                    0015

LOG DRIVE   CART SPEC   CART AVAIL   PHY DRIVE
  0000        0026        0026         0001
  0001        0015        0015         0000

// ASM
*LIST
                          ***************************************************
                          *                                                 *
                          *    A M   (ASSEMBLER MONITOR)                    *
                          *                                                 *
                          ***************************************************
0122      01500000              ENT        AM       AM IS CALL ENTRY POINT
                          ***************************************************
0000      0000            BAMS  BSS   E    0        BEGINING OF AMS
0000 31 04063 1A3         IOAR1 DSA        DATFT    DISK PARAMETERS -- LENGTH
                          *                         *OF FILE IN WORDS, SECTOR
                          *                         *ADDRESS, AND NO OF SECTORS.
0003      0040            GTBL  BSS        64       GARBAGE TABLE IS 64 WORDS
0043      0040            ATB   BSS        64       ADDRESS TABLE IS 64 WORDS
                          ***************************************************
0084      0000            TINST BSS   E    0        TABLE OF INSTRUCTIONS
0084 0    1100                  DC         /1100    00000  WAIT
0085 0    1111                  DC         /1111    GO TO COMPUTE GRADE
0086 0    5700                  DC         /5700    00001  XIO
0087 0    9BDF                  DC         /9BDF    STORE=1. STANDARD ADDRESSING.
0088 0    0000                  DC         /0000    00010  SL
0089 0    0000                  DC         /0000    GO TO XEQ. HAS NO EA. F=SHORT
008A 0    0000                  DC         /0000    00011  SR
008B 0    0000                  DC         /0000    GO TO XEQ. HAS NO EA. F=SHORT
008C 0    0000                  DC         /0000    00100  LDS
008D 0    0000                  DC         /0000    GO TO XEQ. HAS NO EA. F=SHORT
008E 0    5700                  DC         /5700    00101  STS
008F 0    9BDF                  DC         /9BDF    STORE=1. STANDARD ADDRESSING.
0090 0    2200                  DC         /2200    00110  WAIT
0091 0    2222                  DC         /2222    GO TO VALID WAIT ROUTINE
0092 0    1100                  DC         /1100    00111  WAIT
0093 0    1111                  DC         /1111    GO TO COMPUTE GRADE
0094 0    5700                  DC         /5700    01000  BSI
0095 0    9BDF                  DC         /9BDF    STORE=1. STANDARD ADDRESSING.
0096 0    0000                  DC         /0000    01001  BSC
0097 0    0000                  DC         /0000    GO TO XEQ.
0098 0    1100                  DC         /1100    01010  WAIT
0099 0    1111                  DC         /1111    GO TO COMPUTE GRADE
009A 0    1100                  DC         /1100    01011  WAIT
009B 0    1111                  DC         /1111    GO TO COMPUTE GRADE
009C 0    0000                  DC         /0000    01100  LDX
009D 0    0088                  DC         /0088    GO TO XEQ. EX- IA,LONG IS EA.
009E 0    5500                  DC         /5500    01101  STX
009F 0    99DD                  DC         /99DD    STORE=1. STANDARD. EX- NO XR.
00A0 0    0000                  DC         /0000    01110  MDX
00A1 0    3038                  DC         /3038    (3=SPECIAL MDXLO)
00A2 0    1100                  DC         /1100    01111  WAIT
00A3 0    1111                  DC         /1111    GO TO COMPUTE GRADE
00A4 0    4600                  DC         /4600    10000  A
00A5 0    8ACE                  DC         /8ACE    STANDARD ADDRESSING.
00A6 0    4600                  DC         /4600    10001  AD
```

```
COA7 C  8ACE          DC        /8ACE     STANDARC ACCRESSING.                   AM 0255
COA8 0  4600          CC        /4600     10010  S                               AM 0260
COA9 C  8ACE          CC        /8ACE     STANDARC ACCRESSING.                   AM 0265
COAA 0  4600          DC        /4600     10011  SC                              AM 027C
00AB C  8ACE          DC        /8ACE     STANDARC ACCRESSING.                   AM 0275
00AC 0  4600          DC        /46C0     101C0  M                               AM 028C
COAD C  8ACE          DC        /8ACE     STANCARC ACCRESSING.                   AM 0285
00AE 0  4600          DC        /46C0     101C1  C                               AM 0290
COAF 0  8ACE          CC        /8ACE     STANCARC ACCRESSING.                   AM 0295
00B0 C  1100          DC        /11C0     10110  WAIT                            AM 0300
COB1 0  1111          DC        /1111     GO TO CCMPUTE GRACE                    AM 0305
00B2 C  11C0          DC        /1100     10111  WAIT                            AM 0310
00B3 0  1111          DC        /1111     GO TO CCMPUTE GRACE                    AM 0315
00B4 0  460C          DC        /46C0     11000  LC                              AM 0320
COB5 0  8ACE          DC        /8ACE     STANDARC ACCRESSING.                   AM 0325
00B6 0  46C0          CC        /46C0     110C1  LCC                             AM 0330
00B7 0  8ACE          CC        /8ACE     STANDARC ACCRESSING.                   AM 0335
00B8 C  5700          DC        /57C0     11010  STC                             AM 0340
COB9 0  9BDF          DC        /9BCF     STORE=1. STANCARD ACCRESSING.          AM 0345
COBA 0  5700          DC        /57C0     11011  STD                             AM 0350
00BB 0  9BDF          CC        /9BCF     STORE=1. STANCARC ACCRESSING.          AM 0355
00BC 0  4600          DC        /4600     11100  AND                             AM 0360
COBD 0  8ACE          DC        /8ACE     STANDARC ACCRESSING.                   AM 0365
00BE 0  4600          DC        /4600     11101  CR                              AM 0370
00BF C  8ACE          DC        /8ACE     STANDARC ACCRESSING.                   AM 0375
00C0 0  46C0          DC        /4600     11110  EOR                             AM 0380
C0C1 0  8ACE          DC        /8ACE     STANDARC ACCRESSING.                   AM 0385
00C2 0  110C          DC        /1100     11111  WAIT                            AM 0390
00C3 0  1111          DC        /1111     GO TO CCMPUTE GRACE                    AM 0395
                ***********************************************                  AM 0400
00C4    000C    IOCCB BSS  E    0         IOCC(S) TO SENSE DEVICE                AM 0405
00C4 0  0000          CC        0         UNUSEC                                 AM 0410
00C5 C  27C0          DC        /2700     CPU DISK                               AM 0415
00C6 0  0000          DC        0         UNUSEC                                 AM 0420
00C7 0  8F00          DC        /8F00     2310 FIRST DRIVE                       AM 0425
00C8 0  0C0C          DC        0         UNUSEC                                 AM 043C
C0C9 0  9700          DC        /9700     2310 SECOND CRIVE                      AM 0435
C0CA 0  0000          DC        0         UNUSEC                                 AM 0440
C0CB 0  9F00          DC        /9F00     2310 THIRD CRIVE                       AM 0445
00CC C  0C00          DC        0         UNUSEC                                 AM 0450
00CD C  A700          DC        /A700     231C FOURTH CRIVE                      AM 0455
                ***********************************************                  AM 0460
00CE 0  000C    LIST  DC        0         LIST OF MONITCR ENTRY PCINTS           AM 0465
00CF 0  0C28          DC        $PRET     PRE-OP I/O ERROR TRAP                  AM 0470
00D0 0  0081          DC        $PST1     POST-CP I/C ERROR TRAP L 1             AM 0475
C0D1 0  0C85          DC        $PST2     POST-CP I/O ERROR TRAP L 2             AM 0480
C0D2 C  0C89          DC        $PST3     POST-CP I/O ERROR TRAP L 3             AM 0485
C0D3 0  008D          DC        $PST4     POST-OP I/C ERROR TRAP L 4             AM 0490
00D4 C  0091          DC        $STOP     PROGRAM STCP KEY TRAP L 5              AM 0495
                ***********************************************                  AM 0500
00D5    002C    MBUF  DMES      'R '14XS E T  M O C E   S W   T C '             AM 0505
C0EB    0012          DMES        I N T   R U N 'R 'E                            AM 0510
                ***********************************************                  AM 0515
C0F4 1  03AC    TADDR DC        CGA       ADDRESS WITHIN COMMCN                  AM 0520
00F5 1  03AC          DC        CGA       ACDRESS WITHIN CALL TV                 AM 0525
00F6 1  03AC          DC        CGA       ADDRESS WITHIN FLOATING ACC            AM 0530
00F7 1  0276          DC        N414      ADDRESS WITHIN LIBF TV                 AM 0535
00F8 1  03AC          DC        CGA       ACDRESS WITHIN UNUSED CORE             AM 0540
```

```
00F9 1  03AC          DC     CGA      ADDRESS WITHIN ILS AREA
00FA 1  0276          DC     N414     ADDRESS WITHIN SUBROUTINES
00FB 1  03AC          DC     CGA      ADDRESS WITHIN AMS PROGRAM
00FC 1  0273          DC     N410     ADDRESS WITHIN MAINLINE
00FD 1  026D          DC     N402     ADDRESS WITHIN RESIDENT MON.
00FE 1  03AC          DC     CGA      ADDRESS WITHIN FIRST FOUR WDS
00FF 1  0200          DC     XEQ      EA WITHIN COMMON
0100 1  0362          DC     N510     EA WITHIN CALL TV
0101 1  0200          DC     XEQ      EA WITHIN FLOATING ACC
0102 1  034D          DC     N507     EA WITHIN LIBF TV
0103 1  0322          DC     N504     EA WITHIN UNUSED CORE
0104 1  031F          DC     N503     EA WITHIN ILS AREA
0105 1  033C          DC     N506     EA WITHIN SUBROUTINES
0106 1  031F          DC     N503     EA WITHIN AMS PROGRAM
0107 1  020C          DC     XEQ      EA WITHIN MAINLINE
0108 1  02EE          DC     N501     EA WITHIN RESIDENT MONITOR
0109 1  02EB          DC     N500     EA WITHIN FIRST FOUR WORDS
                *************************************************
010A    000C    DPARM BSS  E 0        DISK PARAMETERS
010A 0  0000          DC     0        READ OFF OF DISK INTO BUFFER
010B 1  0000          DC     IOAR1    *LOCATED AT IOAR1
010C 0  0079    D121  DC     121      CONSTANT
010D 0  0000    PRONO DC     *-*      PROBLEM NUMBER
010E 0  0000    STUNO DC     *-*      STUDENT NUMBER
                *                     *OUTSIDE RESIDENT MONITOR
010F 0  0000    STAND DC     *-*      STANDARD PROBLEM INDICATOR
0110 1  0000    ABAMS DC     BAMS     ADDRESS BEGINING AMS PROG.
0111 0  0001    EONE  DC     1        CONSTANT
0112 0  000A    D10   DC     10       CONSTANT
                *************************************************
0113 0  0000    LCORE DC     *-*      LENGTH OF CORE
0114    0000    TBS   BSS    0        TABLE OF LENGTHS OF CORE
0114 0  0000    LCOMM DC     *-*      LENGTH OF COMMON
0115 0  0000    LCLTV DC     *-*      LENGTH OF CALL TV
0116 0  0006    LFAC  DC     6        LENGTH OF FAC AND INDICATORS
0117 0  0000    LLBTV DC     *-*      LENGTH OF LIBF TV
0118 0  0000    LGAR  DC     *-*      LENGTH OF UNUSED CORE
0119 0  0000    LILS  DC     *-*      LENGTH OF ILS AREA
011A 0  0000    LSUB  DC     *-*      LENGTH OF SUBROUTINES
011B 0  0406    LAMS  DC     EAMS-BAMS LENGTH OF AMS PROGRAM
011C 0  0000    LMAIN DC     *-*      LENGTH OF MAINLINE
011D 0  0000    LCRM  DC     *-*      LENGTH OF RESIDENT MONITOR
011E 0  0004          DC     4        LENGTH OF XR SECTION
                *************************************************
011F 1  0406    AEAMS DC     EAMS     ADDRESS OF END AMS PROG.
0120 0  0000    SMALL DC     0        SMALLEST ADDRESS OF ILS
0121 0  0000    EILS  DC     *-*      END OF ILS AREA
                *************************************************
                *                                               *
                *     A M   E N T R Y   P O I N T               *
                *                                               *
                *************************************************
                *************************************************
                *  TWO PARAMETERS -- STUDENT NUMBER AND PROBLEM *
                *  NUMBER ARE LOADED FROM THE MAINLINE SO THEY  *
                *  CAN BE LATER PASSED TO THE OUTPUT PROGRAM.   *
                *************************************************
0122 0  0000    AM    DC     *-*      ENTRY POINT FOR AM
```

