HONEYWELL EDP

SOFTWARE MANUAL

SERIES 200

INTRODUCTION TO SERIES 200/OPERATING SYSTEM-MOD 1 (TAPE RESIDENT)

GENERAL SYSTEM:	SERIES 200/OPERATING SYSTEM - MOD 1
SUBJECT:	General Description of the Series 200/Operating System - Mod 1 (Tape Resident).
SPECIAL INSTRUCTIONS:	This software manual completely supersedes the publication entitled Introduction to Series 200/ Operating System - Mod 1, Order Number 258, dated March 30, 1966.

DATE: August 29, 1966

FILE NO.: 123.0005.001C.1-258*

9067
8766
Printed in U.S.A.

*When ordering this publication please specify Title and Underscored portion of File Number.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I
INTRODUCTION

The Series 200/Operating System represents the result of 15 years of evolution and development. Honeywell's role in this development has been particularly significant, starting with the operating system on the D-1000, continuing through the Executive System and the ADMIRAL System for the H-800/1800, and culminating in the Series 200/Operating System.

THE EVOLUTION AND DEVELOPMENT OF OPERATING SYSTEMS

In the early days of computers, the programmer not only wrote the programs but executed them as well. He set up the programs, loaded them with his own loading routine, monitored their execution, and debugged them on the machine. In effect, it was a one-man operation from beginning to end, with the programmer controlling all aspects of the program's execution via manual intervention at the console. Following the execution of one program, the next programmer stepped in and took over with his own unique methods of operation.

It soon became evident that this mode of operation was economically unfeasible. In many cases, the setup time for a program far exceeded the run time. The lack of uniformity in setup and operating procedures resulted in costly mistakes and made any communication or standardization between programs impossible. In addition, each programmer had to write his own load routines, input/output routines, error routines, etc., instead of being able to incorporate those already written.

As computers developed, the need for a better system of operating also developed. More comprehensive applications resulted in the technique of assigning a team of programmers to each application or job. The fact that some sort of communication had to exist between these job-related programs required standard methods of programming and operation. Users began to develop routines for such common and repetitive operations as loading, dumping, sequencing from program to program, and debugging, and to incorporate these standardized routines into their programs. Although no standards existed among the various users, at least some degree of standardization was realized within each individual installation. Because these aids were used to control program execution, they were called supervisory or executive programs. It was from these that the monitors and operating systems of today evolved.

PHILOSOPHY OF AN OPERATING SYSTEM

The operating system concept is based on one primary goal: the increase of data throughput by assisting the user in utilizing to the fullest extent the hardware and software available. To fulfill this goal, an operating system should:

1. Be simple and convenient to use.
2. Relieve the operator of the necessity for complex procedures and constant supervision.
3. Eliminate unnecessary idle time spent on job setup and last-minute planning.
4. Be flexible and expandable - adaptable to the operating environment in terms of hardware configurations, software complements, and operating policies of the user.
5. Minimize the turnaround time between the submission of a job and the return of the results.
6. Be economical, efficient, and easily maintained.

To the extent that the above goals have been achieved in the design of an operating system, the operator can communicate with the system rather than with the individual programs, and the system operating procedures become standardized. Likewise, because of such features as a comprehensive program test system, the programmer can direct the entire testing procedure by simply preparing the control deck to be used. Thus, both the operator and the programmer are removed to an off-line status and are prevented from any wasteful contact with the computer.

In short, an operating system can be thought of as a framework within which the user's data processing applications can be written, prepared, and executed. To this end, an operating system consists of many language processing and service routines designed to aid the user in these activities.

SECTION II

THE SERIES 200/OPERATING SYSTEM - MOD 1 (TAPE RESIDENT)

The Series 200/Operating System - Mod 1 provides the user with a means of unleashing the full power of any Series 200 computer system having a minimum of 12K characters of memory and three tape units. Designed with simplicity, economy, and flexibility in mind, the Mod 1 Operating System offers a long list of significant features molded together into one integrated operating framework.

OPERATING ENVIRONMENT

The Series 200/Operating System - Mod 1 (Tape Resident) is designed for all Series 200 machine configurations having 12K to 65K characters of core storage and from three to six tape units. In addition, the versatility of the system enables the efficient use of other peripheral devices such as punched cards, drum storage, paper tape, communications, and mass memory.

MOD 1 OPERATING SYSTEM PHILOSOPHY

Specifically aimed at the medium-scale, tape-oriented, Series 200 user, the basis of the Mod 1 Operating System philosophy can be found in the three outstanding characteristics of the system: simplicity, efficiency, and flexibility.

Simplicity

The Mod 1 Operating System provides a comprehensive collection of precoded and tested systems functions and programs which relieve the user of a host of complex programming and operating tasks. For example, tape and drum storage input/output routines are provided ready for specialization and insertion, via macro calls, into the user's programs. For the operator, a complete program test package allows for the checkout of an entire series of programs with no operator intervention from the initial pressing of the RUN button to the end of the job.

Standard, easily learned operating procedures simplify the operator's job. The same maintenance programs and standard operating procedures are used to maintain and execute Honeywell software and the user's own object programs.

Efficiency

One of the most important objectives in the design of the Mod 1 Operating System is the efficient and economic use of core storage. First, since the system is completely modular in design, only those functions required for any given operation occupy memory. For example, separate loader routines are provided for loading programs stored on punched cards, magnetic tape, or drum. Each version contains only those functions required for loading from the particular medium. Secondly, routines (or portions of routines) can be assembled and executed in either two-, three-, or four-character addressing mode. Whenever possible, the shortest addressing mode may be used to conserve core requirements. For example, the three-character address mode version of Floating Tape Loader-Monitor C occupies approximately 520 to 550 fewer core locations than the four-character version. One proof of core usage efficiency is the fact that Honeywell's Series 200 COBOL compilers require much less memory than any competitive compilers offering similar features.

Through the use of the Mod 1 Operating System, the user realizes savings not only from efficient core usage but also from the reduction of setup and idle time and from the elimination of operations errors. In addition, the assembly, sorting, and compiling functions offer speeds which far exceed those of similarly priced competitive equipment.

Flexibility

Flexibility is provided both by the number and variety of functions offered any by the inherent expansibility of the system. For example, the user has a choice of several programming languages ranging from the assembly-level language of Easycoder to the scientific compiler-level language of Fortran. Further, he can choose between card, tape, or drum program storage and loading. He can add his own coded macro routines to the macro library for insertion into his programs. He can take advantage of any communication devices or console typewriters included as part of his computer configuration. He can use any core memory size from 12K up to 262K and can begin with as few as three tape units.

One of the major benefits derived from this flexible design is that of orderly growth potential by which the user can add both hardware and software as needed and still maintain an integrated system. Coupled with the modular design of the Series 200 hardware and software, the Mod 1 Operating System assures the user of upward program compatibility as his system grows.

PROCESSING STRUCTURE

Fundamental to the design of the Mod 1 Operating System is its functional program modularity.

First, although the concept of a program as the basic logical unit is still retained, a program is segmented into one or more loading units. Each loading unit consists of a portion of the code for a particular program and can be individually searched for and loaded by a single call to the Loader-Monitor. This concept of program segmentation provides increased operational flexibility plus a great reduction in the amount of memory core required by a program.

Secondly, related programs can be combined into job-oriented groupings. The ability to store not only several "jobs" but also several versions of each job on a single program tape allows the user to adopt the building-block approach in the implementation of these jobs. For example, a program tape might contain all of the current production programs for the application, checked-out programs to be incorporated in the future, and programs yet to be tested. During any execution of the job, the desired programs are automatically selected, loaded, and executed in whatever sequence the user elects. This is accomplished through the common interface of the Mod 1 Operating System using one or both of two methods: internal control via coding within the programs themselves; external control via a Console Call card control deck.

COMPONENTS OF THE MOD 1 OPERATING SYSTEM

The Mod 1 Operating System contains many subsystems and routines designed to eliminate much of the work of both the programmer and the operator. These components can be divided into three types: those which aid in the preparation of source-language programs and their translation into machine language; those which perform editing and maintenance on machine-language programs; and those which control the execution of programs. These three groupings are shown in Figure 2-1.

Program preparation involves both the writing and translation of the user's own programs and routines and the incorporation of Honeywell-supplied utility programs and routines. The user has a choice of three methods of writing and translating his own programs: assembly systems, compiler systems, and translators. These are collectively called language processors. Honeywell-supplied programs, called utility programs, perform such common data processing functions as the sorting and collating of data files, the manipulation of tapes, and the solving of mathematical formulas. Since the machine-language format of all these programs is the same, regardless of their source, they can be combined and executed in any order.

The editing and maintenance programs enable the user to select, rearrange, and combine all types of object programs within the Mod 1 Operating System onto a single program run tape from which he can load and execute his jobs. Facilities are also provided for modifying and correcting programs.

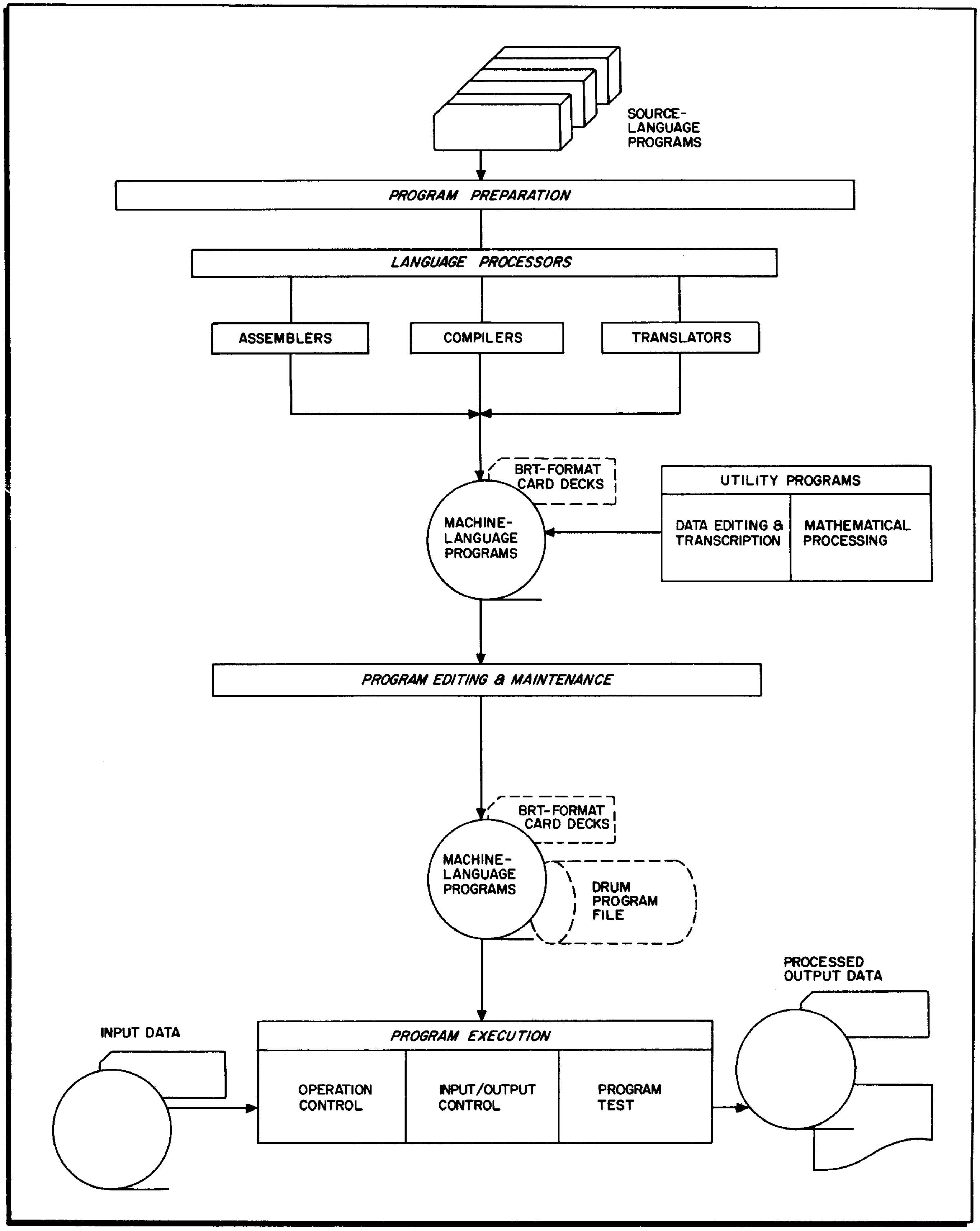


Figure 2-1. Series 200/Operating System - Mod 1

In the area of program execution, the Mod 1 Operating System offers several important capabilities: operation control, input/output control, and program test. Operation control is performed by some version of the loader-monitor which can search for and load any program called for by either the operator or the current program. It can also execute programs on a job basis — i.e., automatically search for, load, and execute an entire series of programs related to one application or job. An associated interrupt control routine can extend these capabilities to the loading and executing of two programs at the same time by utilizing the simultaneity and interrupt capabilities of the Series 200 hardware. File input and output operations are controlled by appropriate input/output routines inserted and specialized within each program at assembly or compilation time via user-specified macro calls. Automatic program test and checkout procedures are provided within the system for the efficient testing and debugging of a single program or a whole series of programs immediately following assembly, compilation, modification, or specialization of the programs.

The following sections present these various components in detail.

SECTION III

PROGRAM PREPARATION AND MAINTENANCE

The Mod 1 Operating System provides the user with a comprehensive language processing capability in the form of assemblers, compilers, and conversion programs. Utility programs complement this capability by offering precoded routines which perform common data processing functions and which can be specialized to the user's individual needs. Editing and maintenance programs offer a means of creating and maintaining symbolic and machine-language program files.

LANGUAGE PROCESSING

Language processing programs are provided to translate programs written in several different source languages into a single machine-language format; thus, the output from the various language processors can be combined on a single program run tape and executed in any order. In writing his programs, the user chooses from the following three languages the one best suited to the application and to his own background and experience.

1. Easycoder - A general-purpose assembly-level language which combines ease of use with power and flexibility. To express the logic of his program, the user employs easily remembered mnemonic op codes and references memory locations by either absolute decimal numbers or symbolic tags. The Easycoder system includes an Easycoder assembler which translates the source language to machine language, a library processor which enables the user to incorporate precoded routines into his program, an analyzer program which prints a cross-reference listing of all symbolic tags used, and a symbolic program maintenance facility which permits the storage and updating of symbolic programs on tape.
2. COBOL - A business-oriented compiler-level language which offers simplicity of format, shorter training time requirements, and compatibility among different models of computers. The user expresses the logic of his program as a series of English-language statements conforming to standardized COBOL conventions.
3. Fortran - A science-oriented compiler-level language which allows the user to express a wide variety of engineering, scientific, and data processing problems in a familiar format. Library and diagnostic facilities are included in the system.

If the user has already written his programs in the language of a competitive system, he can easily and quickly convert these programs to a format compatible with the Series 200 systems. As a result of the Honeywell Liberator concept, such programs can be translated on both the symbolic and machine-language levels without the aid of simulators. For example, the Easytran Symbolic Translator routines translate programs written in Autocoder, SPS, or mixed

SPS/Autocoder source language into Easycoder source language. An important facet of this translation is that all programs are modified to take advantage of the superior throughput power of the Series 200 hardware and, as a result, usually run in a fraction of their former execution time.

Assembly System

The Easycoder Assembly System for the Mod 1 Operating System comprises four elements:

1. A symbolic language,
2. A library processor,
3. An assembler, and
4. An analyzer.

EASYCODER ASSEMBLY LANGUAGE

The Easycoder Assembly Language is a general-purpose, easy-to-use language designed for all types of applications. It provides the user with a comprehensive set of operation codes with which he can specify the following types of operations:

1. Arithmetic - Offer the user a choice of binary or decimal arithmetic functions. In addition, Easycoder Assembler D on a Series 200 computer with the Scientific Unit provides the user with floating-point capabilities.
2. Logic - Provide the user with the functions of extracting, half-adding, substituting, comparing, conditional and unconditional branching, etc.
3. Control - Enable the user to set and clear punctuation, halt the machine, move data, enter and retrieve data from control memory, and change addressing and sequencing modes. Easycoder Assembler D also allows the controlling of memory barricades with the Storage Protect Feature.
4. Interrupt Control - Enable the user to take advantage of the interrupt functions of the Series 200 hardware.
5. Editing - Permits the use of the extensive power of the Edit Feature in producing financially edited fields on printed reports.
6. Input/Output - Provides two basic instructions which allow the user complete control over all data transfers between the central processor and all peripheral units and over the peripheral units themselves.

Also included in the Easycoder symbolic language are a number of assembly control statements which permit the user to control the assembly process itself.

LIBRARY PROCESSOR

To relieve the programmer of the burden of the repetitive and complex coding of commonly used routines (e.g., input/output procedures), the Easycoder Assembly System includes library facilities. In addition to the basic library of general-purpose routines supplied by Honeywell, the user can add his own often used programs and routines to this library. These routines are stored, in source-language format, on a library source-program tape. By writing call statements (macro instructions) within his source coding and processing his program through a library processor prior to assembly, the user can cause these routines to be selected from the library source-program tape, specialized according to the parameters he has included in the call statements, and inserted into his program.