```
0123 01 65800122          LDX   I1 AM       LOAD XR1 WITH PL ADDRESS          AM 0835
0125 00 C5800000          LD    I1 0        LOAD FIRST PARAMETER              AM 0840
0127 0  00E5              STO      PRONO    STORE AS PROBLEM NUMBER           AM 0845
0128 01 4C080130          BSC   L  WROPN,+  GO TO WROPN IF ZERO OR NEG        AM 0850
012A 0  90E7              S        010      SUBTRACT TEN                      AM 0855
012B 01 4C300130          BSC   L  WROPN,Z- GO TO WROPN IF POSITIVE           AM 0860
012D 0  C00F              LD       PRONO    LOAD PROBLEM NUMBER               AM 0865
012E 0  90E2              S        EONE     SUBTRACT ONE                      AM 0870
012F 0  4838              BSC      +Z-      SKIP UNCONDITIONAL                AM 0875
0130 0  1010        WROPN SLA      16       CLEAR ACC TO ZERO                 AM 0880
0131 01 84000001          A     L  IOAR1+1  ADD SECTOR ADDRESS                AM 0885
0133 0  18D0              RTE      16       PLACE ACC INTO EXT                AM 0890
0134 0  C007              LD       0121     LOAD 121                          AM 0895
0135 01 D0000000          STO   L  IOAR1    STORE AS WD CT AND SECTOR         AM 0900
                    *                       *ADDRESS FOR DISK READ            AM 0905
0137 0  C802              LDD      DPARM    LOAD DISK PARAMETERS              AM 0910
0138 00 440000F2          BSI   L  DZ000    GO TO DISK ROUTINE TO READ        AM 0915
                    *                       *121 WORDS INTO IOAR+2            AM 0920
013A 01 44000395          BSI   L  READ     READ FROM BIT SW INTO ACC         AM 0925
013C 00 E5800001          AND   I1 1        LOGICAL AND SECOND PARAMETER      AM 0930
013E 0  D0D0              STO      STAND    STORE AS STANDARD INDICATOR       AM 0935
                    *                       NOTE - IF THE BIT SWITCHES        AM 0940
                    *                       *WERE ALL UP, AND THE ST.         AM 0945
                    *                       *NUMBER WAS -1, THEN THIS IS      AM 0950
                    *                       *A STANDARD PROBLEM .             AM 0955
013F 00 C5800001          LD    I1 1        LOAD SECOND PARAMETER             AM 0960
0141 0  D0CC              STO      STUNO    STORE AS STUDENT NUMBER           AM 0965
0142 0  7102              MDX    1 2        MODIFY XR1 BY 2                   AM 0970
0143 01 600001F3          STX   L1 AMSR+1   STORE XR1 AS RETURN ADDRESS       AM 0975
                    ***************************************************       AM 0980
                    *                                                 *       AM 0985
                    *     INITIALIZE LENGTH VECTOR                    *       AM 0990
                    *                                                 *       AM 0995
                    ***************************************************       AM 1000
                    *  THE VECTOR BEGINING AT TBS IS INITIALIZED FOR  *       AM 1005
                    *  THE PARTICULAR CORE LOAD.  THIS VECTOR GIVES   *       AM 1010
                    *  THE LENGTHS OF THE DIFFERENT PARTS OF THE CORE *       AM 1015
                    *  LOAD FOR USE LATER IN THE PROGRAM IN GIVING    *       AM 1020
                    *  THE EFFECT OF MEMORY PROTECT FOR CERTAIN OF    *       AM 1025
                    *  THESE SECTIONS OF CORE.  THESE SECTIONS ARE    *       AM 1030
                    *  COMMON, CALL TV, FAC, LIBF TV, UNUSED CORE,    *       AM 1035
                    *  ILS AREA, SUBROUTINES, AMS PROGRAM, MAINLINE,  *       AM 1040
                    *  RESIDENT MONITOR, AND INDEX REGISTER AREA.     *       AM 1045
                    ***************************************************       AM 1050
                    ***************************************************       AM 1055
                    *     INITIALIZE LENGTH OF LCMSK                  *       AM 1060
                    ***************************************************       AM 1065
                    *  A MASK IS PREPARED CONTAINING BITS SET IN EACH *       AM 1070
                    *  POSITION WHERE BITS CAN OCCUR IN AN ADDRESS    *       AM 1075
                    *  ON THE BASIS OF THE LENGTH OF CORE.  THEN      *       AM 1080
                    *  INDEX 3 IS TESTED.  IF IT IS NEGATIVE, THERE   *       AM 1085
                    *  IS NO LIBF TV OR FAC (FLOATING ACC).                   AM 1090
                    ***************************************************       AM 1095
0145 00 C400000E          LD    L  $CORE    LOAD LENGTH OF CORE               AM 1100
0147 0  D0CB              STO      LCORE    STORE AS LENGTH OF CORE           AM 1105
0148 0  90C8              S        EONE     SUBTRACT ONE                      AM 1110
0149 01 D400024D          STO   L  LCMSK    STORE AS LEN. CORE MASK           AM 1115
014B 00 6680007B          LDX   I2 $WRD1    LOAD LOADING ADDR. CORE LOAD      AM 1120
```

```
C140 C  6A5F              STX   2 WR01      STORE AS LOADING ACDR.             AM 1125
C14E C  C2C1              LD   X2 'CMON     LOAC LENGTH OF COMMON              AM 1130
C14F C  DCC4              STO     LCOMM     STORE AS LENGTH OF COMMON          AM 1135
0150 C  C2CA              LD   X2 'XR3X     LOAD PROPER VALUE XR3              AM 1140
0151 C1 04C003C8          STO  L  SPXR3+1 STORE AS PROPER VALUE XR3            AM 1145
C153 01 4C28016A          BSC  L  NLBTV,Z+ GO TO NO LIBF TV IF MINUS           AM 1150
                    ************************************************           AM 1155
                    *     INITIALIZE LENGTH OF LIBF TV FOR XR3 POS.    *       AM 1160
                    ************************************************           AM 1165
                    *  DETERMINE LENGTH OF LIBF TV, CALL TV, AND       *       AM 1170
                    *  FAC.  DETERMINE BEGINING OF CALL AND LIBF TV.   *       AM 1175
                    ************************************************           AM 1180
0155 0  COBD              LD      LCORE     LOAC LENGTH OF CORE                AM 1185
0156 0  9CBD              S       LCOMM     SUBTRACT LENGTH OF COMMON          AM 1190
C157 C  920B              S    X2 'TVWC     LENGTH OF TRANSFER VECTOR          AM 1195
C158 01 0400036E          STO  L  BLBTV     STORE AS BEGINING OF LIBF T V      AM 1200
C15A C  C20A              LD   X2 'XR3X     LOAD PROPER VALUE XR3              AM 1205
015B 0  8C50              A       CON       ADD CONST. FOR LIBF END            AM 1210
C15C C1 9400036E          S    L  BLBTV     SUBTRACT LOW LIBF ADDRESS          AM 1215
C15E C  DOB8              STO     LLBTV     STORE AS LENGTH LIBF TRAN.VEC      AM 1220
C15F 01 8400036E          A    L  BLBTV     ADD BEGINING OF LIBF T V           AM 1225
C161 C  8C49              A       06        ADD SIX FOR FAC AREA               AM 1230
C162 C1 0400036D          STO  L  BCALL     STORE AS BEGINING OF CALL TV       AM 1235
C164 C  COAE              LD      LCORE     LOAC LENGTH OF CORE                AM 1240
0165 01 9400036D          S    L  BCALL     SUBTRACT BEGINING OF CALL TV       AM 1245
C167 0  9CAC              S       LCOMM     SUBTRACT LENGTH OF COMMON          AM 1250
C168 0  DCAC              STO     LCLTV     STORE AS LENGTH OF CALL TV         AM 1255
C169 0  7COF              MDX     PLBTV     GO TO PLBTV                        AM 1260
                    ************************************************           AM 1265
                    *     INITIALIZE LENGTH OF LIBF TV FOR XR3 NEG.    *       AM 1270
                    ************************************************           AM 1275
                    *  DETERMINE LENGTH CF LIBF TV, CALL TV, AND       *       AM 1280
                    *  FAC.  DETERMINE BEGINING OF CALL AND LIBF TV.   *       AM 1285
                    ************************************************           AM 1290
C16A C  C208        NLBTV LD   X2 'TVWC     LOAD LENGTH OF TRANS VECTOR        AM 1295
C16B C  903F              S       06        SUBTRACT SIX                       AM 1300
C16C C  D20B              STO  X2 'TVWC     STORE AS LENGTH OF TRANS VECT      AM 1305
016D C  DCA7              STO     LCLTV     STORE AS LENGTH OF CALL TV         AM 1310
C16E C  1010              SLA     16        ENTER ACC WITH ZERO                AM 1315
C16F C  DCA7              STO     LLBTV     STORE AS LENGTH OF LIBF TV         AM 1320
0170 C  DCA5              STO     LFAC      STORE AS LENGTH OF FAC             AM 1325
0171 C  CCA1              LD      LCORE     LOAC LENGTH OF CORE                AM 1330
C172 C  9201              S    X2 'CMON     SUBTRACT LENGTH OF COMMON          AM 1335
C173 C1 94000115          S    L  LCLTV     SUBTRACT LENGTH OF CALL TV         AM 1340
C175 C1 04C0036D          STO  L  BCALL     STORE AS BEGINING OF CALL TV       AM 1345
C177 01 0400036E          STO  L  BLBTV     STORE AS BEGINING OF LIBF TV       AM 1350
                    ************************************************           AM 1355
                    *     INITIALIZE ADDRESS OF END OF CORE LOAD       *       AM 1360
                    ************************************************           AM 1365
                    *  DETERMINE END OF CORE LOAD (IE. END OF ILS      *       AM 1370
                    *  AREA) AND USE AS INITIAL BEGINING ILS AREA.     *       AM 1375
                    ************************************************           AM 1380
C179 C  CC33        PLBTV LD      WR01      LOAC LOADING ADDR CORE LOAC        AM 1385
017A 0  82C9              A    X2 'WCNT     ADD LENGTH OF CORE LOAD            AM 1390
C17B C  92C8              S    X2 'TVWC     SUBTRACT LENGTH OF TRANS VECT      AM 1395
C17C C  DCA3              STO     SMALL     STORE AS INITIAL BILS AREA         AM 1400
C17D 0  DCA3              STO     EILS      STORE AS END ILS AREA              AM 1405
C17E 01 D40003A8          STO  L  DMP+4     STORE AS PARAMETER FOR DUMP        AM 1410
```

```
                              ************************************************   AM 1415
                              *     INITIALIZE LENGTH OF RESIDENT MONITOR    *   AM 1420
                              ************************************************   AM 1425
                              *  DETERMINE LENGTH OF RESIDENT MONITOR FROM   *   AM 1430
                              *  BEGINING OF CORE TO END OF CORE IMMAGE HEADER. * AM 1435
                              ************************************************   AM 1440
0180 0  CC2C                        LD      WRD1    LOAD LOADING ADDR. CORE LOAD  AM 1445
0181 0  8204                        A    X2 'HWET   ADD LENGTH OF CORE IMM HEADER AM 1450
0182 0  D09A                        STO     LCRM    STORE AS LENGTH OF RES. MON.  AM 1455
                              ************************************************   AM 1460
                              *     INITIALIZE LENGTH OF MAINLINE            *   AM 1465
                              ************************************************   AM 1470
                              *  DETERMINE LENGTH OF MAINLINE FROM END OF CORE * AM 1475
                              *  IMAGE HEADER TO BEGINING OF AMS.            *   AM 1480
                              ************************************************   AM 1485
0183 0  C08C                        LD      ABAMS   LOAD ADDR BEGINING OF AMS     AM 1490
0184 0  9098                        S       LCRM    SUBTRACT LENGTH CORE RES.MON. AM 1495
0185 0  D096                        STO     LMAIN   STORE AS LENGTH OF MAINLINE   AM 1500
                              ************************************************   AM 1505
                              *     INITIALIZE LENGTH OF SUBROUTINES         *   AM 1510
                              ************************************************   AM 1515
                              *  DETERMINE BEGINING OF ILS AREA FROM VALUES IN * AM 1520
                              *  THE INTERRUPT TV THAT DO NOT LIE IN THE     *   AM 1525
                              *  MONITOR.   DETERMINE THE LENGTH OF SUBROUTINE * AM 1530
                              *  AREA FROM THE END OF AMS TO THE BEGINING OF  *  AM 1535
                              *  THE ILS AREA.                               *   AM 1540
                              ************************************************   AM 1545
0186 0  6105                        LDX   1 5       ENTER INDEX 1 WITH 5          AM 1550
0187 0  C107                  IN1   LD   X1 7       LOAD VALUE FROM INTER. TV     AM 1555
0188 0  9024                        S       WRD1    SUBTRACT LOADING ADDRESS      AM 1560
0189 01 4C280191                    BSC  L  XX,Z+   GO TO XX ON MINUS             AM 1565
018B 0  C107                        LD   X1 7       LOAD VALUE FROM INTER. TV     AM 1570
018C 0  9093                        S       SMALL   SUBTRACT SMALLEST FOUND       AM 1575
018D 01 4C100191                    BSC  L  XX,-    GO TO XX ON NOT MINUS         AM 1580
018F 0  C107                        LD   X1 7       LOAD VALUE FROM INTER. TV     AM 1585
0190 0  D08F                        STO     SMALL   STORE INTO SMALL              AM 1590
0191 0  71FF                  XX    MDX   1 -1      MODIFY XR1 BY -1,SKIP IF ZERO AM 1595
0192 0  70F4                        MDX     IN1     GO TO IN1                     AM 1600
0193 0  C08C                        LD      SMALL   LOAD ILS AREA ADDRESS         AM 1605
0194 0  908A                        S       AEAMS   SUBTRACT ADDRESS END AMS      AM 1610
0195 0  D084                        STO     LSUB    STORE AS LENGTH SUBROUTINES   AM 1615
                              ************************************************   AM 1620
                              *     INITIALIZE LENGTH OF ILS AREA            *   AM 1625
                              ************************************************   AM 1630
                              *  DETERMINE LENGTH OF ILS AREA FROM PREVIOUSLY *  AM 1635
                              *  DETERMINED BEGINING OF ILS AREA TO END OF   *   AM 1640
                              *  CORE LOAD.                                  *   AM 1645
                              ************************************************   AM 1650
0196 0  C08A                        LD      EILS    LOAD PROG END ADDRESS         AM 1655
0197 0  9088                        S       SMALL   SUBTRACT BILS AREA ADDRESS    AM 1660
0198 0  D080                        STO     LILS    STORE AS LENGTH ILS AREA      AM 1665
                              ************************************************   AM 1670
                              *     INITIALIZE LENGTH OF UNUSED CORE (GARBAGE) * AM 1675
                              ************************************************   AM 1680
                              *  DETERMINE LENGTH OF UNUSED CORE (GARBAGE)   *   AM 1685
                              *  FROM END OF CORE LOAD (IE END OF ILS AREA) TO * AM 1690
                              *  BEGINING OF LIBF TV.  DECREASE LENGTH OF CORE * AM 1695
                              *  RESIDENT MONITOR BY FOUR TO LEAVE SPACE FOR  *  AM 1700
```

```
                    *  INDEX REGISTER (XR) AREA.                               *
                    ***********************************************************
C199 C1 C4CC036E          LD    L  BLBTV    LOAD BEGINING OF LIBF TV
C19B C  9C85              S        EILS     SUBTRACT PROG END ADDRESS
C19C C1 D4CC0118          STO   L  LGAR     STORE AS LENGTH OF GARBAGE
C19E C1 74FCC11D          MDX   L  LCRM,-4  SUBTRACT 4 FRCM LEN. RES MCN
                    ***********************************************************
                    *     PRINT OPERATOR MESSAGE AND RETURN                    *
                    ***********************************************************
                    *  PRINT OUT THE OPERATOR MESSAGE ON THE CONSOLE           *
                    *  PRINTER 'SET MODE SW TO INT RUN'.  MESSAGE IS           *
                    *  NOT PRINTED IF INTERRUPT RUN MODE IS ALREADY            *
                    *  ON.  IF OPERATOR DOES NOT CHANGE MODE TO INT            *
                    *  RUN BEFORE PRESSING PROG START, MESSAGE IS              *
                    *  PRINTED AGAIN.  PRESSING PROGRAM STOP DOES NOT          *
                    *  CAUSE THIS FUNCTION TO BE ALTERED.  WHEN MODE           *
                    *  SWITCH IS PROPERLY SET, PROGRAM BEGINS TRACING          *
                    *  THROUGH THE MAINLINE, WITH A INTERRUPT OCCUR-           *
                    *  ING ON LEVEL 5 BEFORE EACH INSTRUCTION.                 *
                    *  THE LEVEL 5 INTERRUPT ENTRY POINT IS ENT5.              *
                    ***********************************************************
C1A0 CC C40CCCC0          LD    L  13       LOAD FROM LOC 13 (L 5 INT TV)
C1A2 C  DC54              STO      SAVL5    SAVE IN SAVL5
C1A3 C  CC2C              LD       ANL5     LOAD ADDRESS FOR NEW LEVEL 5
C1A4 CC D4C00000          STO   L  13       STORE AS LEVEL 5 INTER. ADDR.
C1A6 C  C218              LD    X2 'ITCK    LOAD 1130 CONSOLE/KEYBOARD
                    *                       *ISS TV ENTRY
C1A7 C  DC24              STO      SAVKC    SAVE IN SAVKC
C1A8 0  CC22              LD       ANKC     LOAD ADDRESS OF NEW CON/KEYBD
                    *                       *ISS ROUTINE
C1A9 C  D218              STO   X2 'ITCK    STORE AS 1130 CON/KEYBD ISS
                    *                       *TV ENTRY
C1AA 0  7C13              MDX      INITX    GO TO INITX
                    ***********************************************************
C1AB C  0CC6        D6    DC       6        CONSTANT
C1AC C  CC7A        CON   DC       /80-6    CONSTANT
C1AD C  0C00        WRD1  DC       *-*      LOADING ADDRESS CORE LOAD
                    *                       *(BEGINING CORE IMAGE HEADER)
                    ***********************************************************
C1AE C  1C00        PRINT NOP               NO-OP INSTRUCTION
C1AF C  0818              XIO      IOCCP    PRINT ONE LETTER
C1B0 C1 74C101C8          MDX   L  IOCCP,1  INCRIMENT ADDRESS IN IOCCP
C1B2 C  3CCC              WAIT              WAIT FOR INTERRUPT
C1B3 C  0CCC        NKC   DC       *-*      ENTRY PT FOR ISS CON/KEYBD
                    *                       *(ALSO A WAIT INSTRUCTION)
C1B4 C  0811              XIO      IOCC4    SENSE DSW AND RESET ILSW BIT
C1B5 C  ECCC              AND      HOC00    REMOVE ALL BUT BUSY IND.
C1B6 01 4C200183          BSC   L  NKC,Z    GO TO NKC IF NOT ZERO
C1B8 C1 74FF01C4          MDX   L  PCNT,-1  MODIFY PCNT BY -1,SKIP IF C
C1BA C  7CC5              MDX      BOSCP    GO TO BOSCP IF NO SKIP
C1BB C  3CCC              WAIT              WAIT FOR OPERATOR
C1BC C  CCC0              LD       AMBUF    LOAD ADDRESS OF MBUF
C1BD C  DCCA              STO      IOCCP    STORE INTO IOCCP TO RESTORE
C1BE C  CCC4        INITX LD       D31      LOAD 31
C1BF C  DCC4              STO      PCNT     STORE AS PRINT COUNT
C1C0 C1 4C4C01AE    BOSCP BOSC L   PRINT    GO TO PRINT AND OFF INTERRUPT
                    *                       *UNLESS LEVEL 5 ALSO ON,
                    *                       *IN WHICH CASE GO TO NL5.
```

```
                              ************************************************
01C2 0   0C00          HOCOO DC      /0C00    CONSTANT
01C3 0   001F          D31 . OC      31       CONSTANT
01C4 0   0C00          PCNT  OC      *-*      PRINT CCUNT (NC OF CHAR.)
01C6     0000          IOCC4 BSS  E  0        IOCC TO SENSE CSW AND
C1C6 0   0000                OC      0        *RESET CSW ANC ILSW
01C7 0   0FC1                OC      /0F01    *FOR CONSOLE PRINTER.
01C8     0000          IOCCP BSS  E  C        IOCC TC PRINT CN CONSOLE
01C8 1   0005                DC      MBUF     *PRINTER ONE CHARACTER AT
01C9 0   090C                OC      /0900    *LOCATICN MBUF
01CA 1   0005          AMBUF OC      MBUF     ADORESS OF MBUF
01CB 1   01B3          ANKC  OC      NKC      ACORESS CF NKC
01CC 0   0C00          SAVKC OC      *-*      LOCATION TC SAVE K/C ISS ENT
01CE     0C02          IOAR2 BSS  E  2        TOP OF CISK BUFFER 2
01D0 1   0102          ANL5  OC      NL5      ADORESS OF NEW LEVEL 5
01D1 1   024E          AENT5 OC      ENT5     ACCRESS CF ENT5 ENTRY PT
                              ************************************************
C1D2 0   0C0C          NL5   OC      *-*      NEW ENTRY PCINT FOR LEVEL 5
01D3 0   0862                XIO     IOCC     SENSE OEVICE STATUS WORC L 5
01D4 C   1C01                SLA     1        SHIFT INT RUN BIT INTO ACC C
01D5 01  4C1001C0            BSC  L  BOSCP,-  GO TO BCSCP IF NOT INT RUN
C1D7 C   CCF9                LD      AENT5    LOAC ENTRY ACCRESS FOR L 5
01D8 00  04000C00            STO  L  13       STORE INTC LCC 13
01DA C   CCF1                LO      SAVKC    LOAC SAVEC CON/KEYBC ISS ENT
01DB 0   0218                STO  X2 'ITCK    RESTCRE CON/KEYBO ISS TV ENT
                              ************************************************
01DC 01  4400037F            BSI  L  IONC     WAIT FOR ALL I/C CFF
C1DE C1  C4000368            LD   L  DM1      LOAC MINUS CNE
01E0 C1  94000114            S    L  LCCMM    SUBTRACT LENGTH OF COMMCN
01E2 0   0CC6                STO     BCOMM+1  STORE AS BEGINING OF COMMCN
01E3 C1  65800114            LOX  I1 LCOMM    LOAC XR1 WITH LEN. CF CCMMON
C1E5 C   6287                LOX   2 -121     ENTER XR2 WITH -121
01E6 01  C600007B      GAGN  LD   L2 GTBL+12C LCAD VALUE FRCM BUFFER
01E8 00  D5000000      BCOMM STO  L1 *-*      STORE IN COMMCN
01EA 0   7201                MDX   2 +1       MODIFY XR2 BY 1, SKIP IF ZERO
C1EB 0   7C02                MOX     AROUN    GO TC AROUN (IF NC SKIP)
01EC C1  74C201EF            MOX  L  INSCH,+2 MOCIFY BRANCH ADCRESS BY +2
01EE 0   71FF          AROUN MOX   1 -1       MOCIFY XR1 BY -1,SKIP IF ZERO
01EF 0   70F6          INSCH MOX     GAGN     GO TC GAGN (IF NO SKIP)
                              ************************************************
01F0 C   4878                BCSC    +-Z      SKIP ANC CFF INTERRUPT
C1F1 C   1C0C                NOP              NO-CP
01F2 00  4C000C00      AMSR  BSC  L  *-*      EXIT FRCM AMS AND RETURN
                              ************************************************
01F4 C   0C1C          HOO10 DC      /0010    CCNSTANT
C1F5 C   8C0C          H800C DC      /8000    CONSTANT
01F6 0   3C0C          H300C CC      /3000    CCNSTANT (ECUALS WAIT INSTR)
C1F7 C   000C          SAVL5 CC      *-*      LOCATION TC SAVE L 5 TV
C1F8 C   000C          WAITC OC      0        WAIT IF NEGATIVE
C1F9 C   000C          WAITO OC      0        WAIT INCICATCR FOR CON ENT SW
                       *                      WAIT IF CON ENT SW C IS UP
01FA     0C0C                BSS  E  0        EVEN CCRE BCUNCARY
C1FA 1   01F9          ICCCD OC      WAITC    REAC INTC WAITC
01FB C   3A0C                OC      /3A00    THE CCNSCLE ENTRY SWITCHES
C1FC C   1C0C          NOP   NCP              A NC-CP INSTRUCTICN
C1FD C   0C0C          LAOOR DC      0        ACCRESS CF LAST INSTRUCTICN
C1FE     0CC2          LINST BSS  E  2        LAST INSTRUCTICN
                              ************************************************
```