The two versions of the library processor - Library Processors C and D - provided in the Mod 1 Operating System perform the following three functions:

1. Specialization - The specialization and insertion of macro routines from the library symbolic program tape into Easycoder source-language programs as indicated by the macro calls and associated parameters.
2. Respecialization - The updating of previously processed programs on a source-language program tape (which contains a source-language version and a machine-language version of each program) with new versions of the macro routines incorporated within them. The library processor generates the new, respecialized versions of the macro routines. During the subsequent assembly run, the assembler replaces the object code of the old macro routines with the respecialized versions.
3. Reproduction - The punching of symbolic source decks containing complete source programs from the library source-program tape.

The setup for Library Processors C and D is shown in Figure 3-1.

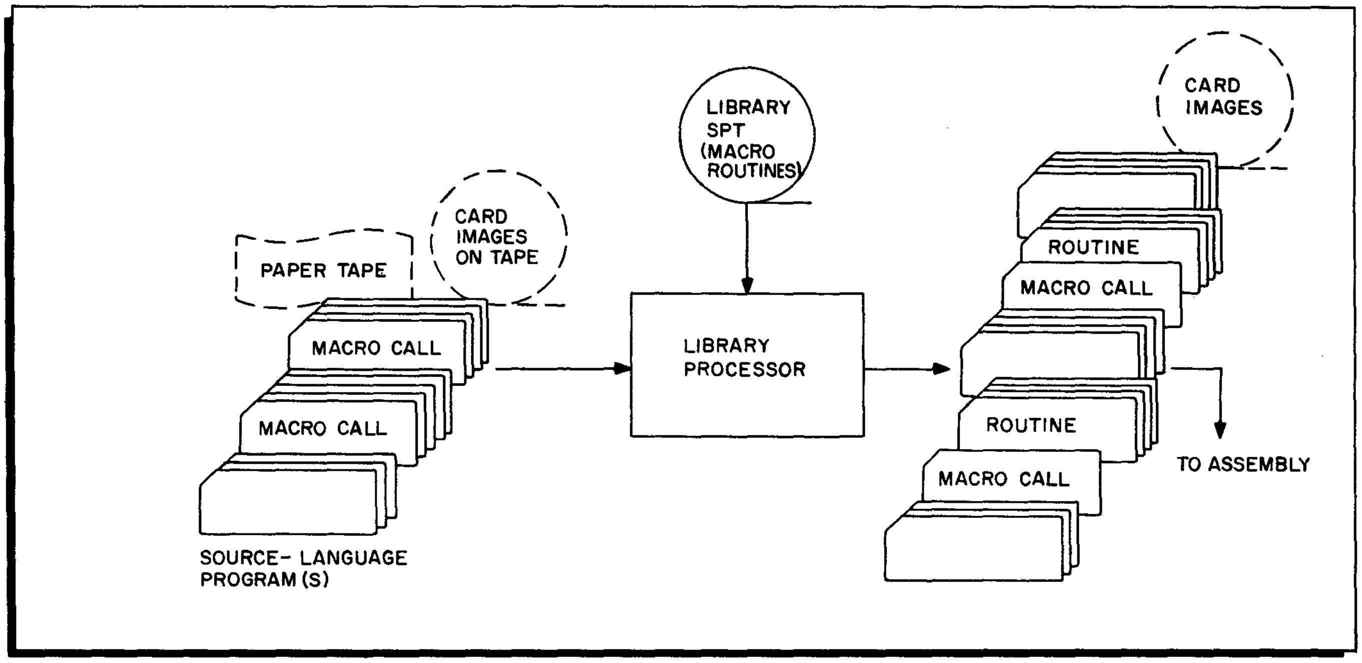


Figure 3-1. The Library Processor

Easycoder Symbolic Card Formats

The need for two versions of the library processor is brought about by the fact that the user has a choice of two symbolic coding formats. The standard coding card format contains a seven-character location field which allows the programmer to specify and use symbolic tags up to six characters in length. The alternate card format contains an 11-character location field and allows the programmer to specify and use symbolic tags up to 10 characters in length. These features are summarized in Table 3-1.

Table 3-1. Library Processors C and D: Features

Library Processor C	Performs the library facility functions of specialization, respecialization, and reproduction on Honeywell-supplied or user-created macro routines stored on a symbolic library tape. Accepts symbolic programs using the standard card format and produces output to Easycoder Assembler C.
Library Processor D	Performs the same functions as the above version. Accepts symbolic programs using the alternate card format and produces output to Easycoder Assembler D.

EASYCODER ASSEMBLER

The Easycoder Assembler translates symbolic coding written in the Easycoder symbolic language into machine language. It writes the assembled programs (in both their symbolic-language format and their machine-language format) onto a symbolic program tape (SPT). If a sufficient number of tape drives are available, it will also write the programs in machine language onto a binary run tape (BRT), from which the programs can be loaded and executed. Should a tape drive not be available, the BRT can be created in a separate run. Optionally, assembled programs can be punched on cards in BRT machine-language format. The cards can then be loaded and the program executed. Facilities are also provided for maintaining programs on the symbolic program tape. Programs can be deleted from or added to an SPT, and corrections can be made to the programs during the same assembly run.

During any single assembly run, the user has a choice of four operating modes:

1. Assembly - Translates programs written in Easycoder symbolic language and places the assembled programs on a symbolic program tape (SPT) in both symbolic form and machine-language form.
2. Selection - Selects the machine-language formats of specified programs from an SPT and places these on a binary run tape or on punched cards. If a sufficient number of tapes is not available during the assembly pass, this mode is utilized following assembly to produce the BRT.

3. Assembly and Updating - Maintains and updates the programs on an SPT in both their symbolic and machine-language formats through the correction of individual programs, the addition of new programs, and the deletion of unwanted programs.

4. Assembly, Updating, and Selection - Performs the same operations as the assembly and updating mode with the added ability to select specified programs and place these programs on tape or on cards in executable form.

The mode chosen by the user depends upon his requirements and upon the number of tape units available. The setup diagrams for the various modes are shown in Figure 3-2.

The Mod 1 Operating System includes two versions of the Easycoder Assembler: Easycoder Assembler C and Easycoder Assembler D. The features provided by the two versions are summarized in Table 3-2.

Table 3-2. Easycoder Assemblers C and D: Features

Easycoder Assembler C	Processes the basic repertoire of symbolic op codes and assembly control instructions. Accepts the standard coding card format and, therefore, operates in conjunction with Library Processor C.
Easycoder Assembler D	Provides, in addition to the features of Easycoder Assembler C, the ability to process memory barricade op codes (require the Storage Protect Feature for execution) and floating-point op codes (require the Scientific Unit for execution). Accepts the alternate card format and, therefore, operates in conjunction with Library Processor D.

ANALYZER

Analyzer C is a powerful programming aid which simplifies the task of analyzing any program written in the Easycoder symbolic language. Analyzer C extracts all symbolic tags, references (to tags, index registers, and absolute addresses), and calls to macro routines and processes them to produce an analysis listing of the program. The listing is arranged in alphanumeric order so that all information about a particular tag, absolute location, or library routine appears together in one place.

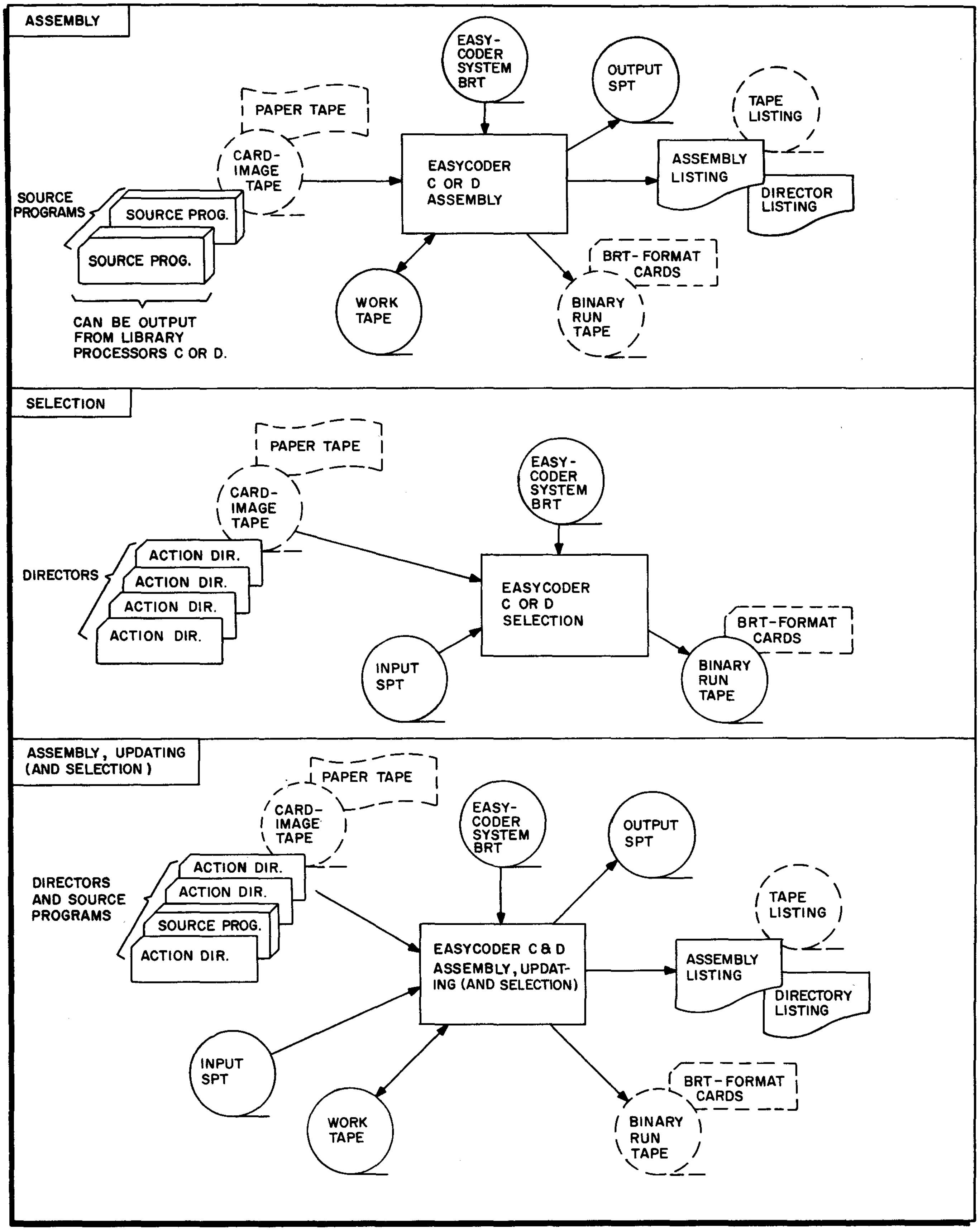


Figure 3-2. Operating Modes of Easycoder Assemblers C and D

The general systems diagram for the program is given in Figure 3-3.

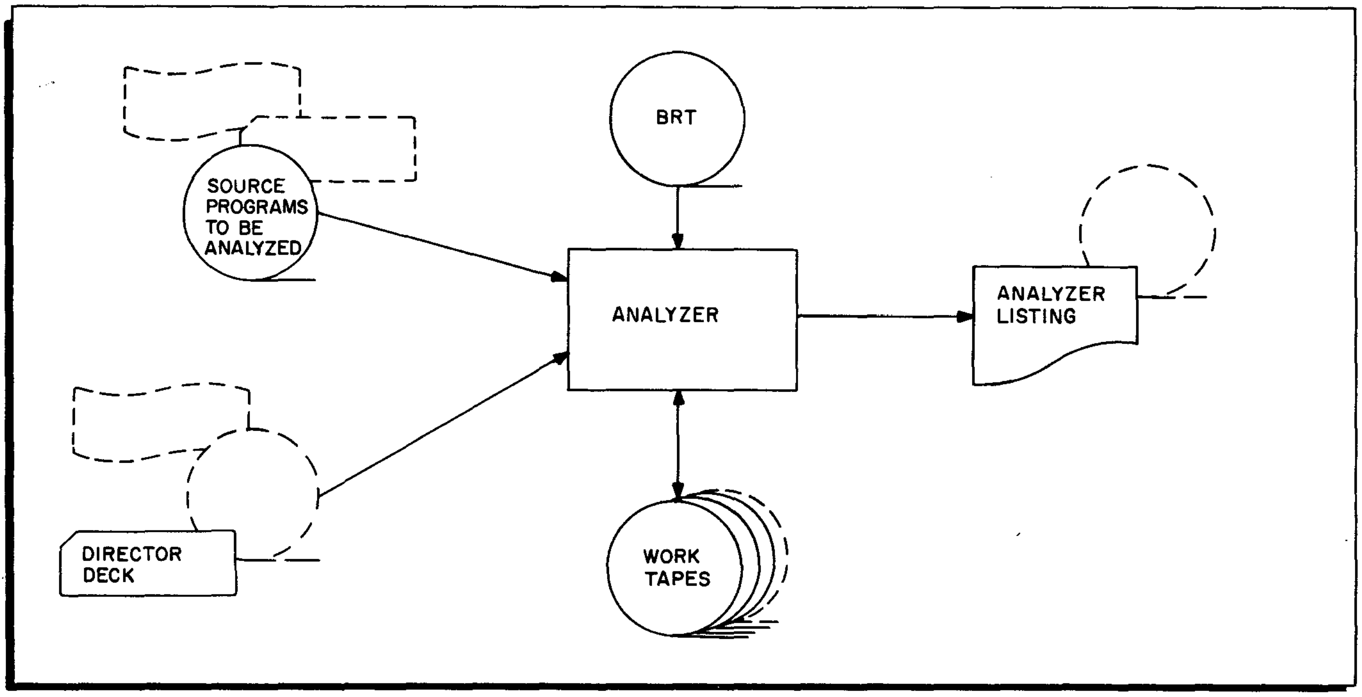


Figure 3-3. Analyzer C Setup

Compiler Systems

The Mod 1 Operating System offers two compiler systems: COBOL (COmmon Business-Oriented Language) for the business user, and Fortran (Formula Translator) for the scientific user. The features implemented and the coding format and vocabulary of each source language are designed to give that particular type of user a familiar and, therefore, easily learned and understood programming language. In addition, both compiling systems offer the following advantages:

1. Inter-system Compatibility - The design and implementation of both compilers are controlled by centralized committees under the sponsorship of the Federal government and are thus standardized among all computer manufacturers. As a result, programs written in the source language of either compiler for one computer system can (with minor modifications due to hardware differences) be recompiled and run on another computer system.

2. Standardization - The fact that both compiler systems are standardized among all computer manufacturers reduces the problems involved in programmer turnover, since any programmer with experience in either compiler language can be trained for a new system in a relatively short time.

3. Intraproject Communication - Intraproject communication is greatly increased and improved by the fact that both source languages are easily understood by those familiar with the application but with little or no experience in programming. Communication is also increased between programmers working on related programs, since they are

all using the same coding conventions, symbolic tags, and other standardized techniques.

4. Fast Program Compilation and Testing - Both systems offer compilation speeds averaging only a matter of a few minutes per source program. Both compilers offer comprehensive diagnostic scans for clerical errors and helpful diagnostic listings of all errors found. As a result, debugging time has been reduced by 50% or more.

5. Reduction of Programmer Training Time - Programmer training (or retraining in the case of a changeover to a new computer) is drastically reduced by the use of these easily learned source languages. Many companies have found that programmers trained in a compiler language reach a satisfactory level of productivity many weeks in advance of those trained in a conventional assembly language.

6. Improved Documentation - The clear and comprehensive listings produced by both compiler systems provide a clear picture of the program's logic and, via diagnostic messages, aid the programmer in achieving his goal of a completely checked-out and operable program. This documentation is especially valuable when a programmer must take over the maintenance of a program written by another programmer.

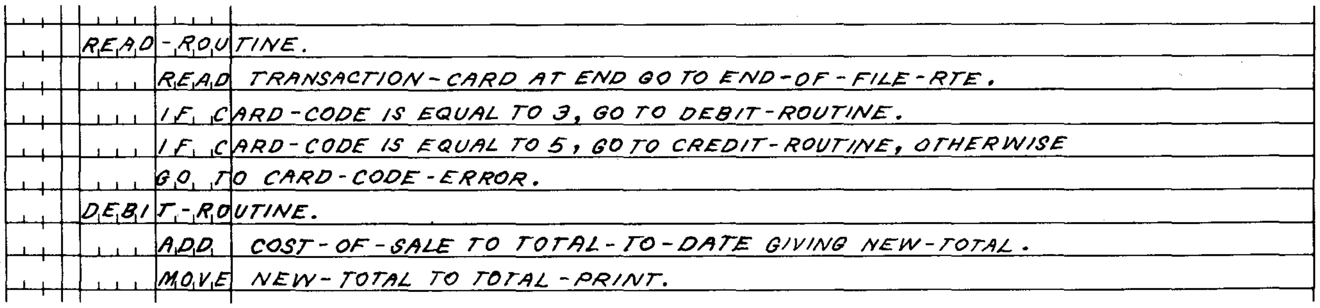
A compiler system consists of two elements:

1. A programming language, and
2. A translator, called a compiler, which translates the programmer's source-language statements or formulas into machine language.