```
                                  *                                                 *
                                  *     X E Q   GO BACK AND EXECUTE NEXT INSTR.     *
                                  *                                                 *
                                  ***************************************************
                                  *  THIS ROUTINE IS ENTERED WHEN IT IS DECIDED TO  *
                                  *  GO ON AND EXECUTE THE NEXT INSTRUCTION.        *
                                  ***************************************************
                                  ***************************************************
                                  *     INCREMENT INSTRUCTION COUNTER AND TEST      *
                                  ***************************************************
0200 0  621B                      XEQ   LDX   2 /1B      ENTER INDEX 2 WITH 1B HEX
0201 01 44000395                        BSI   L  READ    READ CON. ENTRY SW INTO ACC
0203 0  E0FC                            AND      H0010   REMOVE ALL BUT BIT ELEVEN
0204 01 4C2003AC                        BSC   L  CGA,Z   GO TO CGA IF NOT ZERO
0206 0  C831                            LDD      INSCT   LOAD INSTRUCTION COUNTER
0207 0  9834                            SD       RTIME   SUBTRACT DOUBLE FROM RTIME
0208 01 4C1803AC                        BSC   L  CGA,-+  GO TO COMPUTE GRADE IF ZERO
020A 0  C82D                            LDD      INSCT   LOAD INSTRUCTION COUNTER
020B 0  882E                            AD       DONE    ADD DOUBLE  ONE
020C 0  D82B                            STD      INSCT   STORE DOUBLE INTO INST. CTR
020D 0  18C6                            RTE      6       MOVE LOW 6 OF EXT TO HIGH ACC
020E 0  18CA                            SRA      16-6    SHIFT THESE BITS INTO LOW ACC
020F 00 D4000001                        STO   L  1       STORE ACC INTO INDEX 1
0211 01 C40002C1                        LD    L  ADDR    LOAD ACCRESS OF INSTR
0213 01 D5000043                        STO   L1 ATB     STORE INTO ADDRESS TABLE
0215 0  C0E6                            LD       NOP     LOAD A NO-OP INSTRUCTION
0216 0  D011                            STO      WAIT    STORE INTO LOCATION WAIT
0217 0  08E2                            XIO      IOCCD   SENSE CONSOLE SWITCHES
0218 0  081D                            XIO      IOCC    SENSE DEVICE STATUS WORD L 5
0219 0  E8DF                            OR       WAITD   LOGICAL OR IN CON ENT SW.
021A 0  E8DD                            OR       WAITC   LOGICAL OR IN WAITC INDICATOR
                                  *                      WAITC IS NEG. IF INSTR=WAIT
021B 01 4C100227                        BSC   L  NWAIT,- GO TO NWAIT IF NOT MINUS
021D 01 C40002C1                        LD    L  ADDR    LOAD ACCRESS OF INSTR
021F 0  1004                            SLA      4       REMOVE 4 HIGH BITS,SET CARRY
0220 0  1804                            SRA      4       RIGHT JUSTIFY ACC
0221 0  4802                            BSC      C       SKIP IF CARRY OFF
0222 0  E8D2                            OR       H8000   OR IN HIGH ORDER BIT
0223 0  E8D2                            OR       H3000   MAKE INTO WAIT INSTRUCTION
0224 0  D0C3                            STO      WAIT    STORE AS WAIT INSTRUCTION
0225 01 4400037F                        BSI   L  IONC    WAIT FOR ALL I/O OFF
0227 0  4003                      NWAIT BSI      RESTO   RESTORE ACC,EXT,XR1,XR2,STATS
0228 0  0000                      WAIT  DC       *-*     EITHER A NOP INSTR, OR A
                                  *                      *WAIT FOR PROGRAM START KEY
                                  *                      *AND DISPLAY IAR IN STORAGE
                                  *                      *BUFFER REGISTER
0229 01 4C00024E                        BOSC  I  ENT5    RETURN AND OFF INTERRUPT
                                  ***************************************************
                                  *     RESTO - RESTORES ACC,EXT,XR1,XR2,C,O        *
                                  ***************************************************
                                  *  THIS ROUTINE IS ENTERED WHEN IT IS DESIRED TO  *
                                  *  RESTORE THE REGISTERS TO THEIR VALUES WHEN THE *
                                  *  LAST INSRTUCTION WAS EXECUTED. THE ACCUMUL-    *
                                  *  ATOR, EXTENTION, CARRY, OVERFLOW, INDEX1, AND  *
                                  *  INDEX 2 ARE RESTORED.  (INDEX 3 DOES NOT NEED  *
                                  *  RESTORING AS IT IS NOT ALTERED.)               *
                                  ***************************************************
022B 0  0000                      RESTO DC       *-*     ENTRY POINT FOR RESTO
```

```
022C 00 65000000    SAVX1 LDX   L1 *-*      RESTORE INDEX 1                    AM 2575
022E 00 66000000    SAVX2 LDX   L2 *-*      RESTORE INDEX 2                    AM 2580
0230 0  C803              LDD      SAVE1    RESTORE ACC AND EXT                AM 2585
0231 0  2C00        SAVCO LDS      *-*      RESTORE CARRY AND OVERFLOW         AM 2590
0232 01 4C80022B          BSC   I  RESTC    RETURN TO CALLING POINT            AM 2595
                    ************************************************          AM 2600
                    *     CONSTANTS FOR USE BY UPPER HALF              *      AM 2605
                    ************************************************          AM 2610
0234    0002        SAVE1 BSS   E  2        LOCATION TO SAVE ACC EXT           AM 2615
0236 0  0000        IOCC  DC       0        IOCC TO SENSE DEVICE               AM 2620
0237 0  3F01              DC       /3F01    STATUS WORD FOR STOP/INT RUN       AM 2625
0238 0  0000        INSCT DC       0        INSTRUCTION COUNTER                AM 2630
0239 0  0000              DC       0        SECOND HALF INSTR. COUNTER         AM 2635
023A 0  0000        DONE  DC       0        FIRST WORD OF DOUBLE PRE. 1        AM 2640
023B 0  0001        ONE   DC       1        CONSTANT                           AM 2645
023C 0  0001        RTIME DC       1        GIVE ONE MINUTE OF                 AM 2650
023D 0  0000              DC       0        *RUNTIME                           AM 2655
023E 0  0000        DISP  DC       0        DISPLACEMENT                       AM 2660
023F    0006        TEA   BSS      6        TABLE OF EA IS SIX WORDS LONG      AM 2665
0245 0  0008        D8    DC       8        CONSTANT                           AM 2670
0246 0  000D        D13   DC       13       CONSTANT                           AM 2675
0247 0  0038        AEXIT DC       $EXIT    CONSTANT                           AM 2680
0248 0  00F0        HF0   DC       /F0      CONSTANT                           AM 2685
0249 0  0080        H0080 DC       /0080    CONSTANT                           AM 2690
024A 0  0300        H0300 DC       /0300    CONSTANT                           AM 2695
024B 0  0400        H0400 DC       /0400    CONSTANT                           AM 2700
024C 0  03FF        H03FF DC       /03FF    CONSTANT                           AM 2705
024D 0  0000        LCMSK DC       *-*      LENGTH OF CORE MASK                AM 2710
                    ************************************************          AM 2715
                    *                                              *          AM 2720
                    *     LEVEL 5 INTERRUPT ENTRY POINT            *          AM 2725
                    *                                              *          AM 2730
                    ************************************************          AM 2735
                    *  THIS POINT IS ENTERED AFTER THE EXECUTION OF    *      AM 2740
                    *  EACH MACHINE LANGUAGE INSTRUCTION IN THE USER   *      AM 2745
                    *  WRITTEN PROGRAM AND USER CALLED SUBPROGRAMS.    *      AM 2750
                    *  INTERRUPT RUN MODE, OPERATING ON LEVEL 5, IS    *      AM 2755
                    *  USED TO IMPLIMENT THIS FUNCTION.  (SEE IBM      *      AM 2760
                    *  1130 FUNCTIONAL CHARACTERISTICS MANUAL FOR      *      AM 2765
                    *  FURTHER INFORMATION.)                           *      AM 2770
                    ************************************************          AM 2775
                    *  THE ACCUMULATOR, EXTENTION, CARRY, OVERFLOW,    *      AM 2780
                    *  INDEX 1, AND INDEX 2 ARE SAVED SO THAT THEY     *      AM 2785
                    *  MAY BE RESTORED BEFORE EXECUTION OF THE USER'S  *      AM 2790
                    *  NEXT INSRTUCTION BY THE ROUTINE RESTC.  THE     *      AM 2795
                    *  NEXT INSTRUCTION (INST) AND ITS ADDRESS (ADDR)  *      AM 2800
                    *  ARE LOADED, WITH THE OLD VALUES BEING STORED    *      AM 2805
                    *  INTO LAST INSTRUCTION (LINST) AND ITS ADDRESS   *      AM 2810
                    *  (LADDR).                                        *      AM 2815
                    ************************************************          AM 2820
024E 0  0000        ENT5  DC       *-*      LEVEL 5 ENTRY POINT                AM 2825
024F 0  69DD              STX   1  SAVX1+1  SAVE INDEX 1                       AM 2830
0250 0  6ADE              STX   2  SAVX2+1  SAVE INDEX 2                       AM 2835
0251 0  D8E2              STD      SAVE1    SAVE ACC AND EXT                   AM 2840
0252 0  28DE              STS      SAVCO    SAVE CARRY AND OVERFLOW            AM 2845
0253 0  C06D              LD       ADDR     LOAD LAST ADDRESS                  AM 2850
0254 0  DCA8              STO      LADDR    STORE AS LAST ADDRESS              AM 2855
0255 0  C86C              LDD      INST     LOAD DOUBLE LAST INSTRUCTION       AM 2860
```

```
C256 C  D8A7              STD     LINST    STORE DOUBLE AS LAST INSTRUCT
C257 C  CCF6              LD      ENT5     LOAD ADDRESS OF INSTRUCTION
C258 C  ECF4              AND     LCMSK    'DIVIDE' BY LENGTH OF CORE
C259 C  DC67              STO     ADDR     STORE AS ADDRESS
C25A CC D4000002          STO  L  2        STORE INTO INDEX 2
C25C C  C2CC              LD    2 C        LOAD INSTRUCTION
C25D C  DC64              STO     INST     STORE AS INSTRUCTION
C25E C  C2C1              LD    2 1        LOAD SECOND WORD OF INSTR
C25F C  DC63              STD     INST+1   STORE AS INSTRUCTION (LOWER)
          ***********************************************************
          *                                                         *
          *     TEST ADDRESS USING TABLE                            *
          *                                                         *
          ***********************************************************
          *  THE INDICATOR TELLING IF THE INSTRUCTION IS A          *
          *  WAIT IS INITIALIZED TO ZERO (WAITC).  A LOOP           *
          *  IS THEN PREFORMED TO DETERMINE IN WHAT PARTIT-         *
          *  ION OF CORE THE ADDRESS OF THE INSTRUCTION IS          *
          *  LOCATED.  THE INDEX OF THIS LOOP IS USED TO            *
          *  BRANCH TO THE PROPER POINT FOR TESTING OF THE          *
          *  ADDRESS.                                               *
          ***********************************************************
0260 C  1C1C              SLA     16       ENTER ACC WITH ZERO
C261 C  DC96              STO     WAITC    INITIALIZE WAITC TO ZERO
C262 C  C05E              LD      ADDR     LOAD ADDRESS
C263 C  620A              LDX   2 10       ENTER INDEX 2 WITH 10
C264 C1 96000114     BACK S    L2 TBS      SUBTRACT ENTRY IN TABLE
C266 C1 4C28026A          BSC  L  OUT,Z+   GO TO OUT ON MINUS
C268 0  72FF              MDX   2 -1       MODIFY XR2 BY -1,SKIP IF ZERO
C269 C  70FA              MDX     BACK     GO TO BACK
C26A C  6A54         OUT  STX   2 SADRS    SAVE XR2 IN SADRS
C26B C1 4E8C0CF4          BSC  I2 TADDR    GO TO THROUGH TABLE OF ADDR
          ***********************************************************
          *     ADDRESS WITHIN RESIDENT MONITOR                     *
          ***********************************************************
          *  IF THE ADDRESS IS WITHIN THE RESIDENT MONITOR,         *
          *  THIS ROUTINE IS ENTERED.  IF THE ADDRESS OF            *
          *  THE INSTRUCTION (ADDR) IS THE CALL EXIT ENTRY          *
          *  TO THE MONITOR, INDEX 2 SET TO INDICATE A              *
          *  NORMAL EXIT.  IN EITHER CASE THE PROGRAM IS            *
          *  NOT ALLOWED TO CONTINUE EXECUTING.  THIS IS            *
          *  ACCOMPLISHED BY GOING TO CGA.                          *
          ***********************************************************
026D C  C053         N402 LD      ADDR     LOAD ACC WITH ADDRESS
026E C  F0D8              EOR     AEXIT    COMPARE WITH EXIT ENT POINT
C26F C1 4C2003AC          BSC  L  CGA,Z    GO TO CGA IF NOT ZERO
C271 C  6220              LDX   2 /20      ENTER XR2 WITH /20 AS
                                           *INDICATOR OF NORMAL EXIT
0272 0  704A              MDX     CG       GO TO COMPUTE GRADE
          ***********************************************************
          *     ADDRESS WITHIN MAINLINE                             *
          ***********************************************************
          *  IF THE ADDRESS IS WITHIN THE MAINLINE PROGRAM,         *
          *  THE MON INDICATOR IS SET TO ZERO TO INDICATE           *
          *  THAT THE PROGRAM IS WITHIN THE MAINLINE.               *
          ***********************************************************
0273 C  1010         N410 SLA     16       ENTER ACC WITH ZERO
0274 C  DC4F              STO     MON      STORE ZERO INTO MON INDICATOR
```

```
                         *                    MON = 1 WHILE IN MONITOR
                         *                    MON = 0 WHILE IN MAINLINE
                         *                    MON =-1 WHILE IN SUBROUTINES
0275 0  7003                   MDX     N106   GO TO N106
                         ***********************************************
                         *     ADDRESS WITHIN SUBROUTINES OR LIBF TV     *
                         ***********************************************
                         *  IF THE PROGRAM IS WITHIN THE SUBROUTINE AREA    *
                         *  OR THE LIBF TRANSFER VECTOR, THE MON INDICATOR *
                         *  IS TESTED TO DETERMINE IF THE PROGRAM IS        *
                         *  VALIDLY WITHIN THESE AREAS (IE. IT MUST BE      *
                         *  EQUAL TO MINUS ONE).                            *
                         ***********************************************
0276 0  C04D             N414  LD      MON    LOAD ACC WITH MON INDICATOR
0277 01 4C1003AC               BSC  L  CGA,-  GO TO CGA ON NOT MINUS
                         ***********************************************
                         *                                                  *
                         *     FORMS EFFECTIVE ADDRESS                      *
                         *                                                  *
                         ***********************************************
                         *  AFTER IT HAS BEEN DETERMINED THAT THE           *
                         *  INSTRUCTION IS IN A VALID PARTITION OF CORE,    *
                         *  THE TESTING CONTINUES WITH THIS ROUTINE TO      *
                         *  DETERMINE IF THE EFFECTIVE ADDRESS IS IN A      *
                         *  VALID PARTITION OF CORE.  CONSIDERATION IS      *
                         *  MADE OF THE TYPE INSTRUCTION INVOLVED, THE      *
                         *  PARTITION IN WHICH IT IS LOCATED, AND WHETHER   *
                         *  OR NOT IT WILL ALTER CORE IF EXECUTED.  (IF     *
                         *  IT WILL ALTER CORE, THE STORE INDICATOR IS SET *
                         *  TO ONE.  IF NOT, IT IS SET TO ZERO.)            *
                         ***********************************************
                         *  EFFECTIVE ADDRESSES ARE CALCULATED FOR EACH     *
                         *  OF THE SIX POSSIBLE TYPES OF ADDRESSING --      *
                         *  SHORT, SHORT INDEXED, LONG, LONG INDEXED,       *
                         *  INDIRECT, AND INDIRECT INDEXED.                 *
                         ***********************************************
0279 01 6680022F         N106  LDX  I2 SAVX2+1 RESTORE INDEX 2
027B 0  C046                   LD      INST   LOAD INSTRUCTION FOR TESTING
027C 0  E0CF                   AND     H03FF  REMOVE ALL BUT TAG AND DISP.
027D 0  1888                   SRT     8      MOVE -- TAG-LOW ACC, DISP-EXT
027E 0  D005                   STO     INS1+1 STORE AS ADDRESS OF INSTR.
027F 0  D00D                   STO     INS2+1 STORE AS ADDRESS OF INSTR.
0280 0  18D0                   RTE     16     MOVE EXT (DISP) TO ACC
0281 0  1888                   SRT     8      EXTEND SIGN OF DISPLACEMENT
0282 0  D0BB                   STO     DISP   STORE AS THE DISPLACEMENT
0283 00 84000000         INS1  A    L  *-*    ADD THE INDEX REGISTER
0285 0  D0BA                   STO     TEA+1  STORE IN TABLE EFFECTIVE ADDR
0286 0  C0B7                   LD      DISP   LOAD THE DISPLACEMENT
0287 0  8039                   A       ADDR   ADD THE ADDRESS OF INST
0288 0  803E                   A       D1     ADD ONE BECAUSE IAR=1+ADDR
0289 0  D0B5                   STO     TEA+0  STORE IN TABLE EFFECTIVE ADDR
028A 0  C038                   LD      INST+1 LOAD SECOND WORD OF INST
028B 0  D0B5                   STO     TEA+2  STORE IN TABLE EFFECTIVE ADDR
028C 00 84000000         INS2  A    L  *-*    ADD THE INDEX REGISTER
028E 0  D0B3                   STO     TEA+3  STORE IN TABLE EFFECTIVE ADDR
028F 01 C4800241               LD   I  TEA+2  LOAD INDIRECT FROM AN EA
0291 0  D0B1                   STO     TEA+4  STORE IN TABLE EFFECTIVE ADDR
0292 01 C4800242               LD   I  TEA+3  LOAD INDIRECT FROM AN EA
```

```
0294 0  D0AF           STO      TEA+5   STORE IN TABLE EFFECTIVE ADDR   AM 3445
                 ************************************************  AM 3450
                 *     TEST INSTRUCTION                         *  AM 3455
                 ************************************************  AM 3460
                 *  THE INSTRUCTION IS TESTED TO DETERMINE WHICH   *  AM 3465
                 *  OF THE ENTRIES IN THE EFFECTIVE ADDRESS TABLE  *  AM 3470
                 *  IS IN FACT THE EFFECTIVE ADDRESS.  A BRANCH    *  AM 3475
                 *  IS THEN MADE TO THE ROUTINE WHICH TESTS THE    *  AM 3480
                 *  EFFECTIVE ADDRESS BY DETERMINING WHICH         *  AM 3485
                 *  PARTITION OF CORE IT IS IN.                    *  AM 3490
                 ************************************************  AM 3495
0295 0  6100           LDX   1 0        ENTER INDEX 1 WITH ZERO         AM 3500
0296 0  C02B           LD       INST    LOAD INSTRUCTION FOR TESTING    AM 3505
0297 0  E0B2           AND      H0300   REMOVE ALL BUT TAG BITS         AM 3510
0298 0  4820           BSC      Z       SKIP ON ZERO                    AM 3515
0299 0  611C           LDX   1 32-4     ENTER INDEX 1 WITH 28           AM 3520
029A 0  C027           LD       INST    LOAD INSTRUCTION FOR TESTING    AM 3525
029B 0  E0AF           AND      H0400   REMOVE ALL BUT FORMAT BIT       AM 3530
029C 01 4C1802A3       BSC   L  SHORT,+- GO TO SHORT ON ZERO            AM 3535
029E 0  7110           MDX   1 16       MODIFY INDEX 1 BY 16            AM 3540
029F 0  C022           LD       INST    LOAD INSTRUCTION FOR TESTING    AM 3545
02A0 0  E0A8           AND      H0080   REMOVE ALL BUT INDIRECT BIT     AM 3550
02A1 0  4820           BSC      Z       SKIP IF ACC ZERO                AM 3555
02A2 0  71F8           MDX   1 -8       MODIFY INDEX 1 BY -8            AM 3560
02A3 0  C01E     SHORT LD       INST    LOAD INSTRUCTION FOR TESTING    AM 3565
02A4 0  180B           SRA      11      REMOVE ALL BUT OP CODE          AM 3570
02A5 0  1001           SLA      1       MULTIPLY BY TWO                 AM 3575
02A6 00 D4000002       STO   L  2       STORE ACC INTO INDEX 2          AM 3580
02A8 01 CE000084       LDD   L2 TINST   LOAD DOUBLE FROM TABLE INST     AM 3585
02AA 0  19C0           RTE   1 0        ROTATE RIGHT ACC USING XR1      AM 3590
02AB 0  180C           SRA      12      REMOVE ALL BUT HEX DIGIT        AM 3595
02AC 0  901B           S        D4      SUBTRACT FOUR                   AM 3600
02AD 01 4C2802C9       BSC   L  SP,+Z   GO TO SP ON MINUS               AM 3605
02AF 0  18D1           RTE      17      PLACE STORE BIT INTO HIGH ACC   AM 3610
02B0 0  180F           SRA      15      MOVE INTO LOW ACC               AM 3615
02B1 0  D013           STO      STORE   STORE AS STORE INDICATOR        AM 3620
                 *                      STORE INDICATOR = 0 IF LOAD     AM 3625
                 *                      STORE INDICATOR = 1 IF STORE    AM 3630
02B2 0  18C3           RTE      3       MOVE 3 BITS OF EXT INTO ACC     AM 3635
02B3 0  180D           SRA      16-3    RIGHT JUSTIFY THE THREE BITS    AM 3640
02B4 00 D4000001       STO   L  1       STORE ACC INTO INDEX 1          AM 3645
02B6 01 C500023F       LD    L1 TEA     LOAD FROM TABLE OF EA           AM 3650
02B8 0  E094     TEST  AND      LCMSK   'DIVIDE' BY LENGTH OF CORE      AM 3655
02B9 0  D00C           STO      EA      STORE AS EFFECTIVE ADDRESS      AM 3660
02BA 0  4024           BSI      TSTEA   RETURN BRANCH TO TEST EA        AM 3665
02BB 01 4C000200 XEQN  BSC   L  XEQ     GO TO XEQ ACTUAL                AM 3670
02BD 01 4C0003AC CG    BSC   L  CGA     GO TO CG ACTUAL                 AM 3675
                 ************************************************  AM 3680
                 *     CONSTANTS FOR GENERAL USE                *  AM 3685
                 ************************************************  AM 3690
02BF 0  0000     SADRS DC       *-*     LOCATION TO SAVE ADDR INDIC.    AM 3695
02C0 0  0000     SEAS  DC       *-*     LOCATION TO SAVE EA INDICATOR   AM 3700
02C1 0  0000     ADDR  DC       0       ADDRESS OF INSTRUCTION NEXT     AM 3705
02C2    0002     INST  BSS   E  2       NEXT INSTRUCTION                AM 3710
02C4 0  0000     MON   DC       0       MON INDICATOR                   AM 3715
                 *                      MON = 1 WHILE IN MONITOR        AM 3720
                 *                      MON = 0 WHILE IN MAINLINE       AM 3725
                 *                      MON =-1 WHILE IN SUBROUTINES    AM 3730
```