THE COBOL COMPILER SYSTEM

The COBOL Language

The source language of the COBOL compiler system consists of meaningful English-language statements conforming to COBOL conventions. An example of the procedure portion of a COBOL source program is shown in Figure 3-4.



```
READ-ROUTINE.
    READ TRANSACTION-CARD AT END GO TO END-OF-FILE-RTE.
    IF CARD-CODE IS EQUAL TO 3, GO TO DEBIT-ROUTINE.
    IF CARD-CODE IS EQUAL TO 5, GO TO CREDIT-ROUTINE, OTHERWISE
    GO TO CARD-CODE-ERROR.
DEBIT-ROUTINE.
    ADD COST-OF-SALE TO TOTAL-TO-DATE GIVING NEW-TOTAL.
    MOVE NEW-TOTAL TO TOTAL-PRINT.
```

Figure 3-4. Example of the COBOL Source Language

Among the special features included in the COBOL compilers of the Mod 1 Operating System are the following:

1. Internal and Library COPY Functions - Allow the programmer to incorporate within his source-language coding complete file and record descriptions from either some other portion of the program or a source-language library tape.
2. Tape File Handling - The programmer can describe and process five types of tape file formats:
   a. Unblocked, fixed-length records,
   b. Unblocked, variable-length records,
   c. Fixed-blocked, fixed-length records,
   d. Fixed-blocked, variable-length records, or
   e. Variable-blocked, fixed- or variable-length records.

   The compiler can also generate coding to handle tapes recorded in BCD format and/or containing 120-character labels.
3. Two-level Subscripting - Permits the programmer to set up and use two-dimensional tables.
4. Editing Features - Allow the user to express a wide range of report editing, either by means of PICTURE symbols or by means of descriptive edit clauses.
5. PERFORM Verb Options - Four options of the PERFORM verb are implemented to enable the programmer to direct many variations of out-of-sequence processing.
6. ADD, SUBTRACT, and MOVE CORRESPONDING - Permit the programmer to specify in one statement the same action on a series of related items, rather than having to repeat the statement for each item.
7. USE Procedures - Allow the programmer to control the processing of file labels and to specify special procedures to be performed in the case of input/output errors.

The COBOL Compilers

Within the Mod 1 Operating System, COBOL has been implemented by two high-performance, syntax-directed compilers: COBOL Compiler D and COBOL Compiler H. Both possess several unique operating features:

1. Operation is in a batch-compile, load-and-go mode. An entire file of source programs can be compiled under the control of the Mod 1 Operating System without operator intervention.
2. Source-language programs can be maintained on a library tape where they can be easily modified or corrected and recompiled. Portions of source-language coding, such as record descriptions, can also be stored on this tape and can be incorporated into any program through use of the COPY verb.
3. A variety of program testing and debugging aids such as memory dumping, English-language diagnostics, memory mapping, etc., are included.
4. The addresses of peripheral devices can be changed at object execution

time to allow the use of a single program with a variety of different peripheral configurations. For example, if a printer is not available for the running of a program, the output can be assigned to tape for subsequent off-line printing.

The maintenance and updating of source programs and library units stored on tape is performed by the Source Program and Library Update Routine. This routine makes it possible to add, delete, or replace source programs or library units, either in whole or in part, and to create both an updated master source-program and library tape and a library tape alone.

The COBOL Compiler accepts source programs punched on cards or stored on the master source-program and library tape. If library copies are contained within the source programs being compiled, the library tape must be mounted as input. The output of compilation is an object BRT and a complete program listing containing the source-language coding, machine-language coding, memory maps, and diagnostic messages for each program selected for compilation. The object code listing may optionally be suppressed.

The general setup for both of these processes is illustrated in Figure 3-5.

Both versions of the COBOL compilers within the Mod 1 Operating System offer upward compatibility; i.e., source programs written for one version are acceptable input to all larger versions. The features of the two versions are given in Table 3-3.

Table 3-3. COBOL Compilers D and H: Features

COBOL Compiler D	Includes the required elements plus many of the special elective features of the COBOL language. Generates object programs which can use and reference up to 32K characters of memory.
COBOL Compiler H	Implemented in phases. The special added features of each phase are extensions to the language and translation capability of COBOL Compiler D. These extensions include: PHASE I - Four-character addressing, which allows the compiled programs to use and reference up to 262K characters of memory. Further extensions are currently in the planning stage.

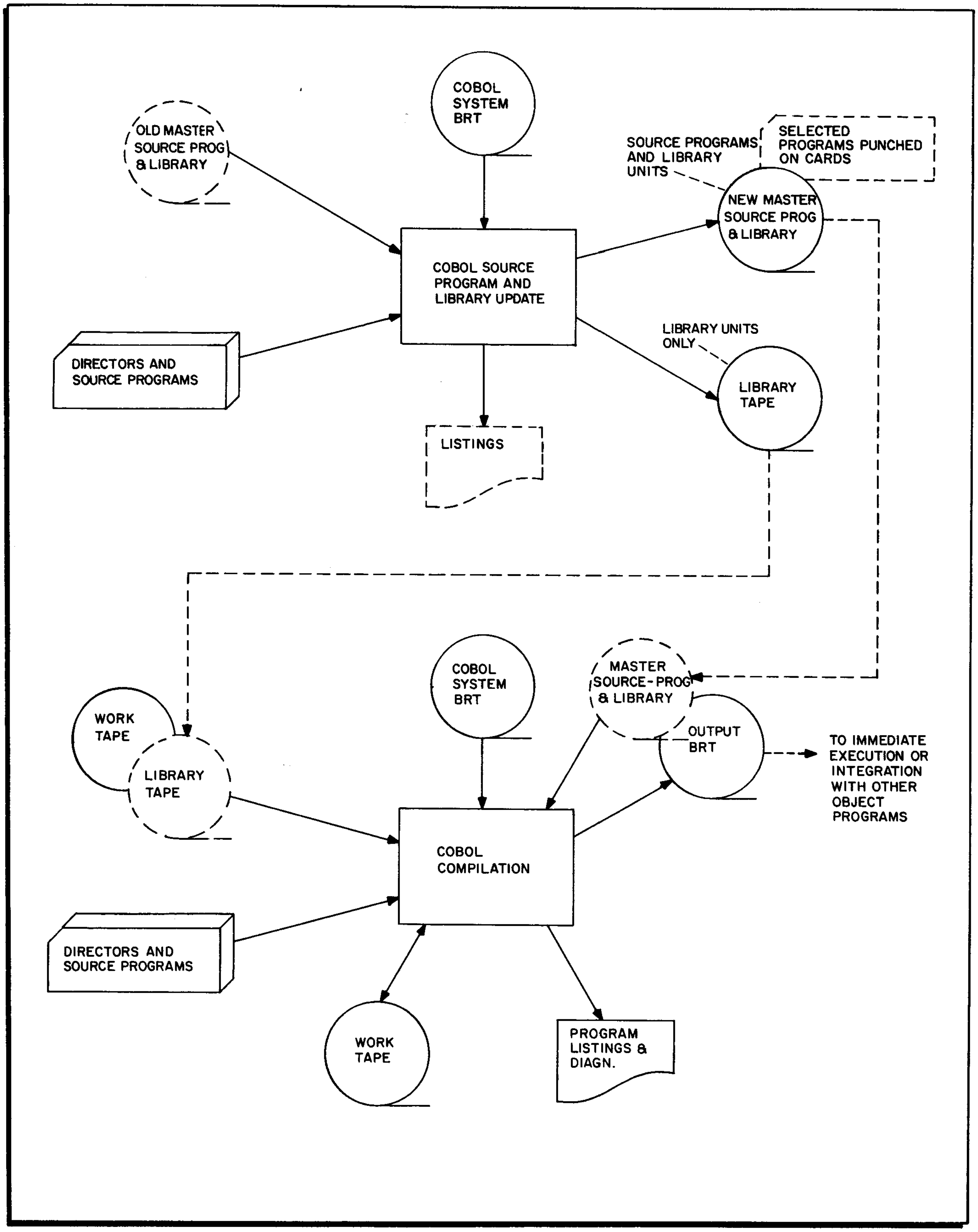

COBOL SYSTEM BRT
OLD MASTER SOURCE PROG & LIBRARY
SOURCE PROGRAMS AND LIBRARY UNITS
SELECTED PROGRAMS PUNCHED ON CARDS
NEW MASTER SOURCE PROG & LIBRARY
COBOL SOURCE PROGRAM AND LIBRARY UPDATE
DIRECTORS AND SOURCE PROGRAMS
LIBRARY UNITS ONLY
LIBRARY TAPE
LISTINGS
COBOL SYSTEM BRT
MASTER SOURCE-PROG & LIBRARY
WORK TAPE
LIBRARY TAPE
OUTPUT BRT
TO IMMEDIATE EXECUTION OR INTEGRATION WITH OTHER OBJECT PROGRAMS
COBOL COMPILATION
DIRECTORS AND SOURCE PROGRAMS
WORK TAPE
PROGRAM LISTINGS & DIAGN.

Figure 3-5. COBOL Compiler System

THE FORTRAN COMPILER SYSTEM

The Fortran Language

The Fortran source language allows the user to express a wide variety of engineering, scientific, and mathematic data processing solutions in a familiar, easily used language format. An example of a mathematical statement written in the Fortran language is shown in Figure 3-6.

```
X=5.*(3.**APW+SQRT(A**2.))/4.*(B**ABS(X))
GO TO 10054
```

Figure 3-6. Example of a Fortran-Language Arithmetic Statement

As implemented in the Mod 1 Operating System, the language includes elements equal to those normally implemented for large-scale competitive compilers. For example, the programmer has the facility of expressing floating-point values ranging in precision from two to twenty characters. All of the important Fortran IV standards, established by the American Standards Association, are implemented in Fortran H. These include such features as logical statements and testing, data initialization, labelled COMMON areas, and type statement declarations. Generated object code can use and reference up to 262K characters of memory. Faster execution speeds are achieved in Fortran H by utilizing the added features and instructions of the scientific hardware option. Additional tape drives and other types of secondary storage allow input and output to be transferred directly to and from these faster media. Yet the smaller version of the compiler (Fortran D) requires only 16K characters of memory, a card reader, four tape drives, a printer, and the advanced programming and edit instructions.

The Fortran Compilers

Both of the Fortran compilers offered with the Mod 1 Operating System, Fortran Compiler D and Fortran Compiler H, are designed for high-speed compilation and generate directly executable coding as opposed to the interpretive coding produced by some competitive systems. Execution speed is increased by extensive optimization of the object code produced. The compilation process is shown in Figure 3-7.

Among some of the outstanding advantages offered by these two versions of the Fortran Compiler are the following:

1. Advanced Operating Features
   a. Magnetic Tape Orientation - The ability to use magnetic tapes during program compilation and execution permits extremely fast reading and writing of program and data files. In addition, it eliminates all of the intermediate card decks otherwise required.

b. Subprogram Modularity - A Fortran program can consist of several subprograms. Once a subprogram is compiled, it can be punched into cards or added to a stack (library) tape. Later, the subprogram can be combined with other subprograms to produce an executable program or a series of executable programs on a binary run tape (BRT). The ability to construct such programs from a series of already-compiled subprograms accelerates program completion and permits more efficient organization and planning of the work to be accomplished.

c. Stack (Library) Tape Usage - Permits the storage of a user subprogram library on magnetic tape. During creation of an executable program, specific subprograms can be retrieved from this tape and incorporated into the user's routine.

d. Fortran/Easycoder Integration - Easycoder subprograms are written according to simple rules and assembled by the Easycoder Assembler. A binary deck or tape can be produced as output during the assembly process and subsequently added to a Fortran program deck or to the stack tape.

e. Variable Format Feature - Allows the user to execute the same program with a number of different input/output formats without having to recompile the program. Thus, a program need only be compiled once regardless of the number of data formats it must handle. This feature allows programs to be written and compiled independently of formatting considerations and eliminates the necessity of creating and maintaining a large number of versions of the same logical program for each different data format.

f. Load-and-Go Operation - Under the direction of the Mod 1 Operating System Loader-Monitor, a stack of separate job decks (each representing a problem) are processed as follows:

   (1) Job 1 is compiled (with the integration of subprograms from the stack (library) tape as indicated), and a binary run tape is created. The job is executed and the results listed or written onto tape.

   (2) Control is returned to the monitor, the next job (Job 2) is brought in, and the cycle is automatically repeated.

   (3) This cycle continues until all of the jobs have been processed.

g. Go-Later Operation - A second mode of operation allows the user to compile a series of programs, create a BRT, and dismount it for use at a later date. Another alternative is to create a program from a series of subprograms stored on a stack (library) tape and execute this program during the same run.

h. Chaining - Permits the execution of large programs within a relatively small amount of memory by dividing such programs into independent segments (chains) which can be loaded and executed at different times. Each chain is overlaid in memory by the subsequent chain, with the COMMON area of memory providing the necessary communication linkage between them. Thus, partial results produced by one chain of the program are stored for further processing by subsequent chains.

i. Debugging and Testing - Several facilities are provided for the debugging and testing of programs, including a diagnostic pre-processor (Fortran D), a diagnostic scan during compilation, a symbolic Easycoder listing of the object coding (Fortran D), a memory map listing, and dynamic and terminating dumping procedures.

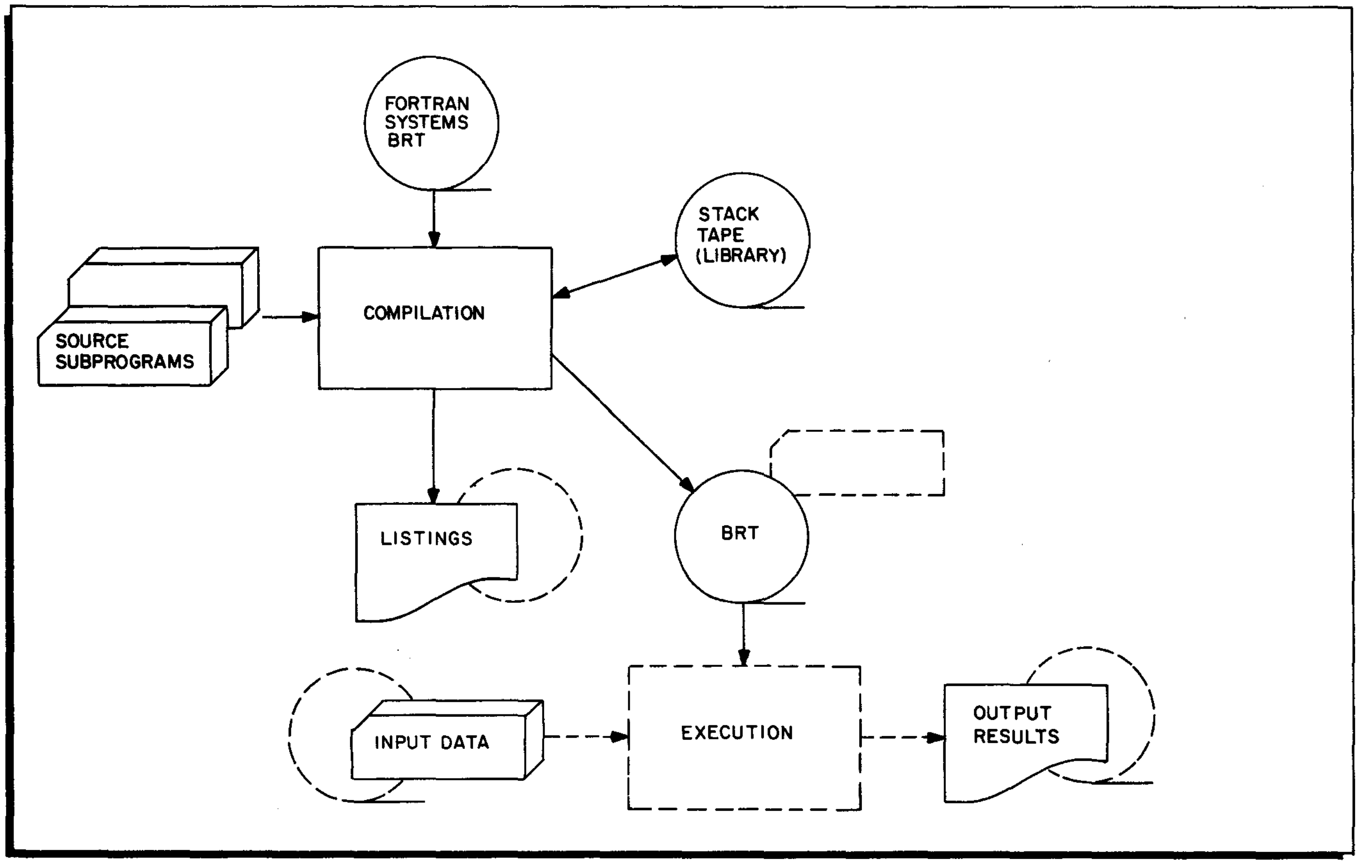


Figure 3-7. The Fortran Compiler System

Both versions of the Fortran Compiler include upward compatibility. Programs written for Fortran Compiler D can be compiled by Fortran Compiler H with little or no modifications. The features offered by each version are summarized in Table 3-4.

Table 3-4. Fortran Compilers D and H: Features

Fortran Compiler D	Has the ability to handle a wide range of specification, input/output, format, conversion, logical, assignment, control, and procedure statements, and features the fullest implementation of Fortran IV available on any machine of comparable size. Generates object programs occupying up to 262K

Table 3-4 (cont). Fortran Compilers D and H: Features

Fortran Compiler D (cont)	characters of memory. It provides high-speed compilation and execution through the utilization of magnetic tapes and object code optimization. It offers programming and operating flexibility by providing subprogram modularity, a stack tape library, Fortran/Easycoder integration, the variable format feature, and the choice of load-and-go or go-later modes.
Fortran Compiler H	Includes all of the features of the A.S.A. Fortran IV plus many additional operational features. Incorporates additional language features such as BEGIN TRACE and END TRACE statements, BEGIN FLOW and END FLOW statements, character strings, T format descriptor, an IMPLICIT statement, list-directed input/output statements, and mixed-mode arithmetic expressions.