```
02C5 0  0000        STORE DC        0        STORE INDICATCR
                    *                        STORE INDICATCR = 0 IF LCAD
                    *                        STORE INDICATCR = 1 IF STORE
02C6 0  0000        EA    DC        0        EFFECTIVE ACCRESS STORAGE
02C7 0  0001        D1    DC        1        CONSTANT
02C8 0  0004        D4    DC        4        CONSTANT
                    ***************************************************
                    *      TEST FOR SPECIAL CASES                     *
                    ***************************************************
                    *  THIS ROUTINE IS ENTEREC WHEN IT IS DESIRED TO  *
                    *  TEST THE SPECIAL CASES WHERE THE INSTRUCTION   *
                    *  HAS NO EFFECTIVE ACCRESS.  THESE CASES ARE     *
                    *  CLASSIFIED FOUR WAYS -- INSTRUCTIONS THAT CAN  *
                    *  NOT BE ALLOWED TO EXECUTE, INSTRUCTIONS THAT   *
                    *  ARE ALWAYS ALLOWED TO EXECUTE, WAIT INSTR-     *
                    *  UCTIONS, AND THE MDX INSTRUCTION.              *
                    ***************************************************
02C9 0  621C        SP    LDX    2 /1C       ENTER XR2 WITH /1C AS INDIC.
02CA 00 D4000001          STO  L   1         STORE ACC INTC XR1
02CC 01 4D8002D2          BSC  I1 TSPR       BRANCH THROUGH TSPR USING XR1
02CE 1  0200              DC       XEQ       GO TC XEQ
02CF 1  03AC              DC       CGA       GO TO CGA
02D0 1  02D2              DC       WAITS     GO TC WAITS
02D1 1  02D6              DC       MDXLO     GO TO MDXLO
02D2 01 74FF01F8    WAITS MDX  L   WAITC,-1 DECREMENT WAITC TO NEGATIVE
02D2                TSPR  EQU      WAITS
02D4 0  1000              NOP
02D5 0  70E5              MDX      XEQN      GO TC XEC
02D6 0  6100        MDXLO LDX    1 0         ENTER INDEX 1 WITH ZERO
02D7 01 4400036F          BSI  L   BITS      RETURN BRANCH TO BITS
02D9 0  FFFF              DC       /FFFF       TEST FOR MDX INSTR. WITH
02DA 0  7400              DC       /7400       LONG FORMAT, NO INDEX, AND
                    *                          *ZERO DISPLACEMENT
02DB 0  6101              LDX    1 1         ENTER XR1 WITH 1 IF FALSE
02DC 0  69E8              STX    1 STORE     STORE INDEX 1 AS STORE INC.
02DD 0  C0E5              LD       INST+1    LOAC 2NC HALF CF INSTRUCTION
02DE 0  7009              MDX      TEST      GO TO TEST
                    ***************************************************
                    *                                                 *
                    *      TEST EFFECTIVE ACCRESS USING TABLE         *
                    *                                                 *
                    ***************************************************
                    *  THE EFFECTIVE ADDRESS IS TESTEC BY DETERMINING *
                    *  IN WHICH PARTITION OF CORE IT LIES.            *
                    ***************************************************
02DF 0  0000        TSTEA DC       *-*       ENTRY PCINT FCR TEST EA
02E0 0  620A              LDX    2 10        ENTER INDEX 2 WITH 10
02E1 01 96000114    BACK1 S    L2 TBS        SUBTRACT ENTRY IN TABLE
02E3 01 4C2802E7          BSC  L  OUT1,Z+    GO TO OUT1 ON MINUS
02E5 0  72FF              MDX    2 -1        MODIFY XR2 BY -1,SKIP IF ZERC
02E6 0  70FA              MDX      BACK1     GO TO BACK1
02E7 0  7208        OUT1  MDX    2 11        MCDIFY XR2 BY 11 IN ORCER TC
                    *                        *USE THE LCWER HALF CF TACCR
02E8 0  6AD7              STX    2 SEAS      SAVE XR2 IN SEAS
02E9 01 4E8000F4          BSC  I2 TACCR      GO TC THROUGH TABLE OF ACCR
                    ***************************************************
                    *      EFFECTIVE ACCRESS IN FIRST FCUR WCRCS      *
                    ***************************************************
```

```
                        *  IF THE EFFECTIVE ADDRESS IS EQUAL TO ZERO,      *   AM 4025
                        *  IT IS TESTED AS PART OF THE MONITOR.  IF THE    *   AM 4030
                        *  EFFECTIVE ADDRESS IS WITHIN THE INDEX           *   AM 4035
                        *  REGISTERS (LOCATIONS ONE, TWO, AND THREE IN     *   AM 4040
                        *  CORE), A BRANCH IS MADE TO XEQ IN ORDER TO      *   AM 4045
                        *  RETURN TO THE CALLING PROGRAM.                  *   AM 4050
                        ***************************************************   AM 4055
02EB 0  CODA            N500  LD        EA        LOAD EFFECTIVE ADDRESS       AM 4060
02EC 01 4C200365              BSC   L  XXX,Z     GO TO XEQ ON NOT ZERO        AM 4065
                        ***************************************************   AM 4070
                        *     EFFECTIVE ADDRESS IN RESIDENT MONITOR        *   AM 4075
                        ***************************************************   AM 4080
                        *  IF THE EFFECTIVE ADDRESS IS WITHIN THE MONITOR  *   AM 4085
                        *  AREA, THE PROGRAM IS NOT ALLOWED TO ENTER THE   *   AM 4090
                        *  MONITOR.  IF THE ATTEMPTED ENTRY POINT IS AN    *   AM 4095
                        *  I/O ERROR TRAP IN 'LIST', OR IS THE DUMP        *   AM 4100
                        *  ENTRY, THEN THE ENTRY IS MADE FROM THIS AMS     *   AM 4105
                        *  ROUTINE RATHER THAN THE CALLING PROGRAM.        *   AM 4110
                        ***************************************************   AM 4115
02EE 0  COD7            N501  LD        EA        LOAD EFFECTIVE ADDRESS       AM 4120
02EF 0  FC77                  EOR       IOCT      COMPARE WITH I/O OFF INDIC-  AM 4125
                        *                         *ATOR ADDRESS                AM 4130
02F0 01 4C180365              BSC   L  XXX,+-    GO TO XXX ON ZERO            AM 4135
02F2 0  4C7C                  BSI       BITS      RETURN BRANCH TO BITS        AM 4140
02F3 0  FC00                  DC        /FCC0       CHECK FOR LONG             AM 4145
02F4 0  4400                  DC        /4400       BSI INSTRUCTION (010001).  AM 4150
02F5 0  7C29                  MDX       N503        GO TO N503 IF TEST FALSE   AM 4155
02F6 0  6106                  LDX     1 6         ENTER INDEX 1 WITH 6         AM 4160
02F7 0  COCE            RT    LD        EA        LOAD EA FOR COMPARISON       AM 4165
02F8 01 F50000CE              EOR   L1 LIST      COMPARE AN ENTRY ADDRESS     AM 4170
                        *                         TO THE CORE RESIDENT MONITOR AM 4175
02FA 01 4C180313              BSC   L  X,-+      GO TO X IF ACC ZERO          AM 4180
02FC 0  71FF                  MDX     1 -1        MODIFY XR1 BY -1,SKIP IF ZERO AM 4185
02FD 0  7CF9                  MDX       RT        GO TO RT                     AM 4190
02FE 0  COC7                  LD        EA        LOAD EFFECTIVE ADDRESS       AM 4195
02FF 0  F01E                  EOR       ADMP      COMPARE WITH DUMP ENTRY ADDR. AM 4200
0300 01 4C2003AC              BSC   L  CGA,Z     GO TO CGA IF NOT ZERO        AM 4205
0302 01 658002C1              LDX   I1 ADDR      LOAD XR1 WITH ADDRESS        AM 4210
0304 0  C103                  LD      1 3         LOAD BEGINING ADDRESS        AM 4215
0305 0  D0C7                  STO       D+3       STORE INTO PDMP STATEMENT    AM 4220
0306 0  C1C4                  LD      1 4         LOAD  END ADDRESS            AM 4225
0307 0  D0C6                  STO       D+4       STORE INTO PDMP STATEMENT    AM 4230
0308 01 4400022B              BSI   L  RESTO     RESTORE ACC,EXT,XR1,XR2,C,O. AM 4235
030A                    D     PDMP      *-*,*-*  DUMP CORE AS SPECIFIED BY    AM 4240
                        *                         *THE CALLING PROGRAM.        AM 4245
030F 01 7405024E              MDX   L  ENT5,5    MODIFY RETURN ADDRESS BY 5   AM 4250
0311 01 4C00024F              BSC   L  ENT5+1    RETURN TO TEST NEXT INSTR.   AM 4255
0313 01 4400022B        X     BSI   L  RESTO     RESTORE ACC,EXT,XR1,XR2,C,O. AM 4260
0315 01 448002C6              BSI   I  EA        BRANCH (BSI) TO EFF. ADDR.   AM 4265
0317 01 7402024E              MDX   L  ENT5,2    MODIFY RETURN ADDRESS BY 2   AM 4270
0319 01 4C00024F              BSC   L  ENT5+1    RETURN TO TEST NEXT INSTR.   AM 4275
031C    00C2            IOAR3 BSS   E  2         TOP OF DISK BUFFER 3         AM 4280
031E 0  003F            ADMP  DC        $DUMP     DUMP ENTRY POINT             AM 4285
                        ***************************************************   AM 4290
                        *     EFFECTIVE ADDRESS IN AMS OR ILS AREA         *   AM 4295
                        ***************************************************   AM 4300
                        *  IF THE INSTRUCTION IS OF A TYPE THAT ALTERS     *   AM 4305
                        *  CORE (IE. THE STORE INDICATOR IS EQUAL TO ONE)  *   AM 4310
```