Translators

The Honeywell Liberator concept allows users of a number of older competitive systems to enjoy the benefits of the Series 200 without the cost and time of reprogramming. This is done by providing the user with a means of automatically converting from the language of the competitive system to the language of Series 200 Easycoder.

An example of the effectiveness of this concept can be seen in the Easytran Symbolic Translator System. Input programs written in SPS/Autocoder symbolic language for the 1401/1460 series are completely analyzed and then translated statement by statement. During this process, most symbolic statements are replaced on a one-for-one basis with equivalent Easycoder statements due to the similarities in hardware design and program instruction format between the two systems. Those statements which have no Easycoder equivalent (such as most input/output routines) are replaced either with in-line macro coding or with calls to Easytran subroutines which perform the desired functions; those whose functions are not required by the Honeywell hardware are deleted.

The Mod 1 Operating System includes four programs which perform language translation: Easytran Symbolic Translators C and D, Easytran Program Modifier C, and Easytran Source Program Generator D.

EASYTRAN SYMBOLIC TRANSLATORS

The two versions of the Easytran Symbolic Translator translate SPS and/or Autocoder programs into Easycoder symbolic language. The principal output is the translated program; other output includes a parallel listing of the SPS/Autocoder and Easycoder symbolic programs, a cross-reference listing of all tags used in the input program, and an English-language diagnostic listing pointing out any areas where modification might be required.

If the source program contains macro calls to the Autocoder IOCS functions, these are replaced with equivalent calls to the 1/2-Inch Tape Input/Output Package, which are then processed by a special preassembly routine (Library Processor C).

Memory requirements for the translated program are only 10% more than those of the original program, plus additional memory for the Easytran subroutines and for double-buffering of tape files. Despite this additional memory, the running time of the translated program is usually improved due to the faster cycle time and extensive peripheral simultaneity of the Series 200 processors.

The general systems diagram for the translators is shown in Figure 3-8.

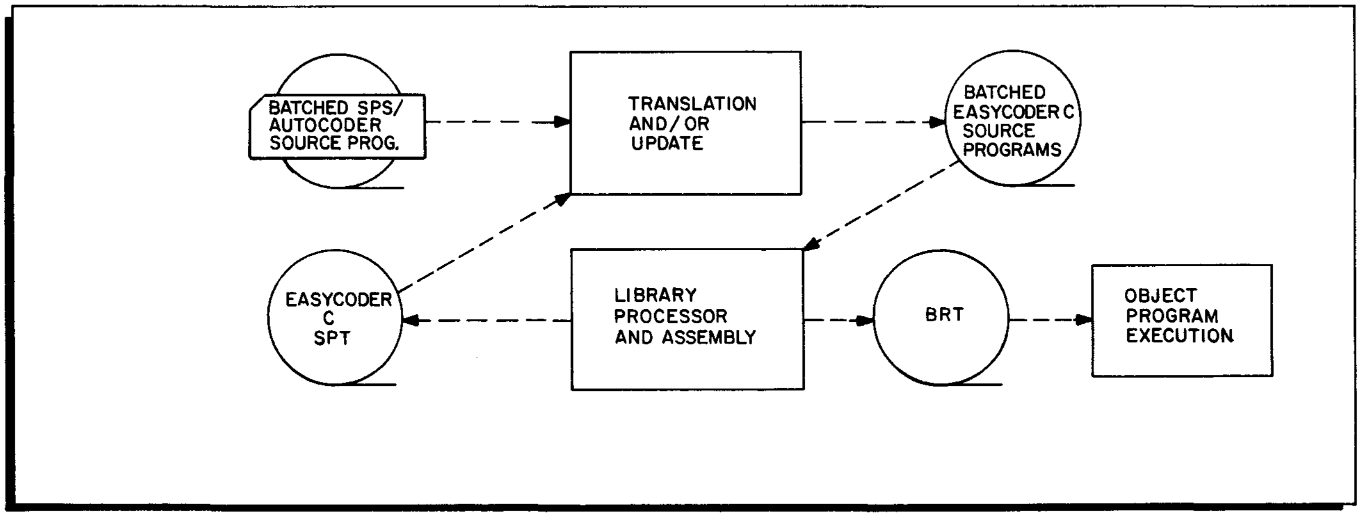

BATCHED SPS/ AUTOCODER SOURCE PROG.
TRANSLATION AND/OR UPDATE
BATCHED EASYCODER C SOURCE PROGRAMS
EASYCODER C SPT
LIBRARY PROCESSOR AND ASSEMBLY
BRT
OBJECT PROGRAM EXECUTION

Figure 3-8. The Easytran Symbolic Translator System

The unique features of each of the versions are summarized in Table 3-5.

Table 3-5. Easytran Symbolic Translators C and D: Features

Easytran Symbolic Translator C	Will accept source programs written in SPS or Autocoder. The Autocoder source programs may contain macro calls. Produces an Easycoder C symbolic source deck, a parallel source-program listing, an English-language diagnostic listing, and a cross-reference tag listing. The symbolic deck must be processed through Library Processor C and Easycoder Assembler C to produce an executable program.

Table 3-5 (cont). Easytran Symbolic Translators C and D: Features

Easytran Symbolic Translator D	Will accept 1401/1460 Autocoder or mixed SPS/Autocoder source programs. Included in the system are the functions of conversion, file update, Library Processor C, and Easycoder Assembler C; therefore, the final output is a directly executable program in Series 200 machine language. Also added are such features as increased compatibility with Autocoder, automatic IOCS call conversion by the substitution of a pretailored version of the 1401 IOCS package, floating address assignments to allow simple address adjustment during subsequent updating, handling of op code overlay (e.g., moving a Branch op code over a NOP op code), and batch processing which allows the conversion of an entire batch of mixed types of source programs in one run.

EASYTRAN PROGRAM MODIFIER

Easytran Program Modifier C is a translator program which modifies Easycoder source-language programs originally created by either 1401 Easytran or Easytran Symbolic Translator B (Basic Programming System) so that they will be acceptable input to Easycoder Assembler C and will be operable with the Mod 1 Operating System. The modified program is written out on a card-image tape which can then be input to Library Processor B or C and/or Easycoder Assembler C.

Easytran Program Modifier C incorporates the following functions:

1. Remaps and revises the Easytran B subroutines to convert them to operate under Floating Tape Loader-Monitor C. Routines may be relocated to allow room for the communication area of the loader-monitor.
2. Pseudo-DA (Define Area) statements in the Easycoder Assembler A generated coding are changed to Easycoder Assembler C source statements. Hand-tailoring techniques not acceptable to Easycoder Assembler C are checked for and flagged.
3. A side-by-side listing containing the originally generated Easycoder Assembler A source language and the corresponding Easycoder Assembler C source language is produced. Diagnostic messages are included within the listing.

The general systems diagram for Easytran Program Modifier C is presented in Figure 3-9.

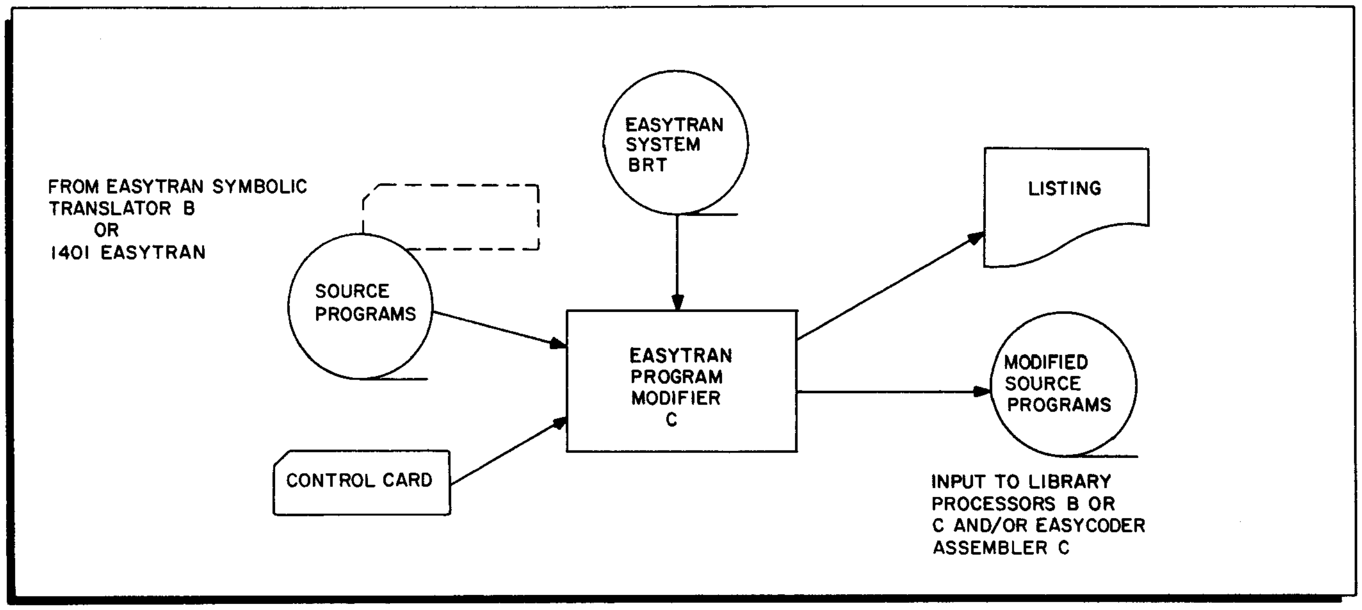

FROM EASYTRAN SYMBOLIC TRANSLATOR B OR 1401 EASYTRAN
SOURCE PROGRAMS
EASYTRAN SYSTEM BRT
EASYTRAN PROGRAM MODIFIER C
CONTROL CARD
LISTING
MODIFIED SOURCE PROGRAMS
INPUT TO LIBRARY PROCESSORS B OR C AND/OR EASYCODER ASSEMBLER C

Figure 3-9. Easytran Program Modifier C

EASYTRAN SOURCE PROGRAM GENERATOR

Easytran Source Program Generator D translates 1401 machine language programs into Autocoder symbolic language and is part of the Easytran D system. Following the translation, Easytran Symbolic Translator D processes this symbolic program output along with other Autocoder/SPS symbolic programs and translates them into directly executable Series 200 machine language. Thus, any intermix of symbolic Autocoder programs and 1401 machine language programs can be handled by the Easytran D system.

Easytran Source Program Generator D incorporates the following features:

1. Accepts as input SPS1 or SPS2 single load card formats, SPS condensed card format, and Autocoder condensed card format decks.
2. Produces a Card Image Tape (CIT) containing the programs in symbolic format acceptable to Easytran Symbolic Translator D.
3. Prints an analysis listing showing the Autocoder symbolic language produced for each input machine-language program along with flags pinpointing possible problem areas.

UTILITY PROGRAMS

The utility programs provided in the Mod 1 Operating System perform two types of functions: data transcription and editing, and mathematical processes.

Data Transcription and Editing

Data transcription and editing functions include those of tape handling, media conversion, report generating, and the sorting and collating of data. They are completely compatible in operation with all of the programs written for, and processed by, any of the language processors; therefore, they can be intermixed on a BRT with user-written programs. By incorporating these functions into his system, the user not only saves the time which would have to be spent in writing his own routines, but also benefits from a reduction of the memory space and exection time required.

TAPE HANDLING

Tape Handling Routine C includes a set of general tape-handling and correction routines for use with 1/2-inch and 3/4-inch magnetic tapes. Parameters for controlling the various functions provided can be entered from cards, paper tape, or the control panel. Functions which operate on a record-by-record basis (e.g., copying) can be directed to terminate either upon the processing of a certain number of records or upon the sensing of a standard label or file identification record.

The functions of Tape Handling Routine C are as follows:

1. Edit - Records (or portions of records) can be edited from a specified tape to an on-line printer, in either alphanumeric or octal mode.
2. Rewind - One or more magnetic tapes can be rewound in a single operation.
3. Copy - A specified number of records can be copied from one tape to another.
4. Correct and Copy - A designated record is copied from one tape to another with specified corrections.
5. Forward - A tape can be positioned forward a specifed number of records.
6. Backspace - A tape can be backspaced a specified number of records.
7. Compare and Print - A specified number of records from each of two tapes are compared record-for-record. Those records which are not identical are printed in either alphanumeric or octal mode.
8. Locate - A tape is searched for a record containing a specified piece of information. Upon locating such a record, the tape is backspaced one record, thus permitting another operation to be performed on the located record.
9. Write dummy header label - A dummy header label, containing a physical tape reel serial number, is written on a new tape.

MEDIA CONVERSION

Media conversion routines transfer data from one medium to another. This might be done for one of several reasons: to increase the speed of subsequent processing of the data by placing it on a faster medium (e.g., converting punched cards to card images on magnetic tape), to permit visual examination of the data (e.g., converting print images on tape to printed output), or to permit physical manipulation of the data (e.g., converting card images on tape to punched cards).

Data Conversion C Routines

Data Conversion C routines are actually macro routines which can be adapted to process any file format and to operate in several environments. Three generalized conversion routines are provided:

1. Card-to-Tape C, which converts a punched-card file to a card-image file on magnetic tape.
2. Tape-to-Printer C, which converts a print-image file on magnetic tape to printed output.
3. Tape-to-Punch C, which converts a card-image file on magnetic tape to a punched-card file.

These three routines can be executed in any of three operating environments:

1. As independent programs,
2. As coroutines operating together under Simultaneous Media Conversion C, or
3. As foreground programs operating simultaneously with some other program under Interrupt Control D.

As independent programs, the routines can be executed apart from any controlling program. Running under the control of the Simultaneous Media Conversion (SCOPE) monitor, two (or three) of the routines can be executed simultaneously. As a foreground program, any one of the routines can operate in the interrupt mode under Interrupt Control D. Under this last method, the conversion routine is assigned all of the processor cycles until it initiates a Peripheral Data Transfer (PDT) instruction. While this instruction is being carried out, all processing cycles are allocated to a background program which consists of a great amount of internal processing, such as a sort or an assembly. At the end of the data transfer operation, Interrupt Control D receives an interrupt signal which directs that control be reassigned to the conversion routine.

All three routines accept as input or generate as output a wide variety of tape file formats including Honeywell or IBM files containing fixed-length or variable-length records, blocked or unblocked. Banner and print control characters may or may not be present.

Among the several processing advantages offered by these conversion routines are the following:

1. Control cards - The specialization of Data Conversion C routines (through Library Processor C) establishes the operating mode and general file type to be handled. However, parameters describing the file structure can be modified at execution time by parameters entered via a control card.
2. Own-coding option - All routines provide exits which allow data to be edited before it is converted to the output medium.

3. Card count - Routines handling punched cards count the number of cards converted.

4. Variable card length - Routines can accept or generate cards or card images of less than 80 columns in length.

5. Item print bypass - Printing of specified card-image items can be bypassed during a Tape-to-Printer C run.

6. IBM print compatibility - Tape-to-Printer C may be specialized to include coding for automatically translating special IBM print characters to Honeywell print characters. IBM channel skipping can also be simulated in Honeywell Type 222 Printers.

7. Sequence checking - Card-to-Tape C and Tape-to-Punch C can perform a sequence check on input items and halt when an item is out of sequence.

Simultaneous Media Conversion C

Simultaneous Media Conversion (SCOPE) C consists of a group of independent coroutines which can operate simultaneously to perform conversion of data from one medium to another. Among the operations performed are:

1. Punched cards to magnetic tape,
2. Magnetic tape to punched cards,
3. Magnetic tape to printed output,
4. Paper tape to magnetic tape, and
5. Magnetic tape to paper tape.

Up to three of these routines can be combined and executed simultaneously. In making up his own version of the package, the user selects the source-language program decks for each of the operations desired plus the source deck for the monitor (main control routine) and assembles them together via Easycoder Assembler C. (NOTE: The paper tape conversion routines must first be specialized by Library Processor C.) During execution, the monitor acts as the controlling routine in determining the allocation of machine cycles to the various input/output conversion routines in the most efficient manner.

REPORT GENERATION

Source-language programs which simulate and expand the report preparation functions of an E.A.M. tabulating machine are generated by a report generating program. Report Generator C generates programs in Easycoder C source language which, when assembled by Easycoder Assembler C, produce reports from card or tape input according to the language parameters entered during generation by the programmer. The output reports can be produced in the form of printed copy, punched cards, and/or magnetic tape.

To direct the generation of the source-language program, the user punches a series of language parameter cards. These parameter cards are compatible with those of the 1401 Report

Program Generator. The parameter cards specify to Report Generator C the types of record formats present in the input file (Input cards), the fields to be processed and any simple calculations to be performed on them (Data cards), any extensive calculations to be made (Calculation cards), the format of the fields and lines constituting the output (Format cards), the operating environment of the generated program (Control card), and information to be used in the simulation of the carriage control paper tape functions on the tabulator (Carriage Control card).

The primary purpose of Report Generator C is to eliminate the need for the user to develop and code a separate program for each of a variety of reports. Instead, he prepares the specification cards and merges them with the Report Generator C master deck. Report Generator C then produces a program which writes the report in accordance with the specifications stated by the programmer.

SORTING AND COLLATING

A number of sorting and collating routines are available in the Mod 1 Operating System to process data stored on magnetic tapes or on drum storage.

Magnetic Tape

All tape sorting routines are based on the Honeywell-developed polyphase merge technique, which uses as few as three tape drives while minimizing the number of passes required over the data. Each sort operation is performed in two stages: the presort and the merge. The presort arranges the input data in ordered strings, the number and length of which depend on the amount of core storage available and on the degree of preordering which exists in the data. The merge phase produces longer ordered strings by combining strings produced during the presort. This continues until there is only a single ordered string on each work tape. At this point, the last pass of the merge combines the remaining strings to form an ordered file on the output tape.

The collate routines, which may be used independently or in conjunction with the sorts, combine two or more ordered tape files to form a single ordered tape file. Each file to be collated, as well as the combined output file, may reside on one or more reels of tape.

Routines are provided for sorting and collating both fixed- and variable-length items, blocked or unblocked. Parameters, entered from either punched cards or paper tape or set up by instructions of some program executed prior to the sort or collate routine, specify the item and record lengths, the number and locations of the key fields within each item, the desired output sequence (ascending or descending), the collating sequence to be used (standard Honeywell or other), and other characteristics. Both routines contain provisions for own-coding, which the user can prepare in Easycoder source language if desired.

Drum Storage

Drum Sort C sorts data stored on a magnetic drum unit by using the key sort technique. During the presort phase, only the key fields of each item (along with the associated drum address of the item) are extracted, examined, and ordered into strings which are written back into the work area reserved on the drum for the file. The subsequent merge phase reads back these strings and combines them in longer ordered strings until a single ordered string results. By incorporating a macro routine into the program which follows Drum Sort C, the user can retrieve the items in order and perform on them whatever processing he desires.

Table 3-6. Sort and Collate Programs: Features

Tape Sort C	Performs tape read backward polyphase merging on fixed-length items, blocked or unblocked. Utilizes from three to six tape drives. Sorts on up to ten key fields. Allows for deletion of unreadable records during the presort or last pass of the merge. Automatically sets restart points throughout the sort to enable restarting immediately or at some later time.
Tape Sort C (3V)	Provides same features as Tape Sort C, except that it has the ability to sort variable-length items, unblocked or blocked a variable number per record. Utilizes core storage of up to 262K characters.
Drum Sort C	Performs sorting on files existing on a drum. Accepts as input files using multirecord, fixed-length format; single-record, fixed-length format; multirecord variable-length format; or single-record, variable-length format. Sorts on up to ten key fields and includes an automatic sequence check and item count check. Own-coding exits are provided in both the sort routine itself and in the subsequent macro routine which retrieves the items from the drum in the desired sequence.
Collate C	Combines two through five ordered tape files of fixed-length, blocked or unblocked items into a single ordered file. Collates on up to ten key fields. Allows the user to accept or delete unreadable records and allows changes to label records. Also allows item-by-item own-coding.
Collate C (3V)	Provides same functions as Collate C, except that it handles variable-length items blocked a variable number per record. Utilizes core storage of up to 262K characters.

Mathematical Processing Functions

The Mod 1 Operating System provides the scientifically oriented Series 200 user with an extensive library of functions which complement the capabilities of the Fortran compilers. A number of packages are written in the Fortran language and are thus easily modified by the user.

The library contains basic math functions, Fortran functions, multiply/divide subroutines, and floating-point arithmetic and conversion routines. All of these functions can be executed with or without the scientific hardware or multiply/divide options. The mantissae of floating-point numbers can vary from two to twenty decimal characters; the integer precision is from three to twelve characters.

A list of mathematical functions is given in Table 3-7.

Table 3-7. Mathematical Processing Functions

Floating-Point Arithmetic/Comparisons C	Comparison of two floating-point numbers for equality, inequality, etc. Does not utilize multiply/divide hardware.
Floating-Point Arithmetic/Comparisons C (N)	Same as above, but utilizes multiply/divide hardware.
Fortran functions:	
Exponential C	Evaluates in floating decimal e^x for an argument of the form: $x = M.10^P$.
Natural Logarithm C	Evaluates in floating decimal $\log_e x$ for an argument of the form: $x = M.10^P$.
Square Root C	Computes in floating decimal the square root of a positive floating decimal number.
Square Root C (V)	
Sine C	Evaluates in floating decimal sine x for an argument of the form: $x = M.10^P$ radians.
Cosine C	Evaluates in floating decimal cosine x for an argument of the form: $x = M.10^P$ radians.
Arc Tangent C	Evaluates in floating decimal $\tan^{-1}x$ for an argument of the form $x = M.10^P$ obtaining a positive angle in the 1st quadrant or a negative angle in the 4th quadrant measured in radians.
Linear Equation Solution C	
Floating-Point/Fixed-Point Conversion C	Performs the conversion between these two modes of numerical expression.
Integer Multiply/Divide:	
Integer Multiply/Divide C (2)	
Integer Multiply/Divide C (2V)	
Integer Multiply/Divide C (3)	
Integer Multiply/Divide C (3V)	
Statistics Package D:	A set of five programs which perform various statistical analyses on numerical data.
Chi-Square D	Evaluates Chi Square.
Least Squares Curve Fitting D	Fits a polynomial of degree n to a set of m observations by the method of least squares.

Table 3-7 (cont). Mathematical Processing Functions

Mean, Variance and Correlation D	Computes the mean, variance, covariance, standard deviation, and correlation coefficient of the variables which are stored in two groups on the tape: the X-group and the Y-group.
Step-Wise Multiple Regression Analysis D	Finds the best fit of an equation of the following form: $y = B_0 + B_1x_1 + B_2x_2 + \ldots\ldots + B_{n-1}x_{n-1}$ where y is the dependent variable and x_1, x_2, ... are independent variables.
Random Number Generator D	Generates a set of random numbers.
Differential Equations D	Solves differential equations using the Clippinger-Dimsdale method.

PROGRAM EDITING AND MAINTENANCE

The functions of storing, modifying, and maintaining source-language and machine-language programs come under the heading of program editing and maintenance. These functions enable programs to be selected and ordered to create master run tapes which contain only those systems programs and processing programs required in the order best suited to the individual jobs.

Symbolic Programs

The editing and maintenance of symbolic programs include program updating and program selection. These functions are performed by routines which are part of the assembly and compilation systems themselves. As mentioned previously, the Easycoder Assemblers C and D write all assembled programs onto a symbolic program tape (SPT) in both their source- and machine-language formats. Operating in the update mode, Easycoder Assemblers C and D can correct (and reassemble) individual programs on, add new programs to, and delete unwanted programs from, this SPT. Likewise, the Source Program and Library Update Routine of COBOL Compilers D and H enables the user to add, delete, and replace programs and library units to create an updated source program and library tape.

The SPT Merge C program increases the facility of handling programs stored on SPT's by performing the selection and rearrangement of symbolic programs from as many as four different input SPT's and writing them on a new SPT. Thus, it is possible to consolidate programs stored on several SPT's onto one master SPT in any desired order. An important aspect of this process is that the symbolic programs (source- and machine-language formats) can be selected, copied, and rearranged without reassembly onto the new tape.

Machine-Language Programs

Just as in the case of symbolic programs, the editing and maintenance processes for executable machine-language programs also include program updating and selection. Update and Select C and D, which performs these processes, accepts all binary run tapes created within the Mod 1 Operating System. Both versions can update a master binary run tape by either correcting, replacing, or deleting programs already on it or by adding new programs to it. They can also select programs, in any order, from the master BRT and produce a selected BRT containing these programs.

A second program, BRT Punch C, punches object programs from a BRT onto cards for loading and executing under Card Loader-Monitor B.

A third program, Drum Program Store C, converts one or more BRT programs for storage on a Type 270 Random Access Drum File. A program file as stored on a drum consists of Drum Bootstrap-Loader C (optional), Drum Monitor C (optional), the user's object programs (to be loaded into memory by either Drum Bootstrap-Loader C or Drum Monitor C), and a program directory of the program file. Drum Program Store C accepts as input any object programs in BRT format produced within the Mod 1 Operating System and converts this input into a format acceptable to the drum loader-monitor routines.

Table 3-8. Program Maintenance and Editing Functions

SPT Merge C	Selects and rearranges symbolic programs from as many as four input SPT's onto a new SPT.
Update and Select C	Enables the user to maintain a master BRT by allowing him to correct programs on it with octal patching, delete programs from it, replace programs on it, and add new programs to it. It can also produce a selected BRT containing specified programs from the master BRT as specified by the programmer. Manipulates object coding by segment units only. Prints a listing of the director cards, a directory of the new BRT, and a directory of the selected BRT.
Update and Select D	Same as the version above, except that it can manipulate program units as well as segment units.
BRT Punch C	Converts binary run tapes to BRT-format punched cards (binary run decks) which are acceptable to Card Loader-Monitor B. Programs from up to six input BRT's can be selected for punching in any order.
Drum Program Store C	Converts one or more object programs from BRT format for storage on a Type 270 Random Access Drum File. A program file is created on the drum and consists of Drum Bootstrap Loader C (optional), Drum Monitor C (optional), the user's object programs modified for drum storage and loading, and a directory of the program file.

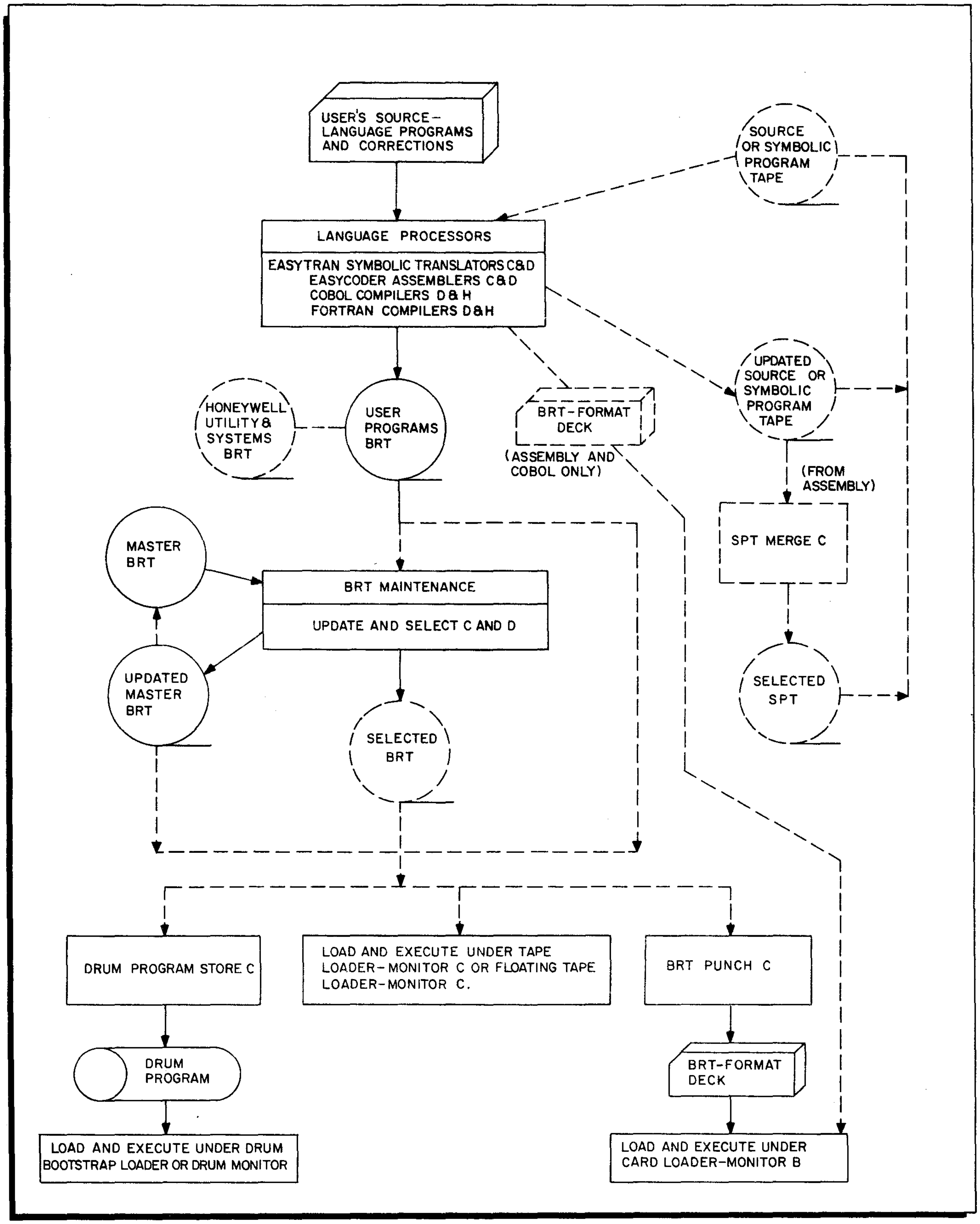

USER'S SOURCE-LANGUAGE PROGRAMS AND CORRECTIONS
SOURCE OR SYMBOLIC PROGRAM TAPE
LANGUAGE PROCESSORS
EASYTRAN SYMBOLIC TRANSLATORS C&D
EASYCODER ASSEMBLERS C&D
COBOL COMPILERS D&H
FORTRAN COMPILERS D&H
UPDATED SOURCE OR SYMBOLIC PROGRAM TAPE
HONEYWELL UTILITY & SYSTEMS BRT
USER PROGRAMS BRT
BRT-FORMAT DECK
(ASSEMBLY AND COBOL ONLY)
(FROM ASSEMBLY)
SPT MERGE C
MASTER BRT
BRT MAINTENANCE
UPDATE AND SELECT C AND D
UPDATED MASTER BRT
SELECTED BRT
SELECTED SPT
DRUM PROGRAM STORE C
LOAD AND EXECUTE UNDER TAPE LOADER-MONITOR C OR FLOATING TAPE LOADER-MONITOR C.
BRT PUNCH C
DRUM PROGRAM
BRT-FORMAT DECK
LOAD AND EXECUTE UNDER DRUM BOOTSTRAP LOADER OR DRUM MONITOR
LOAD AND EXECUTE UNDER CARD LOADER-MONITOR B

Figure 3-10. Mod 1 Operating System: Program Preparation and Maintenance

SECTION IV
PROGRAM EXECUTION AND CONTROL

The program execution and control capabilities of the Mod 1 Operating System are divided into three categories:

1. Operation control,
2. Input/output control, and
3. Program test.

OPERATION CONTROL

Operation control encompasses all of the functions involved in the searching for, and the loading and monitoring of, object programs. The Mod 1 Operating System extends these functions to all programs, whether stored on punched cards, magnetic tape, or drum. Also included is interrupt software which enables the user to take advantage of the interrupt features and peripheral simultaneity of the Series 200 computers. Lastly, a utility program reads comments cards containing instructions to the operator, report headings, etc., and displays them on the console typewriter or printer.

Loading and Monitoring

The loading and monitoring of object programs are accomplished with essentially the same efficiency regardless of the program storage medium. Most of the same functional characteristics and convenient operational features are also retained.

LOADING FROM TAPE

Both of the tape loader-monitor routines, Tape Loader-Monitor C and Floating Tape Loader-Monitor C, search for and load programs stored on any binary run tape (BRT) produced within the Mod 1 Operating System environment.

Object programs, as assembled or compiled and stored on a BRT, are composed of one or more loading units or segments. Any loading unit can be searched for, loaded, and executed independently. A call to search for and load a given program segment can originate from several sources:

1. Current program - The programmer may include programmed instructions which set up the required search parameters in the loader communications area and then branch to the appropriate loader routine to initiate the searching for and loading of the next segment to be executed.

2. Operator - The operator can enter the parameters via the control panel or console typewriter and manually branch to the loader routine.
3. Console Call card - Search parameters can be entered through the card reader via a Console Call card. The reading of the card and the search and load operation can be initialized either manually by the operator or automatically by programmed instructions.

Basically, the search parameters specify the search mode to be used and the program name, segment name, visibilities, etc. The search modes include the following:

1. Search by visibility and relative position - Searching in the specified direction (forward or backward) on the BRT, load the nth loading unit having the specified visibility.
2. Search by program name and segment name - Searching in the specified direction (forward or backward) on the BRT, load the loading unit having the specified program name and segment name regardless of visibility.
3. Search by segment name within the current program - Searching in the specified direction, load the loading unit having the specified segment name within the current program.
4. Search by program name, segment name, and visibility - Searching in the specified direction, load the loading unit having the specified program name, segment name, and visibility. This mode allows for the presence of several versions of a program on the same BRT by assigning a different visibility to each of the versions.
5. Search by segment name and visibility within the current program - Searching in the specified direction, load the loading unit having the specified segment name and visibility within the current program. This mode allows for the presence of several versions of a routine within a program.

Other parameters which may be specified are load parameters (relocation augment, etc.) and start parameters (branch to normal start location, special start location, etc., in loaded unit, set trapping mode, etc.).

Two versions of the tape loader-monitor are provided: Tape Loader-Monitor C and Floating Tape Loader-Monitor C. Tape Loader-Monitor C is not relocatable and occupies 1,276 locations of main memory (locations 64 through 1,339). Floating Tape Loader-Monitor C consists of two segments. Segment 1 occupies memory locations 2,150 through 4,096 and is required only for the purpose of loading Segment 2. Segment 2 is the actual tape loader-monitor routine and can be reallocated or "floated" into any memory bank above memory bank ∅. Once Segment 2, which occupies a minimum of 1,400 memory locations, is loaded, Segment 1 can be overlaid. Thus, user programs which have been assembled to reside in memory below location 1339 need not be modified and reassembled as would be required with the use of Tape Loader-Monitor C. Floating Tape Loader-Monitor C, which provides all of the functions of Tape Loader-Monitor C plus several additional features, is a macro program which must be specialized and processed by Library Processor C and then assembled. By filling in the required parameters in the macro instruction, the user can specialize the loader-monitor to load programs in any portion of memory, load binary run decks, provide own-coding exits, and utilize a console typewriter to display messages and to receive operator responses. As explained later, Floating Tape Loader-Monitor C must be present in memory in order to use Interrupt Control D.