```
                        *  THEN IT IS NOT ALLOWED TC EXECUTE.  IF NOT, IT *
                        *  IS TREATEC AS IF THE EA IS WITHIN UNUSEC CCRE. *
                        **************************************************
031F 0  C0A5            N503  LD       STORE    LCAC STCRE INDICATCR
0320 01 4C2003AC              BSC  L   CGA,Z    GO TO CCMPUTE GRADE IF NCT C
                        *                       STCRE INDICATCR = 0 IF LCAC
                        *                       STCRE INDICATCR = 1 IF STCRE
                        **************************************************
                        *     EFFECTIVE ADDRESS IN UNUSEC CCRE (GARBAGE)  *
                        **************************************************
                        *  IF THE EFFECIIVE ADDRESS (EA) IS WITHIN UNUSED *
                        *  CORE (GARBAGE), THEN THE ADDRESS IS            *
                        *  RECCRDED IN THE GARBAGE TABLE.  IF THAT        *
                        *  ADDRESS IS ALREACY IN THE GARBAGE TABLE, NC    *
                        *  NEW ENTRY IS MADE IN THE TABLE.  IF THE TABLE  *
                        *  BECOMES FULL, THE PROGRAM IS ABBCRTED AND A    *
                        *  SPECIAL ERROR MESSAGE IS PRINTED BY THE OUTPUT *
                        *  PROGRAM.                                       *
                        **************************************************
0322 0  C049            N504  LD       GCTR     LCAD GARBAGE CCUNTER
0323 01 4C180334              BSC  L   PP,-+    GO TC PP IF ZERO
0325 00 D4000001              STO  L   1        STORE INTO INCEX 1
0327 0  C099            BCK   LD       ADDR     LCAC ACC WITH ADDRESS
0328 01 F5000002              EOR  L1  GTBL-1   CCMPARE WITH GARBAGE TABLE
032A 01 4C180365              BSC  L   XXX,-+   GO TO XEC CN ZERO
032C 0  71FF                  MDX    1 -1       MODIFY XR1 BY -1,SKIP IF ZERO
032D 0  70F9                  MDX      BCK      GO TO BCK IF NC SKIP
032E 0  C03D            PAST  LD       GCTR     LOAD GARBAGE COUNTER
032F 0  903A                  S        D64      SUBTRACT 64 (LENGTH OF GTBL)
0330 01 4C280334              BSC  L   PP,Z+    GO TC PP IF NEGATIVE
0332 0  621A                  LDX    2 /1A      ENTER XR2 WITH /1A AS INDIC.
0333 0  7089                  MDX      CG       GO TC CCMPUTE GRADE
0334 01 7401036C        PP    MDX  L   GCTR,1   INCRIMENT GARBAGE COUNTER
0336 01 6580036C              LDX  I1  GCTR     LOAD XR1 WITH GARBAGE CCUNTER
0338 0  C088                  LD       ADDR     LOAD ACC WITH ADDRESS
0339 01 D5000002              STO  L1  GTBL-1   STORE INTO GARBAGE TABLE
033B 0  7029                  MDX      XXX      GO TO XEC
                        **************************************************
                        *     EFFECTIVE ADDRESS IN SUBRCUTINES            *
                        **************************************************
                        *  IF THE EFFECTIVE ADDRESS IS WITHIN THE SUBRCU- *
                        *  TINES, IT IS PREMITED TO EXECUTE IF THE        *
                        *  INSTRUCTION IS WITHIN THE SUBRCUTINES, CR IF   *
                        *  THE INSTRUCTION IS A VALIC CALL ENTRY TO THE   *
                        *  SUBROUTINE AREA THROUGH THE CALL TRANSFER      *
                        *  VECTOR.  IN THIS LAST CASE, THE MCN INDICATCR  *
                        *  IS SET TO INDICATE THAT THE PROGRAM IS VALICLY *
                        *  WITHIN THE SUBROUTINE AREA.  IF BOTH THESE     *
                        *  TESTS FAIL, THE EFFECTIVE ADDRESS IS TREATED   *
                        *  AS IF WITHIN AMS CR ILS AREA (IE. IT IS WITHIN *
                        *  AN AREA THAT MUST NOT BE ALTERED.)             *
                        **************************************************
033C 0  C087            N506  LD       MON      LOAC MON INDICATOR
033D 01 4C280365              BSC  L   XXX,Z+   GO TC XEC IF PROGRAM IS IN
                        *                       SUBROUTINE AREA
033F 0  402F                  BSI      BITS     RETURN BRANCH TO BITS ROUTINE
0340 0  FF80                  DC       /FF80      CHECK FOR BSI IO
0341 0  4480                  DC       /4480      INSTRUCTION (010001001).
```

```
0342 0  7C0C              MDX      N503      GO TO N503 IF CHECK FALSE
0343 01 C4C002C3          LD    L  INST+1  PUT ADDRESS PORTION INTO ACC
0345 0  9027              S        BCALL    SUBTRACT ADDR BEG. OF CALL
0346 01 4C2803AC          BSC   L  CGA,+Z   GO TO CGA ON MINUS
0348 01 94000115          S     L  LCLTV    SUBTRACT LENGTH OF CALL
034A 01 4C1003AC          BSC   L  CGA,-    GO TO CGA ON NOT MINUS
034C 0  7C11              MDX      N508     GO TO N508
                    **************************************************
                    *     EFFECTIVE ADDRESS IN LIBF T V              *
                    **************************************************
                    *  IF THE EFFECTIVE ADDRESS IS WITHIN THE        *
                    *  LIBF TRANSFER VECTOR, THE INSTRUCTION, INDEX  *
                    *  THREE, AND THE EFFECTIVE ADDRESS ARE TESTED   *
                    *  TO DETERMINE IF IT IS A PROPER ENTRY INTO THE *
                    *  LIBF TV.  IF THE TEST FAILS, IT IS TREATED AS *
                    *  IF THE EA WAS WITHIN THE CALL TRANSFER VECTOR. *
                    *  IF THE TEST IS SUCCESSFUL, THEN THE MON       *
                    *  INDICATOR IS SET TO INDICATE THAT IT IS VALID *
                    *  FOR THE PROGRAM TO BE WITHIN THE SUBROUTINE   *
                    *  OR LIBF TV AREAS.                             *
                    **************************************************
034D 0  4C21        N507  BSI      BITS     RETURN BRANCH TO BITS
034E 0  FF00              DC       /FF00      CHECK FOR SHORT BSI INSTR.
034F 0  4300              DC       /4300      WITH XR3 (01000011).
0350 0  7C11              MDX      N510       GO TO N510 IF CHECK FALSE
0351 00 C4000003          LD    L  3        LOAD INDEX 3
0353 0  F074              EOR      SPXR3+1 COMPARE WITH PROPER VALUE XR3
0354 01 4C2003AC          BSC   L  CGA,Z    GO TO CG ACTUAL IF NOT ZERO
0356 0  C017              LD       BLBTV    LOAD LOW END ADDR LIBF TV
0357 01 940002C6          S     L  EA       SUBTRACT EFFECTIVE ADDRESS
0359 0  1890              SRT      16       SHIFT INTO A TWO WORD OPERAND
035A 0  A80E              D        D3       DIVIDE BY THREE
035B 0  1800              RTE      16       PLACE EXT INTO ACC
035C 01 4C2003AC          BSC   L  CGA,Z    GO TO COMPUTE GR IF NOT ZERO
035E 0  C009        N508  LD       DM1      LOAD ACC WITH MINUS ONE
035F 01 D40002C4          STO   L  MON      STORE INTO MON INDICATOR
                    *                       MON = 1 WHILE IN MONITOR
                    *                       MON = 0 WHILE IN MAINLINE
                    *                       MON =-1 WHILE IN SUBROUTINES
0361 0  7003              MDX      XXX      GO TO XEQ
                    **************************************************
                    *     EFFECTIVE ADDRESS IN CALL T V              *
                    **************************************************
                    *  IF THE INSTRUCTION IS OF A TYPE THAT ALTERS   *
                    *  CORE, IT WILL NOT BE PERMITTED TO EXECUTE.    *
                    *  IF IT IS NOT OF A TYPE THAT ALTERS CORE, IT   *
                    *  WILL BE PERMITTED TO EXECUTE, WITHOUT AN ENTRY *
                    *  IN THE GARBAGE TABLE.                         *
                    **************************************************
0362 01 740002C5    N510  MDX   L  STORE,0 SKIP IF STORE INDIC. IS ZERO
0364 0  7047              MDX      CGA      GO TO CGA IF NOT ZERO
                    *                        STORE INDICATOR = 0 IF LOAD
                    *                        STORE INDICATOR = 1 IF STORE
0365 01 4C8002DF    XXX   BSC   I  TSTEA    EXIT FROM TEST EA ROUTINE
                    **************************************************
                    *     CONSTANTS FOR USE BY LOWER HALF            *
                    **************************************************
0367 0  0032        IOCT  DC       $IOCT
```

```
0368 0  FFFF       CM1   DC      -1         CCNSTANT
0369 0  00C3       D3    DC      3          CCNSTANT
036A C  0C40       D64   DC      64         CCNSTANT
036B C  0C32       H32   DC      /32        CONSTANT
036C 0  0C0C       GCTR  DC      0          GARBAGE CCUNTER
C36D 0  0C0C       BCALL DC      *-*        BEGINING CF CALL TV
036E 0  0000       BLBTV DC      *-*        BEGINING CF LIBF T V
                   ************************************************
                   *                                              *
                   *     B I T S   RCUTINE                        *
                   *                                              *
                   ************************************************
                   *  ROUTINE TC TEST BITS CF THE INSTRUCTICN.    *
                   *  FIRST WORD OF CALLING SEQUENCE INDICATES WHICH *
                   *  BITS ARE TC BE TESTED, THE SECCND TELLS WHICH  *
                   *  OF THESE MUST BE SET FCR TEST TO HCLD.  EXITS  *
                   *  AT THIRD WCRD IF THE TEST DCES NCT HCLD, AT    *
                   *  THE FOURTH WORD IF IT DCES HCLD.              *
                   ************************************************
036F 0  0C00       BITS  DC        *-*      ENTRY PCINT FCR BITS RCUTINE
0370 01 C40002C2         LC   L  INST       LOAD INSTRUCTICN FCR TESTING
0372 C  6909             STX   1 SXR1+1     SAVE INDEX 1
0373 01 6580036F         LDX  I1 BITS       LOAD RETURN ACDR INTO XR1
0375 0  E1C0             AND  X1 0          LEAVE SET CNLY BITS DESIRED
0376 0  F1C1             ECR  X1 +1         CCMPLEMENT DESIRED BITS
0377 0  4818             BSC     +-         SKIP CN NCT ZERC (TEST FALSE)
0378 0  7101             MOX   1 +1         INCRIMENT XR1 IF TEST HCLDS
C379 0  7102             MDX   1 +2         MODIFY XR1 BY TWO
037A C  6903             STX   1 RETU+1     STCRE XR1 INTC ACCR OF INSTR
037B 00 65000000   SXR1  LDX  L1 *-*        RESTCRE INDEX 1
037D 00 4C000000   RETU  BSC  L  *-*        EXIT FRCM BITS ROUTINE
                   ************************************************
                   *                                              *
                   *     IOND - ROUTINE TC WAIT FOR ALL I/C CFF   *
                   *                                              *
                   ************************************************
037F 0  0000       IOND  DC      *-*        ENTRY PCINT FCR IONC
0380 0  4838             BSC     +-Z        SKIP
0381 0  3000       BACKB WAIT               WAIT FCR INTERRUPT
C382 0  610A             LDX   1 10         LOAC XR1 WITH TEN
C383 01 0D0000C2   LOOPB XIO  L1 IOCCB-2    SENSE CSW FCR CISK
0385 0  E00E             AND     H1000      AND CUT ALL BUT BUSY BIT
0386 01 4C200381         BSC  L  BACKB,Z    GC TC BACKB IF NOT ZERO
0388 0  71FE             MOX   1 -2         MODIFY XR1 BY -2,SKIP IF ZERO
C389 0  7CF9             MDX     LCOPB      GC TO LCCPB (IF NC SKIP)
038A C1 747F0392   ICH   MDX  L  ICH1,127   *
038C 0  7CFD             MDX     ICH          * WAIT FCR APROX.
C38D 01 743F0393         MOX  L  ICH2,63    * THREE SECCNDS
038F C  7CFA             MOX     ICH        *
0390 01 4C80037F         BSC  I  IONC       RETURN TC CALLING PCINT
0392 0  0000       ICH1  DC      0          CCUNTER
0393 C  0000       ICH2  DC      0          CCUNTER
0394 0  1000       H1000 DC      /1000      CONSTANT
                   ************************************************
                   *                                              *
                   *     READ RCUTINE - READS CON SW. INTO ACC    *
                   *                                              *
                   ************************************************
```

```
C395 C  OCCC        REAC  CC       *-*     ENTRY PCINT FCR REAC
C396 C  08C3              XIC      ICCCN   REAC THE CONSCLE SWITCHES
C397 C  CCC4              LD       RPAUS   LOAC THE NUMBER REAC IN
C398 01 4C800395          BSC   I  REAC    RETURN TO CALLING PCINT
C39A    OCCC        ICCCN BSS   E  0       IOCC TO REAC THE CONSOLE
C39A 1  C39C              DC       RPAUS   ENTRY SWITCHES INTC CORE
C39B C  3ACC              DC       /3A00   AT LOCATION RPAUS.
C39C C  OCCC        RPAUS CC       *-*     CON SW REAC INTC HERE
                    ************************************************
                    *                                              *
                    *     DUMP -- ROUTINE TO DUMP CORE IF SW 14 SET *
                    *                                              *
                    ************************************************
C39D C  OCCC        DUMP  CC       *-*     DUMP ENTRY PCINT
C39E C  4CF6              BSI      REAC    REAC CON SW INTO ACC
C39F C  ECCB              AND      D2      REMCVE ALL BUT BIT 14
C3AC C1 4C98039D          BSC   I  DUMP,+- RETURN TC CALLING PCINT CN C
C3A2 C  CCCB              LD       BLBTV   LOAC ACCRESS BEGINING LIBF TV
C3A3 C  DCC3              STO      DMP+3   STORE AS PARAMETER FOR DUMP
C3A4                DMP   PDMP  -*,*-*     DUMP ALL BUT UNUSEC CORE
C3A9 C1 4C8C039D          BSC   I  DUMP    RETURN TO CALLING PCINT
C3AB C  OCC2        D2    CC       2       CONSTANT
                    ************************************************
                    *                                              *
                    *     C G   - CCMPUTE GRACE                    *
                    *                                              *
                    ************************************************
                    *  THIS ROUTINE IS ENTEREC WHEN IT IS DESIRED TC *
                    *  ABCRT THE CURRENT PROGRAM AND CUTPUT THE      *
                    *  REASONS FOR ABORTING FURTHER EXECUTION.  THIS *
                    *  IS ACCOMPLISHEC BY PLACING INTO CORE WITHIN   *
                    *  THIS PRCGRAM ANY PARAMETERS WHICH MIGHT BE    *
                    *  NEEDED BY THE CUTPUT RCUTINE, WRITING THIS    *
                    *  PRCGRAM ITSELF ON THE DISK, THEN LINKING TO   *
                    *  THE OUTPUT ROUTINE CALLEC CBUG.  THIS CUTPUT  *
                    *  RCUTINE IS RESPCNSABLE FCR INTERPRETING THESE *
                    *  PARAMETERS ON THE DISK ANC CUTPUTING THEM IN  *
                    *  REACABLE FCRM CN THE PRINCIPLE OUTPUT CEVICE. *
                    ************************************************
C3AC    OCCC        CGA   BSS      0       ENTRY PT FCR CCMPUTE GRACE
C3AC C  4CD2              BSI      IONC    WAIT FOR ALL I/O OFF
C3AD C  4CE7              BSI      REAC    REAC CON SW INTO ACC
C3AE C  EC4D              AND      H0001   REMCVE ALL BUT LOW BIT
C3AF C1 4C1803C4          BSC   L  NCWT,+- GC TC NCWT CN ZERC
C3B1 CC C4000C02          LD    L  2       LOAC ACC WITH INDEX 2
C3B3 C1 E4CC024C          AND   L  H03FF   MAKE CP CODE ZERO
C3B5 C  OCC5              STC      WAITE   STORE AS WAIT INSTRUCTION
C3B6 C1 C4CCC2C6          LC    L  EA      LCAC EFFECTIVE ADDRESS
C3B8 C  1800              RTE      16      MOVE ACC TO EXT
C3B9 C1 C40CC2C1          LD    L  ADCR    LOAC ADCR OF INSTRUCTION
C3BB C  3COC        WAITE WAIT             WAIT FOR OPERATCR
C3BC C  4CEC              BSI      DUMP    DUMP CORE IF SWITCH 14 IS CN
C3BD C  3000              WAIT             WAIT FOR OPERATOR
C3BE C  4CC6              BSI      REAC    REAC CON SW INTG ACC
C3BF C1 E4C0C2C8          AND   L  C4      REMCVE ALL BUT BIT 13
C3C1 C1 4C2C0200          BSC   L  XEQ,Z   GO TC XEC IF NCT ZERO
C3C3 C  4838              BSC      +-Z     SKIP UNCONCITIONAL
C3C4 0  4CD8        NOWT  BSI      DUMP    DUMP CORE IF SWITCH 14 IS ON
```

```
03C5 0  6B2F            STX   3 XR3      STORE XR3 INTO XR3               AM 5475
03C6 0  6A2D            STX   2 XR2      STORE XR2 INTO XR2               AM 5480
03C7 00 67000000  SPXR3 LDX  L3 *-*      RESTORE PROPER VALUE XR3         AM 5485
03C9 0  CB7A            LDD   3 122      LOAD FIRST 2 WORDS OF FAC        AM 5490
03CA 0  D823            STD     SAVF1    SAVE FOR OUTPUT ROUTINE          AM 5495
03CB 0  CB7C            LDD   3 124      LOAD SECOND 2 WORDS OF FAC       AM 5500
03CC 0  D823            STD     SAVF2    SAVE FOR OUTPUT ROUTINE          AM 5505
03CD 0  CB7E            LDD   3 126      LOAD THIRD 2 WORDS OF FAC        AM 5510
03CE 0  D823            STD     SAVF3    SAVE FOR OUTPUT ROUTINE          AM 5515
03CF 01 CC000238        LDD   L INSCT    LOAD DOUBLE INSTRUCTION CT       AM 5520
03D1 0  A833            D       D1E4     DIVIDE BY 10000                  AM 5525
03D2 01 DC000238        STD   L INSCT    STORE DOUBLE INSTR COUNT         AM 5530
03D4 00 C4000032        LD    L $IOCT    LOAD I/O BUSY INDICATOR          AM 5535
03D6 0  D01F            STO     SIOCT    SAVE IN SIOCT                    AM 5540
03D7 0  1010            SLA     16       CLEAR ACC                        AM 5545
03D8 00 D4000032        STO   L $IOCT    CLEAR I/O BUSY INDICATOR         AM 5550
03DA 00 D40000EE        STO   L $DBSY    CLEAR DISK BUSY INDICATOR        AM 5555
03DC 0  6106            LDX   1 6        ENTER INDEX 1 WITH 6             AM 5560
03DD 0  C824      DLOOP LDD     IOAR     LOAD SECTOR LENGTH AND ADDR.     AM 5565
03DE 0  C01A            LD      D320     LOAD 320 AS SECTOR LENGTH        AM 5570
03DF 01 D08003FB        STO  I1 DPAR-1   STORE AT TOP OF BUFFER           AM 5575
03E1 01 CD0003FA        LDD  L1 DPAR-2   LOAD DISK PARAMETERS             AM 5580
03E3 00 440000F2        BSI   L DZ000    GO TO DISK ROUTINE               AM 5585
03E5 01 74010403        MDX   L IOAR+1,1 MODIFY SECTOR ADDRESS BY ONE     AM 5590
03E7 0  71FE            MDX   1 -2       MODIFY XR1 BY -2,SKIP IF ZERO    AM 5595
03E8 0  70F4            MDX     DLOOP    GO TO DLOOP ON NO SKIP           AM 5600
03E9 00 040A41E3        LINK    DBUGT    CALL LINK TO DBUGT               AM 5605
03EE    0002      SAVF1 BSS   E 2        LOCATION TO SAVE FAC             AM 5610
03F0    0002      SAVF2 BSS   E 2        LOCATION TO SAVE FAC             AM 5615
03F2    0002      SAVF3 BSS   E 2        LOCATION TO SAVE FAC             AM 5620
03F4 0  0000      XR2   DC      *-*      LOCATION TO SAVE GRADE INC.      AM 5625
03F5 0  0000      XR3   DC      *-*      LOCATION TO SAVE END VALUE       AM 5630
                  *                      *OF INDEX 3                      AM 5635
03F6 0  0000      SIOCT DC      *-*      LOCATION TO SAVE I/O BUSY INC    AM 5640
03F7 0  EEEE      HEEEE DC      /EEEE    CONSTANT                         AM 5645
03F8 0  0020      H20   DC      /20      CONSTANT                         AM 5650
03F9 0  0140      D320  DC      320      CONSTANT                         AM 5655
03FA 0  2000      H2000 DC      /2000    CONSTANT                         AM 5660
03FC    0000      H0001 BSS   E 0        CONSTANT (ONE)                   AM 5665
03FC    0000      DPAR  BSS   E 0        TABLE OF DISK PARAMETERS         AM 5670
03FC 0  0001            DC      1        DISK WRITE                       AM 5675
03FD 1  031C            DC      IOAR3    ADDRESS OF DISK BUFFER 3         AM 5680
03FE 0  0001            DC      1        DISK WRITE                       AM 5685
03FF 1  01CE            DC      IOAR2    ADDRESS OF DISK BUFFER 2         AM 5690
0400 0  0001            DC      1        DISK WRITE                       AM 5695
0401 1  0000            DC      IOAR1    ADDRESS OF DISK BUFFER 1         AM 5700
0402 31 22C65109  IOAR  DSA     SAVGR    DISK FILE LENGTH, SECTOR         AM 5705
                  *                      *ADDRESS, AND NO OF SECTORS.     AM 5710
0405 0  2710      D1E4  DC      10000    CONSTANT                         AM 5715
0001              'CMON EQU     1        LENGTH OF COMMON                 AM 5720
0004              'HWET EQU     4        LENGTH OF CORE IMAGE HEADER      AM 5725
0008              'TVWC EQU     8        LENGTH OF TRANSFER VECTOR        AM 5730
0009              'WCNT EQU     9        LENGTH OF CORE LOAD              AM 5735
000A              'XR3X EQU     /A       SETTING FOR INDEX 3              AM 5740
0018              'ITCK EQU     24       INTERRUPT ENTRY TO KBD/CON PR    AM 5745
000E              $CORE EQU     /000E    SIZE OF CORE                     AM 5750
0028              $PRET EQU     /0028    PRE-OP I/O ERROR TRAP            AM 5755
0032              $IOCT EQU     /0032    I/O BUSY INDICATOR               AM 5760
```

```
C038                 $EXIT EQU      /003B   ENTRY PCINT FOR EXIT          AM 5765
C03F                 $DUMP EQU      /003F   DUMP ENTRY POINT              AM 5770
C07B                 $WRD1 EQU      /007B   LCADING ACCR CF THE CORE LCAC AM 5775
CC81                 $PST1 EQU      /0081   POST-CP I/C ERRCR TRAP L 1    AM 5780
C085                 $PST2 EQU      /0085   PCST-CP I/C ERROR TRAP L 2    AM 5785
C089                 $PST3 EQU      /0089   POST-CP I/C ERROR TRAP L 3    AM 5790
C08D                 $PST4 EQU      /008D   POST-CP I/C ERRCR TRAP L 4    AM 5795
0091                 $STCP EQU      /0091   PROGRAM STCP KEY TRAP L 5     AM 5800
COEE                 $CBSY EQU      /00EE   DISK BUSY INCICATOR           AM 5805
COF2                 DZ00C EQU      /00F2   DISK ROUTINE ENTRY ACCRESS    AM 5810
                     ******************************************************* AM 5815
0406    OCOC         EAMS  BSS   E  C       LAST LOCATION IN AMS PRCG.    AM 5820
C406                       END            END OF AM PRCGRAM               AM 5825

   OCC OVERFLCW SECTCRS SPECIFIEC
   OCC OVERFLCW SECTCRS RECUIREC
   2C5 SYMBCLS DEFINEC
    NC ERRCR(S)  FLAGGED IN ABOVE ASSEMBLY

// DUP

*DELETE          UA   AM
CART IC OC26   DB ACCR   4E41    CB CNT    C034

*STORE       WS  UA   AM
CART ID OC26   DB ACDR   4EE2    CB CNT    CC34
```