A further discussion of the loader-monitor routines can be found in Section V.

LOADING FROM CARDS

Object programs punched on binary run decks (such as those produced by Easycoder Assemblers C and D or BRT Punch C) can be searched for, loaded, and initiated by Card Loader-Monitor B. The functions and operational characteristics of this routine are compatible with those of the tape loader-monitors in many ways:

1. The loader-monitor communication areas of both are identical.
2. A call can originate from the same three sources: the current program, the operator, or a Console Call card.
3. The same search modes are provided. Exceptions: Card Loader-Monitor B cannot, of course, search in a backward direction and cannot load by visibilities.
4. The same loading and starting parameter options are applicable.

Card Loader-Monitor B requires 936 memory locations (64 through 999) and has a fixed allocation. However, if this allocation is not desirable, Floating Tape Loader-Monitor C can be specialized to load from cards.

LOADING FROM DRUM

The loading and monitoring of object programs which are part of a drum program file created by Drum Program Store C are performed by two routines: Drum Bootstrap-Loader C and Drum Monitor C.

Drum Bootstrap-Loader C is the simpler of the two and is designed primarily to bring Drum Monitor C into main memory from the drum program file; however, it is also capable of searching for and loading any object program on the drum. Drum Bootstrap-Loader C is the only program in the drum program file which is stored in condensed card images (to facilitate bootstrapping) instead of in modified BRT format. The entire routine occupies approximately 750 characters and can exist in one of two forms:

1. As a bootstrap-loader routine which can be manually bootstrapped from the drum itself into a fixed memory area beginning at location 1,340, or
2. As a loader routine which can be loaded into memory by Drum Monitor C, or by some other loader-monitor routine (Card Loader-Monitor B, Tape Loader-Monitor C, etc.) if stored on some other device. In this

case, the routine can be relocated in memory by simply reassembling it. This allows lower memory to be used by the object programs.

The bootstrap-loader routine contains a 33-character parameter area into which the programmer must enter the required values before executing the loader. A program can be searched for and loaded according to program name, segment name, and visibility. The area is initially set to search for and load Drum Monitor C and must be modified either manually (when Drum Bootstrap-Loader C is bootstrapped) or by programmed instructions (when Drum Bootstrap-Loader C is loaded by some other method) if some other object program is desired. The loading of the program can then be initiated by branching to the loader manually through a console fixed start or automatically through programmed instruction.

Drum Monitor C is designed to be both functionally and operationally compatible with the card and tape loader-monitor routines in the following areas:

1. The loader-monitor communication area is identical.
2. A call to search for and load an object program can originate from the same three sources: the current program, the operator, or a Console Call card.
3. The search modes are the same five modes included in the tape loader-monitor routines.
4. The same loading and starting options are available.

Drum Monitor C occupies locations 64 through 1,339 in main memory and can be loaded from a drum storage unit by Drum Bootstrap-Loader C, or from cards or tape by any other loader-monitor routine which does not occupy locations 64 through 1,200.

Table 4-1. Operation Control: Loading and Monitoring Functions

Tape Loader-Monitor C	Searches for and loads object programs stored on binary run tapes (BRT's). A call to search and load a program segment can originate from the operator, the current program, or a Console Call card. Five types of searches, operating in either a forward or backward direction, are provided: (1) visibility and relative position, (2) program name and segment name, (3) segment name within current program, (4) program name, segment name, and visibility, (5) segment name and visibility within current program. Loading and starting options, as well as user own-code exits are also provided. Occupies 1,276 locations (64 through 1,339) of core storage and is not relocatable.
Floating Tape Loader-Monitor C	Provides all of the functions of Tape Loader-Monitor C plus the capacity to be relocated or "floated" to any memory bank above bank Ø. Offered in the form of a macro routine, the user can specialize it to load programs in any portion

Table 4-1 (cont). Operation Control: Loading and Monitoring Functions

Floating Tape Loader-Monitor C (cont)	of memory, load BRT-format program decks, provide own-coding exits, or utilize a console typewriter. Occupies a minimum of 1, 400 memory locations.
Card Loader-Monitor B	Searches for and loads program segments stored in BRT-format on cards. Implements all of the capabilities of Tape Loader-Monitor C with the exceptions that searching is in a forward direction only and visibilities are not taken into account. Occupies 936 memory locations (64 through 999).
Drum Bootstrap-Loader C	Designed primarily to load Drum Monitor C, this basic routine can be utilized to load any object program which resides on a drum unit as part of a program file. As a bootstrap-loader routine, it can be manually bootstrapped into main memory beginning at location 1, 340; as a loader routine it can be loaded from drum, cards, or tape by the appropriate loader-monitor and can be relocated by modifying the origin and reassembling. Program searching and loading by program name, segment name, and visibility can be initiated by the operator or by programmed instructions. Occupies approximately 750 memory locations.
Drum Monitor C	Extends all of the functions and operational characteristics of Tape Loader-Monitor C to object programs stored on a drum unit. Occupies locations 64 through 1, 339 in main memory and can be loaded from a drum by Drum Bootstrap-Loader C or from any other storage medium by the appropriate loader-monitor routine.

Interrupt Capabilities

Series 200 computer systems provide an interrupt feature which, along with their inherent peripheral simultaneity, allows two programs to be executed together in much less time than if they were run serially. One of the programs, termed the foreground program, is normally a terminal peripheral-type routine such as card-to-tape, communication, tape-to-printer, etc. The other program, called the background program, usually contains a high percentage of internal processing, such as a sort, collate, assembly, etc. At the start of execution both programs are loaded into different areas of memory and control is given to the foreground program. The foreground program is then executed until a Peripheral Data Transfer instruction (PDT) is encountered. While the data transfer is actually taking place, an interrupt control routine transfers control to the background program, a portion of which is executed during the machine cycles allotted to main memory during the data transfer. At the end of the data transfer, the peripheral control generates an interrupt signal which causes the interrupt routine to return control to the foreground program. The foreground program continues execution once again until another PDT instruction is encountered, at which time the control is given to the background program which continues where it left off. This back-and-forth operation continues until both programs reach their termination.

INTERRUPT CONTROL D

Interrupt Control D is a Honeywell-supplied routine designed to aid the user in taking advantage of these interrupt capabilities. It runs in conjunction with Floating Tape Loader-Monitor C, which must be resident in memory immediately following Interrupt Control D.

Interrupt Control D is a generalized macro program which must be specialized by Library Processor C or D before being assembled by Easycoder Assembler C or D. Six prespecialized versions are provided to handle three- or four-character mode operation, the presence or absence of an external INTERRUPT button, etc. These versions require 500 to 625 locations starting at location 200 plus 750 to 1, 150 locations immediately below the Floating Tape Loader-Monitor C routine. Interrupt Control D permits the independent sequencing of both foreground and background programs as indicated by the user. Background programs are searched for, loaded, and started normally under control of Floating Tape Loader-Monitor C. Foreground programs, however, cannot communicate directly with the loader-monitor and must instead terminate with a macro call to Interrupt Control D. If the user has previously set an indicator in a specified memory location, Interrupt Control D loads the next foreground segment and continues execution of the new segment and of the background program; if the user has not set the internal indicator, Interrupt Control D allows the background program to continue processing to completion.

Foreground Programs

Currently, Honeywell offers prespecialized foreground programs to perform tape-to-printer, tape-to-card, and card-to-tape data conversions. These three routines are known collectively as Data Conversion C and are described on page 3-20. If the user writes his own foreground program, he must follow certain programming considerations and must include a return macro call immediately after each PDT instruction. This causes a return of control to Interrupt Control D which, in turn, enters the background program. At the end of the foreground program, the user must include an exit macro call to return control to Interrupt Control D, which will either cause the next foreground segment to be loaded or continue the processing of the background program to completion. Figure 4-1 illustrates these functions.

SIMULTANEOUS SORT AND PRINT

To aid the user in combining two commonly used functions, Honeywell has modified the Tape Sort C program to allow printing while sorting. The print program to which Tape Sort C can be linked is called MONTOR and is a two-segment program which can be used alone or with the sort process. The first segment is the linkage between the sort routine and the print routine and performs the function of an interrupt control routine; the second segment is a utility routine for printing unblocked standard print-image tapes on a 132-position printer. Any utility routine can be substituted for the second segment provided that the first-segment linkage routine is retained.

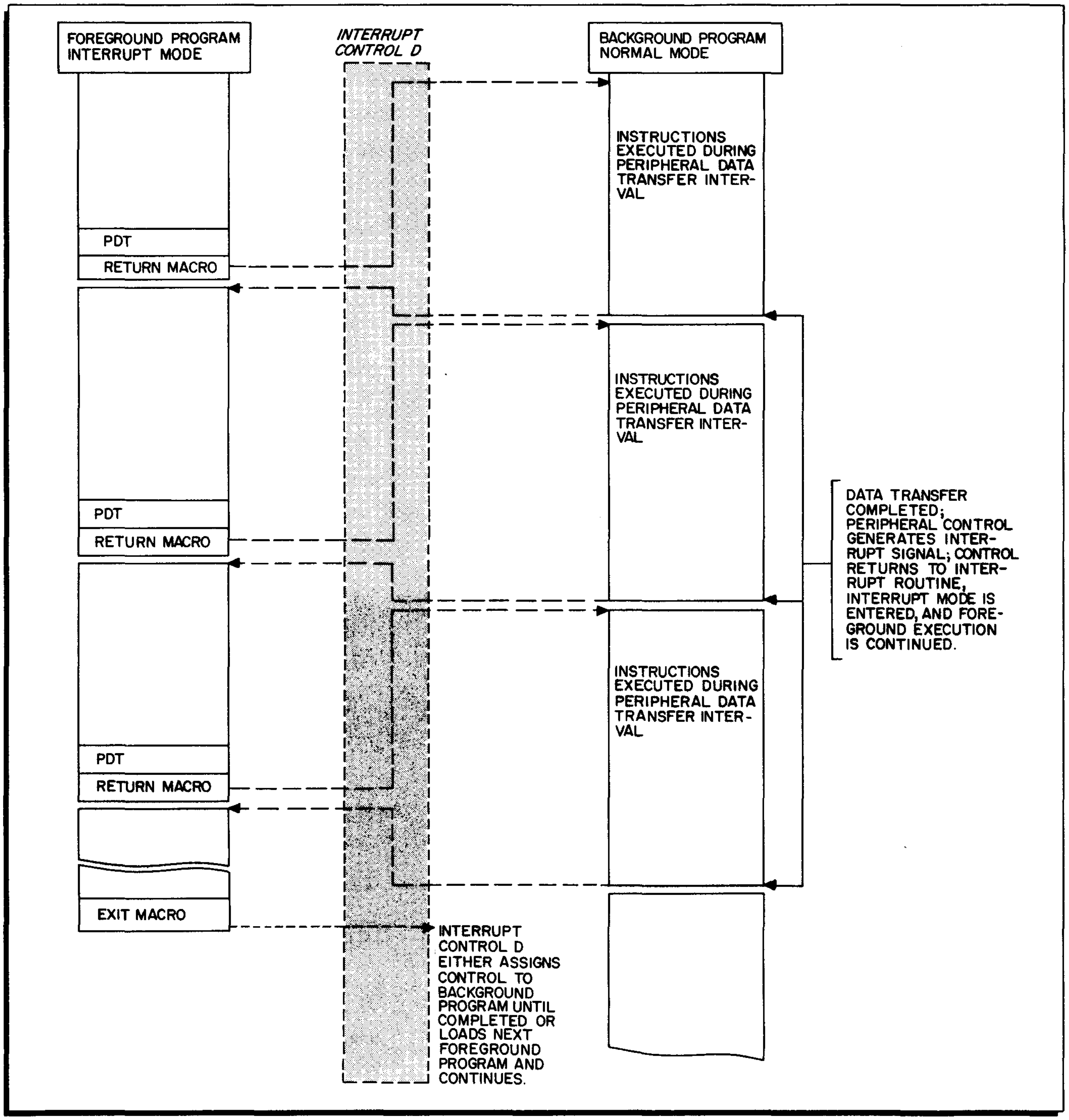

FOREGROUND PROGRAM INTERRUPT MODE
INTERRUPT CONTROL D
BACKGROUND PROGRAM NORMAL MODE
INSTRUCTIONS EXECUTED DURING PERIPHERAL DATA TRANSFER INTERVAL
PDT
RETURN MACRO
INSTRUCTIONS EXECUTED DURING PERIPHERAL DATA TRANSFER INTERVAL
PDT
RETURN MACRO
INSTRUCTIONS EXECUTED DURING PERIPHERAL DATA TRANSFER INTERVAL
DATA TRANSFER COMPLETED; PERIPHERAL CONTROL GENERATES INTERRUPT SIGNAL; CONTROL RETURNS TO INTERRUPT ROUTINE, INTERRUPT MODE IS ENTERED, AND FOREGROUND EXECUTION IS CONTINUED.
PDT
RETURN MACRO
EXIT MACRO
INTERRUPT CONTROL D EITHER ASSIGNS CONTROL TO BACKGROUND PROGRAM UNTIL COMPLETED OR LOADS NEXT FOREGROUND PROGRAM AND CONTINUES.

Figure 4-1. Multiprogramming with Interrupt Control D

List Comments C

List Comments C is a short routine which reads cards containing comments such as operator instructions, report headings, etc., and displays them on either the printer or the console typewriter as directed. Spacing and carriage control functions are included. List Comments C is loaded from a BRT under the control of Tape Loader-Monitor C.

INPUT/OUTPUT CONTROL

Honeywell provides a comprehensive set of input/output control functions in the form of macro routines which can be specialized and incorporated into the user's programs. These functions handle all of the standard input/output devices such as magnetic tape units, card readers and punches, printers, drum units, and consoles; thus, the need for the writing of detailed and exhaustive input/output coding by the user is eliminated. Moreover, the standardization of file handling, label creation, read/write error routines, etc., creates a uniformity in both data format and operating procedures.

Magnetic Tape Input/Output Control

1/2-Inch Tape I/O C is a group of macro routines which handle data files of either fixed- or variable-length records stored on 1/2-inch magnetic tape. These routines are capable of handling Honeywell-created tapes, or if the proper hardware compatibility options are installed, IBM-created tapes. The user incorporates these routines into his Easycoder source-language program by means of file-definition entries and macro calls. The source program is then specialized by Library Processor C or D before being assembled.

In writing his program, the user must first supply a definition (Define Communications Area - DCA) for each tape file to be processed. He includes information such as the file type (input or output), the parity (odd or even), the read/write channel to be used, the blocking factor, item size, locations of buffers reserved for the file, etc. Then, in writing the logic for the program, he utilizes the appropriate macro calls (@OPEN, @GET, @PUT, @CLOSE, @FEOR) to open or close the files, read or write records, and force an end of reel. The checking and creation of standard labels, the blocking and unblocking of records, and the detection and correction of read/write errors are all performed automatically by the macro routines. User's own-coding exits are provided for additional processing of header and trailer labels and for initiating the programmer's own end-of-file procedure.

The 1/2-Inch Tape I/O C routines are written and assembled in three- or four-character mode and, depending upon the processes called for, occupy between 2,200 and 3,500 core memory locations. In addition, each file processed requires a file table of up to 73 character locations and the necessary buffer areas.

Magnetic Tape and Terminal Input/Output Control

1/2-Inch Tape and Terminal I/O C is a more comprehensive input/output control package consisting of a series of routines which perform the standard input/output procedures for magnetic tape, punched card, and printer operations. Read/write channel tests are also included to take advantage of the read/write/compute simultaneity of the Series 200 systems. Honeywell-format and IBM-format (if required hardware options are present) 1/2-inch magnetic tape files are acceptable and the routines are functionally compatible with those of the IBM 1401 Input/Output Control System (IOCS).

The programmer must supply to the package three types of descriptive entries:

1. A Descriptive IOCS (DIOCS) entry which describes, in general, all the files to be processed and the system configuration used,
2. A Define the File (DTF) entry for each of the files which describes, in detail, the type, format, etc., of a specific file, and
3. Carriage control entries to specify punches in the printer carriage-control tape.

In addition to a number of extensions to the macro calls provided in 1/2-Inch Tape I/O C, five additional macro calls have been added:

1. RDLIN - Allows the label information specified in the DTF entry for the file to be changed from run to run; e.g., the creation date constant to be compared.
2. SPACE - Allows the programmer to control the spacing of printer forms.
3. SKIP - Allows the programmer to control the skipping of printer forms.
4. RELSE - Permits the programmer to skip over the remaining items of a blocked record and to continue processing with the first item of the next record.
5. DCLOS - If, in one or more DTF entries, the programmer has specified that all records containing parity errors are to be written on an error tape, this macro call causes the deactivation of that tape. Additional parameters can indicate that a tape mark is to be written after the last record and that the reel is to be rewound and unloaded.

As in 1/2-Inch Tape I/O C, user's own-coding exits are provided for the processing of nonstandard labels or the additional processing of standard labels, the programmer's own end-of-reel or end-of-file routines, etc. In addition, the package can optionally check for wrong-length tape records.

The 1/2-Inch Tape and Terminal I/O C routines can be assembled in either three- or four-character mode and have the following minimum memory requirements:

1. DIOCS table - 800 locations
2. Each tape file - 650 to 1,200 locations
3. Each card file - 250 to 400 locations
4. Each printer file - 1,900 locations

Memory space for buffers is not included in the above figures.

Drum Input/Output Control

Drum I/O C is a set of macro routines which allow the user to handle files stored on drums with the same ease and simplicity as those stored on cards or tape. These routines read and write data sectors, block and unblock items, and execute standard error procedures. Items may be of fixed or variable length. Files can be read only, written only, or read and written. Processing can be on a serial basis (item by item), a random basis (the file is read until a specific item is found), or a designated basis (a specific segment is read or written).

The programmer defines the files using a series of parameters which state the file format, item length, the limits of the file, and the locations of several own-coding routines for end of file, illegal address, file not found, and read error conditions. He directs the processing through the same type of macro calls (OPEN, GET, PUT, CLOSE, etc.) as are used with the tape and terminal input/output control packages.

Console Input/Output Control

Console I/O C consists of a group of macro routines which control data transfer between main memory and a Type 220-1, 220-2, or 220-3 console typewriter. Data messages can be up to 80 characters in length and can be in either alphanumeric (six-bit), octal (three-bit), or decimal (four-bit) format.

To incorporate these routines into his program, the user must include a @CONSL macro call which specifies the data format(s) selected, the read/write channel to be used, etc., and causes the inclusion of these common routines into the program. Within the logic of the program, he includes a @TYPE macro call at each point where he wishes to display data on, or accept data from, the console. If he wishes to do both, he can use one @TYPE macro statement to specify both the location of the message to be typed out (to request the typein) and the location into which the response is to be placed.

The Console I/O C routines can be assembled in three- or four-character mode and have the following minimum memory requirements:

Data format(s) selected	Minimum memory required
Alphanumeric	540 character locations
Alphanumeric and decimal	730 character locations
Alphanumeric and octal	920 character locations plus (4 x maximum message length)
Alphanumeric, decimal, and octal	1, 110 character locations plus (4 x maximum message length)

Communications Input/Output Control

Communications I/O C is an input/output package which aids the user in the programming of communication network applications such as message switching, data collection, information retrieval, inquiry handling, and management information systems. Its primary functions are the control and translation of data to and from such communication units as telephones, teletypewriters, data stations, and other remote terminal equipment. These functions are selected and controlled by macro calls (OPEN, GET, PUT, etc.) inserted into a source program by the user and specialized by Library Processors C or D. Such functions as initialization, interrupt processing, error detection and handling, monitoring of lines, and updating of line status information are all performed automatically.

Communications I/O C is a general-purpose communications package and can be easily adapted to the user's present systems requirements and readily modified in the future to handle any changes or additions to the system. The programmer need not have a detailed knowledge of either the communication network itself or the systems considerations for real-time data flow.

Table 4-2. Input/Output Control Functions

1/2-Inch Tape I/O C	A macro routine package which handles data files of fixed- or variable-length items stored on 1/2-inch magnetic tape. Macro routines are provided to open and close files, read and write records, and force an end-of-reel condition. The checking and creation of labels, blocking and unblocking of items, and the detection and correction of read/write errors are handled automatically.
1/2-Inch Tape and Terminal I/O C	A series of macro routines which handle all standard input/output procedures for 1/2-inch magnetic tape files, punched card files, and printer files. RWC-availability tests are made to take advantage of the inherent simultaneity of the system. Functions provided are compatible with those of the IBM 1401 IOCS routines.
Drum I/O C	A series of macro routines which allow the user to handle files stored on drum units with the same ease and in the same manner as files stored on magnetic tape or punched cards. Items can be of fixed or variable length and processing can be on a serial, random, or designated segment basis.
Console I/O C	A series of macro routines which control data transfer between main memory and a Type 220-1, 220-2, or 220-3 Console. Data messages can be up to 80 characters in length and can be in any of three modes: alphanumeric, octal, or decimal.
Communications I/O C	A series of macro routines which aid the user in the programming of communication network applications by controlling and translating data to and from communication units such as telephones, teletypewriters, data stations, and other remote terminal equipment.

PROGRAM TEST FACILITIES

An important part of the program execution and control functions in the Mod 1 Operating System is the program test facilities.

Automatic Program Checkout

Program Test System C is an automatic, open-ended checkout system which operates under Tape Loader-Monitor C and provides automatic run-to-run sequencing, test data generation, program patching, program checkout diagnostics, memory dumping, and tape dumping. The open-ended design allows the user to write his own program test utility routines and incorporate them within the test system. As provided by Honeywell, Program Test System C is composed of nine utility programs stored on a BRT. These are Initializer C, List Comments C, Test Data Generator C, Memory Dump Control C, Memory Dump C, Patch C, Tape Dump C, Emergency Dump C, and End C.

INITIALIZER C

Initializer C prepares the Program Test System for automatic reading of Console Call cards from the test director deck and the loading of the other utility programs from the BRT. It sets the computer to the nontrapping mode, rewinds the BRT, and suppresses the Tape Loader-Monitor C console call halt to enable nonstop sequencing and loading of the programs.

LIST COMMENTS C

List Comments C reads punched cards containing operator instructions, report heading lines, etc., and prints or types this information on the printer or the console typewriter. This utility program has been previously described under Operation Control on page 4-8.

TEST DATA GENERATOR C

Test Data Generator C creates test data on 1/2-inch magnetic tape from punched cards and enables the programmer to test his programs against a wide range of possible input variables. Tape files can be created with bannered or bannerless records of blocked or unblocked, fixed- or variable-length items. Header, trailer, and tape mark records are created as directed by the programmer.

MEMORY DUMP CONTROL C

Memory Dump Control C facilitates the printing out of core storage contents by loading and initiating Memory Dump C to edit and print these contents. Memory dumping can be called for by either or both of two methods: programmed instruction, which allows for the taking of memory dumps by use of symbolic coding within the program; item-mark trapping, which permits the use of an item mark over the operation code of an instruction to trigger memory dumping.

MEMORY DUMP C

Memory Dump C is used by both Memory Dump Control C and Emergency Dump C to edit and print the contents (data and punctuation) of core storage. It must be loaded by one of these two programs and cannot be used independently within the Program Test System. All printouts are given in both alphanumeric and octal mode with any punctuation displayed beneath the associated character positions.

PATCH C

Patch C enables the programmer to make octal patches to his program in main memory without affecting the program as stored on the BRT. Thus, on-the-spot modifications can be made to a program during testing to try out different variations of some routine, check out proposed corrections to the program before making a permanent change, etc.

TAPE DUMP C

Tape Dump C positions 1/2-inch magnetic tape files and edits and prints their contents as directed by the user. Functions include rewinding, positioning (forward or backward), and editing and printing. Editing can be in either alphanumeric or octal format. Any number of consecutive functions can be performed with a single loading of the routine.

EMERGENCY MEMORY DUMP C

Emergency Memory Dump C can be used to take a printout of core storage if unexpected difficulty is encountered during execution. Memory Dump C is loaded and utilized to edit and print the contents. In the three-character addressing mode, Emergency Memory Dump C requires 125 core locations and Memory Dump C requires an additional 625 core locations. The operator can call in the Emergency Memory Dump C program by entering the three standard call parameters (program name, segment name, and tape drive number) through either the control panel (or console) or the card reader.

Use of the Program Test C Utility Programs

Under the Program Test System C, these nine utility programs, along with the user's programs to be tested, are searched for, loaded, and executed automatically under the direction of the test director deck. This deck can contain Console Call cards, data cards for test data generation, octal corrections, etc. A whole series of runs can be checked out with little or no operator intervention by placing Console Call cards for both the user's object programs and the Program Test System C utility programs in the proper order, along with other required cards, in the test director deck.

With the exception of the separate memory dump and memory dump control programs, which must be run together as already noted, each utility program making up Program Test System C can be run separately as a regular BRT program without executing Initializer C.

SECTION V

PROGRAM SEARCHING AND LOADING

Basically, all program searching and loading performed in the Mod 1 Operating System is controlled by the contents of the loader communication area. These contents can be changed by either the programmer or the operator in any one or more of three different ways:

1. Manually through the control panel or console (all fields).
2. By programmed instructions (all fields).
3. By a Console Call card (program name, segment name, BRT tape drive number).

The layout of the entire loader communication area is presented in Table A-1, page A-5 . For the convenience of the reader, that portion of the area concerned with program searching and loading is repeated in Table 5-1.

Table 5-1. Program Searching and Loading Parameters

Parameter Name	Locations		Values	Methods of Altering			Automatically Reset		
	Decimal	Octal		Manually (control panel)	Programmed Instructions	Console Call card	By Console Call	After Loading	Special Call
Search Mode	111	157	20 prog. and seg. name 01 vis. and rel. pos. 00 seg. (within curr. program) 60 program, segment, visibility 40 seg. and vis. (within curr. prog.)	x	x		20		01
Search Direct.	106	152	22 - forward 23 - backward	x	x		22	22	
Program Name	68-73	104-111		x	x	x			
Segment Name	74-75	112-113		x	x	x			
Visibility Msk	113-118	161-166	Initial value = 40 00 00 00 00 00 (A)	x	x				
Relative Pos.	110	156	Initial value = 1	x	x		1	1	
BRT Tape Drive	76	114	Initial value = 0	x	x	x			

Thus, the user, by modifying the values in the communication area by any of the three methods mentioned above, can direct any of the loader-monitor routines in the Mod 1 Operating System to search for the next program to be loaded and executed.

Table 5-2. Loader-Monitor Searching Options

	Card Loading	Tape Loading	Drum Loading
LOADER-MONITOR	Card Loader-Monitor B or Floating Tape Loader-Monitor C	Tape Loader-Monitor C or Floating Tape Loader-Monitor C	Drum Monitor C
DIRECTION OF SEARCH	Forward only	Forward or backward	Forward or backward
CRITERIA FOR SEARCHING	Program name and segment name	Program name, segment name and/or visibility or relative position and visibility	Program name, segment name, and/or visibility or relative position and visibility
SPECIFIED LIMITS	Within or beyond the boundaries of the current program		
DEVICE AND MEDIA	Input binary run deck in card reader	BRT mounted on tape unit specified in communications area (location 76_{10})	Drum program file located on specified drum unit

The user must indicate to the loader routine the method by which he is entering the parameters and whether he requires a halt before the search for the next program is initiated. He does this in two ways:

1. The address by which he returns to the loader, and
2. The value which he has placed in the Method of Console Call Entry field (location 64_{10}).

This is summarized in Table 5-3.

Table 5-3. Methods of Entering Search Parameters

Method Desired	Type of Return Required	Setting of Location 64_{10}
Loader is to halt to allow the operator to enter the values into the communication area manually through the control panel.	Either the operator manually executes Fixed Start 0 (sets sequence register to 126_8 and presses RUN), or the current program terminates with a branch to the indirect address (General Return Address) stored in location 139_{10}: B/(139).	Either the current program or the operator must set the Method of Console Call Entry field to 01_8.

Table 5-3 (cont). Methods of Entering Search Parameters

Method Desired	Type of Return Required	Setting of Location 64_{10}
Loader is to halt to allow the operator to insert a Console Call card in the card reader and to make any manual entries to the communication area through the control panel.	Same as above.	Either the current program or the operator must set the Method of Console Call Entry field to 00_8.
Loader is to automatically read the next Console Call card in the reader without halting.	The current program must terminate with a branch to the indirect address (Alternate Return Address) stored in location 148_{10}: B/(148).	Setting is ignored.
The parameter values have already been entered by the current program. The loader is to begin searching according to these values without halting.	The current program must terminate with a branch to the Return Address for Normal Call: B/130.	Setting is ignored.

SECTION VI
SAMPLE OPERATING APPLICATIONS

As a brief summary of the components of the Mod 1 Operating System and as a guide to its use and capabilities, this section presents several sample operating applications with their suggested solutions. Each solution is a simple one, yet it takes full advantage of the automatic operating features of the system.

APPLICATION 1 - EASYCODER PROGRAM SPECIALIZATION, ASSEMBLY, AND TEST

The user has three Easycoder source programs (PROGA, PROGB, and PROGC) which he wishes to specialize via Library Processor C, assemble via Easycoder Assembler C, and test via Program Test System C. Figure 6-1 illustrates these processes.

Run Deck Setup

Figure 6-2 illustrates the run deck setup, which is explained below.

1. AACLIB Console Call card - This Console Call card directs Tape Loader-Monitor C or Floating Tape Loader-Monitor C (whichever has been bootstrapped into memory by the operator) to search for, and initiate the loading of, Library Processor C from the systems BRT.

2. Equipment Configuration Descriptor (ECD) card - This card indicates to the Library Processor and Easycoder Assembler programs the equipment configuration available for their use. In this case, standard equipment configuration #2 (five tapes, card reader, card punch, and printer) is selected.

3. System Specific Header card (1HDRΔ) - Identifies the director deck.

4. Easycoder source-language program decks - The source-program decks for PROGA, PROGB, and PROGC.

5. End-of-File card (1EOFΔ) - Signals the end of the input deck to the Library Processor.

NOTE: Because of the equipment configuration selected, Easycoder Assembler C will be loaded and executed immediately following the library processing of the three programs: no Console Call card is required.

6. AAATST Console Call card - This card directs the loader-monitor to search for and load Initializer C, which positions the BRT and modifies the loader-monitor for the Program Test System.

7. AAAG12 Console Call card - This card directs the loader-monitor to load and execute Test Data Generator C, which reads the test data cards following and places them in the specified format on magnetic tape as test data input to PROGA.

8. AAADUM Console Call card - This card directs the loader-monitor to search for and load Memory Dump Control C, which, in turn, loads Memory Dump C. Memory Dump C is not executed at this time, but resides in memory

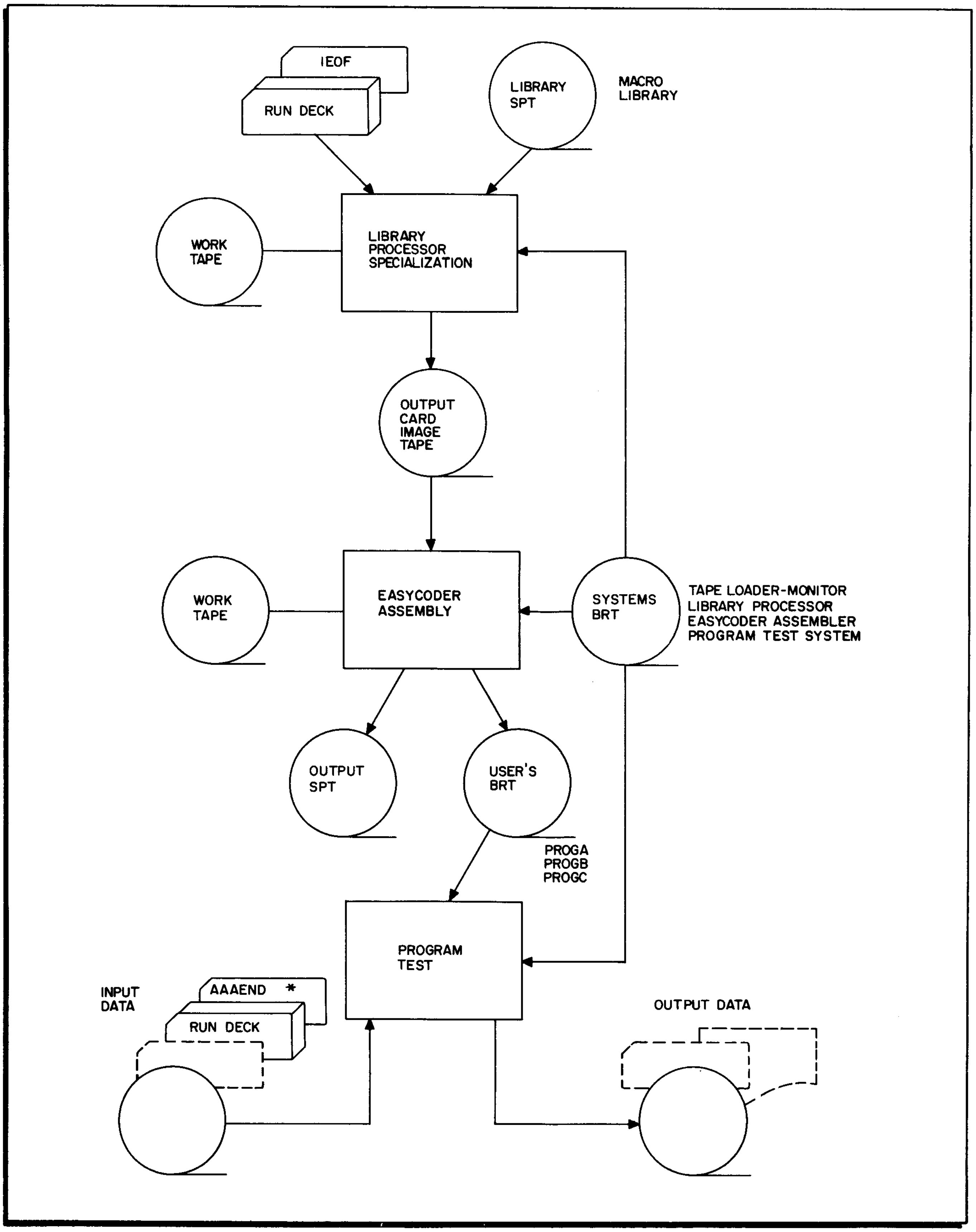


Figure 6-1. Application 1: Run Setup

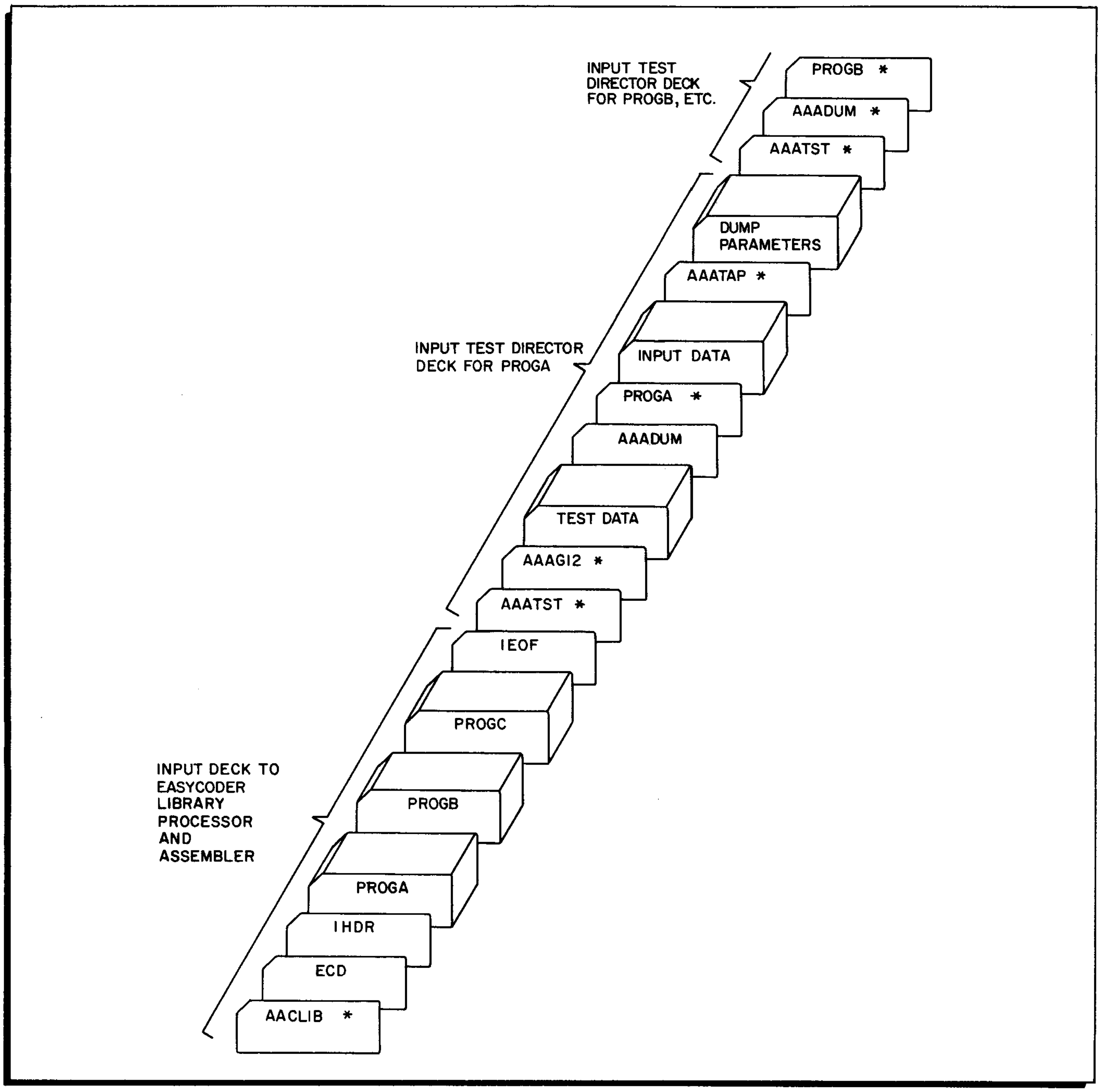

INPUT TEST DIRECTOR DECK FOR PROGB, ETC.
PROGB *
AAADUM *
AAATST *
DUMP PARAMETERS
AAATAP *
INPUT TEST DIRECTOR DECK FOR PROGA
INPUT DATA
PROGA *
AAADUM
TEST DATA
AAAG12 *
AAATST *
IEOF
PROGC
INPUT DECK TO EASYCODER LIBRARY PROCESSOR AND ASSEMBLER
PROGB
PROGA
IHDR
ECD
AACLIB *

Figure 6-2. Application 1: Input Run Deck

ready to produce any memory dumps requested by the next program to be loaded and executed (PROGA).

9. PROGA Console Call card - This card directs the loader-monitor to search for and load the previously specialized and assembled program, PROGA, from the BRT created by the Easycoder Assembler. The user must make sure that the memory dump routines are not overlaid during the loading. PROGA can then, through programmed instruction or item-mark trapping, direct that printouts of the contents of core storage be performed at specified points.

10. Input data - Besides the tape file created by Test Data Generator C, punched card input data is also required input to PROGA.

11. AAATAP Console Call card - This directs the loader-monitor to search for and load Tape Dump C from the system BRT. Once loaded and initiated, Tape Dump C will read the parameter cards following and perform the specified positioning, editing, and printing functions on the tape files just created or processed by PROGA.

12. The remainder of the test director deck consists of similar Console Call cards, test data, input data, etc., for PROGB and PROGC.

APPLICATION 2 - PREPARING AND COMBINING EASYCODER AND COBOL PROGRAMS FOR TESTING

The user has two Easycoder source programs (PROGB and PROGE) to specialize and assemble through Library Processor C and Easycoder Assembler C and three COBOL source programs (PROGA, PROGC, and PROGD) to compile via COBOL Compiler D. Following this, he wants to combine the five object programs onto one BRT for testing by means of Program Test System C. Under the Mod 1 Operating System, he can direct that the five processes, as listed below, be executed with almost no operator intervention.

1. Library Processor C - Specializes the macro routines called for and incorporates them into PROGB and PROGE in preparation for assembly.

2. Easycoder Assembler C - Assembles PROGB and PROGE and produces an output BRT containing the object coding for these two programs.

3. COBOL Compiler D - Compiles PROGA, PROGC, and PROGD and produces an output BRT containing the object coding for these two programs.

4. Update and Select C - Combines the two output BRT's from the assembly and compilation runs above on a master BRT.

5. Program Test System C - Tests the five programs as directed by the various Console Call cards, test data, etc.

Figure 6-3 illustrates these processes.

Run Deck Setup

Figure 6-4 illustrates the run deck setup. Each card or card deck is explained below.

1. AACLIB Console Call card - This card directs either the Tape Loader-Monitor or Floating Tape Loader-Monitor (whichever has been bootstrapped into memory by the operator) to search for, and initiate the loading of, Library Processor C from the systems BRT.

2. Equipment Configurator Descriptor (ECD) card - This card serves to indicate the equipment configuration present.

3. Systems Specific Header card (1HDRΔ) - Identifies the Easycoder director deck.

4. Easycoder source-language program decks - The source-program decks for PROGB and PROGE.

5. End-of-File card (1EOFΔ) - Signals the end of the input deck to the Library Processor.

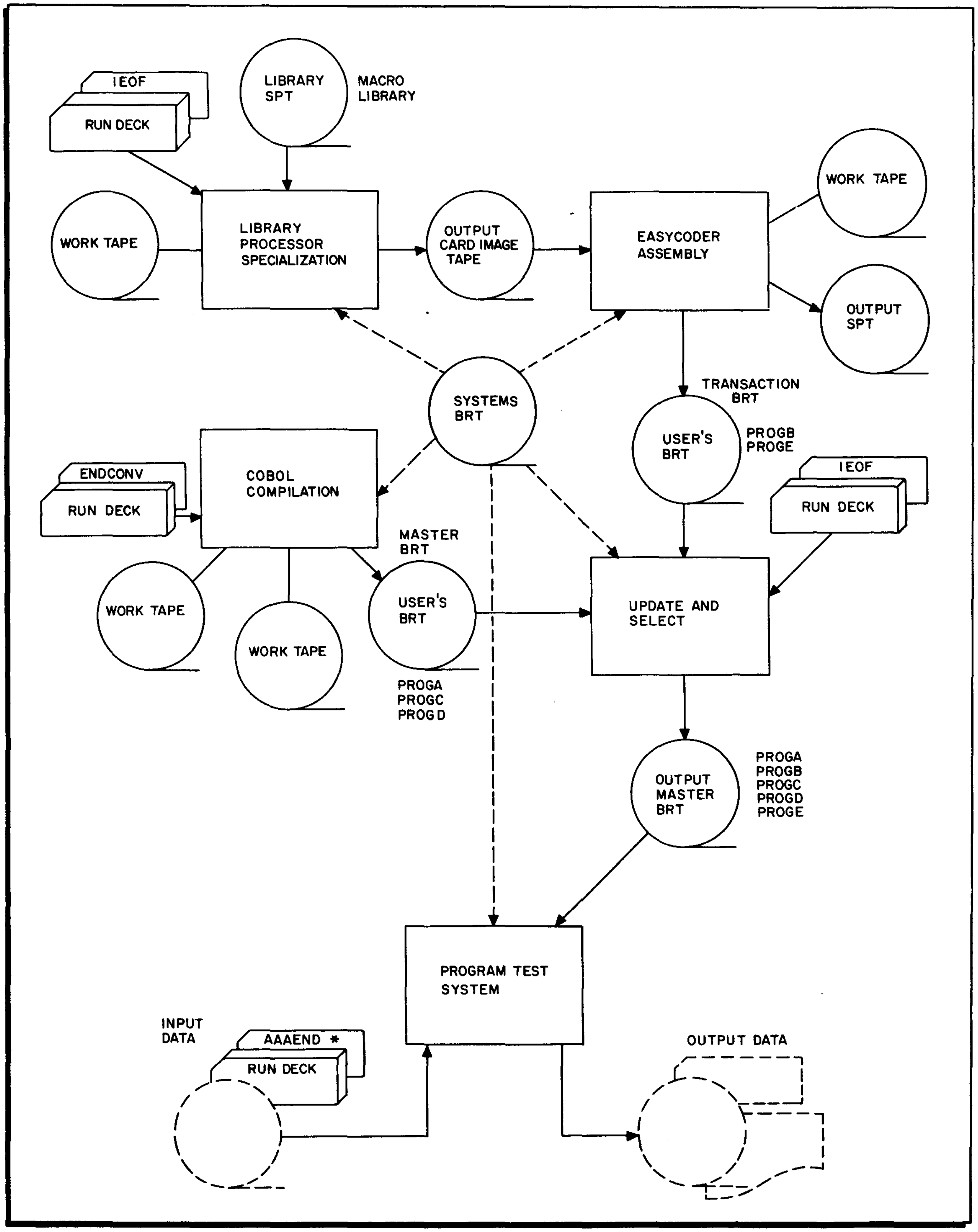


Figure 6-3. Application 2: Run Setup

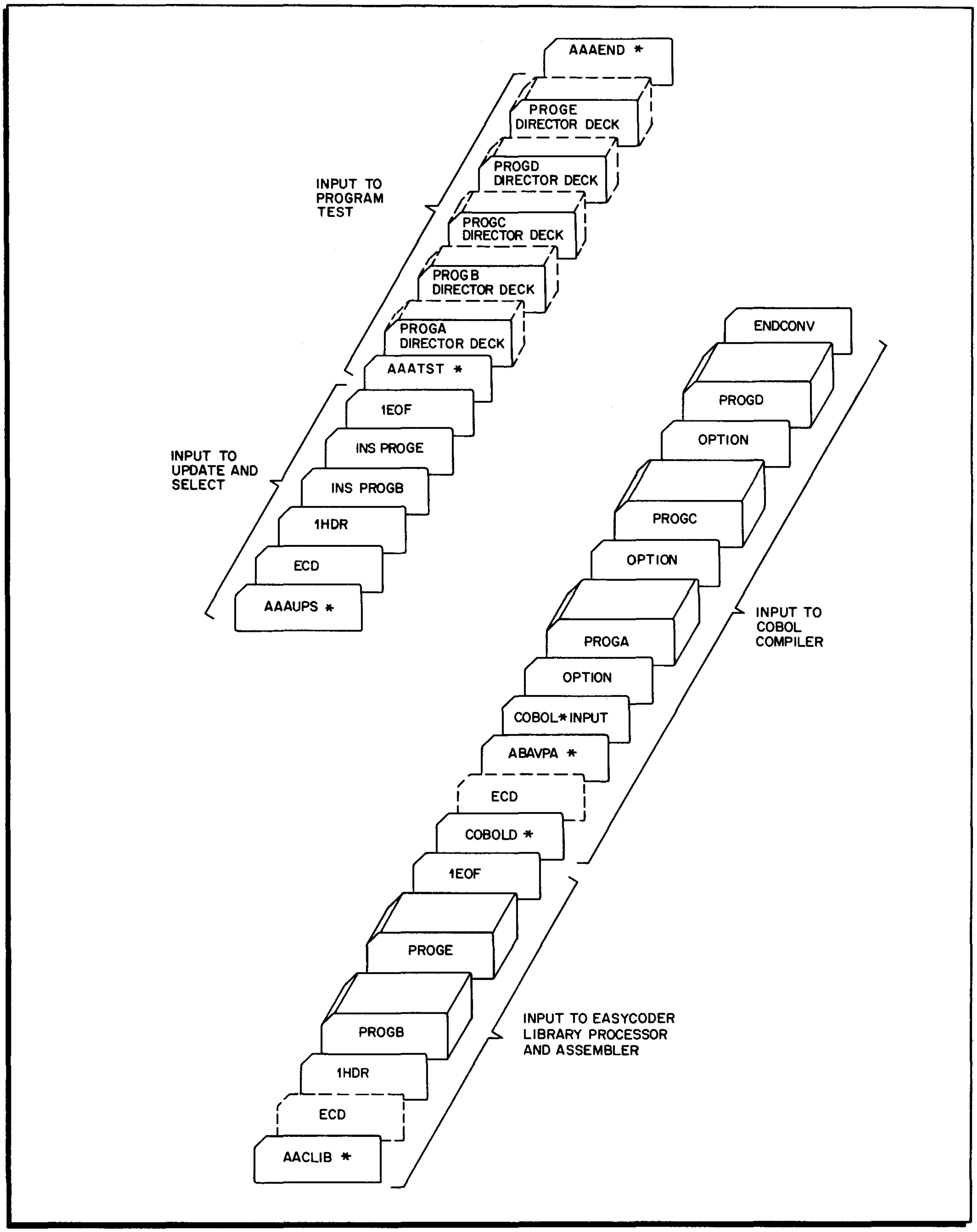


Figure 6-4. Application 2: Input Run Deck

NOTE: If a sufficient number of tape drives has been indicated in the ECD entry, Easycoder Assembler C will be automatically loaded and executed immediately following the specialization of the two programs; no Console Call card is required.

6. COBOL D Console Call card - This card directs the loader-monitor to search for, and initiate the loading of, COBOL Compiler D from the systems BRT.

7. ECD card - Required to indicate the configuration present for compilation.

8. ABAVPA Console Call card - This card directs the loader-monitor to search for and load the initialization routine for the COBOL Compiler.

9. COBOL*INPUT card - This card identifies the beginning of the input deck to the compilation.

10. OPTION card - This card signals that the next program to be compiled follows in punched card format.

11. Source program deck for PROGA.

12. OPTION card.

13. Source program deck for PROGC.

14. OPTION card.

15. Source program deck for PROGD.

16. ENDCONV card - This card identifies the end of the input deck to the compiler.

17. AAAUPS Console Call card - This card directs the loader-monitor to search for and load Update and Select C.

18. ECD card - Required to indicate the configuration present for Update and Select C.

19. Systems Specific Header card (1HDRΔ) - Identifies the Update and Select director deck.

20. INSERT director card - This card directs the Update and Select program to insert PROGB after PROGA on the output master BRT.

21. INSERT director card - This card directs the Update and Select program to insert PROGE after PROGD on the output master BRT.

NOTE: All programs from the COBOL output BRT (designated as the input master BRT) are automatically copied on the output master BRT unless otherwise directed.

22. End-of-File card - This card signals the end of the input deck to Update and Select C.

23. AAATST Console Call card - This card directs the loader-monitor to search for and load Initializer C in preparation for program testing.

24. Following this is a Program Test System director deck for each of the five programs to be tested. Each deck can contain Console Call cards to search for and load Memory Dump Control C, Tape Dump C, Test Data Generator C, etc., test data input, parameter cards, etc.

25. AAAEND Console Call card - Terminates the Program Test System operation.

APPLICATION 3 - LOADING BY VISIBILITY

The user has a series of programs which he has placed on a BRT in the order shown below. The Tape Sort C program is recorded only once (after PROGCC). Each of the programs is run on the days indicated.

PROGRAM NAME	DAYS ON WHICH PROGRAM IS RUN
PROGAA	TUESDAY, FRIDAY
PROGBB	MONDAY, WEDNESDAY
PROGCC	TUESDAY, FRIDAY
SORTC	TUESDAY, FRIDAY
PROGDD	FRIDAY
SORTC	FRIDAY
PROGEE	MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY
COLLATE C	WEDNESDAY, FRIDAY
PROGFF	MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY

First, a visibility code must be assigned to represent each day of the week:

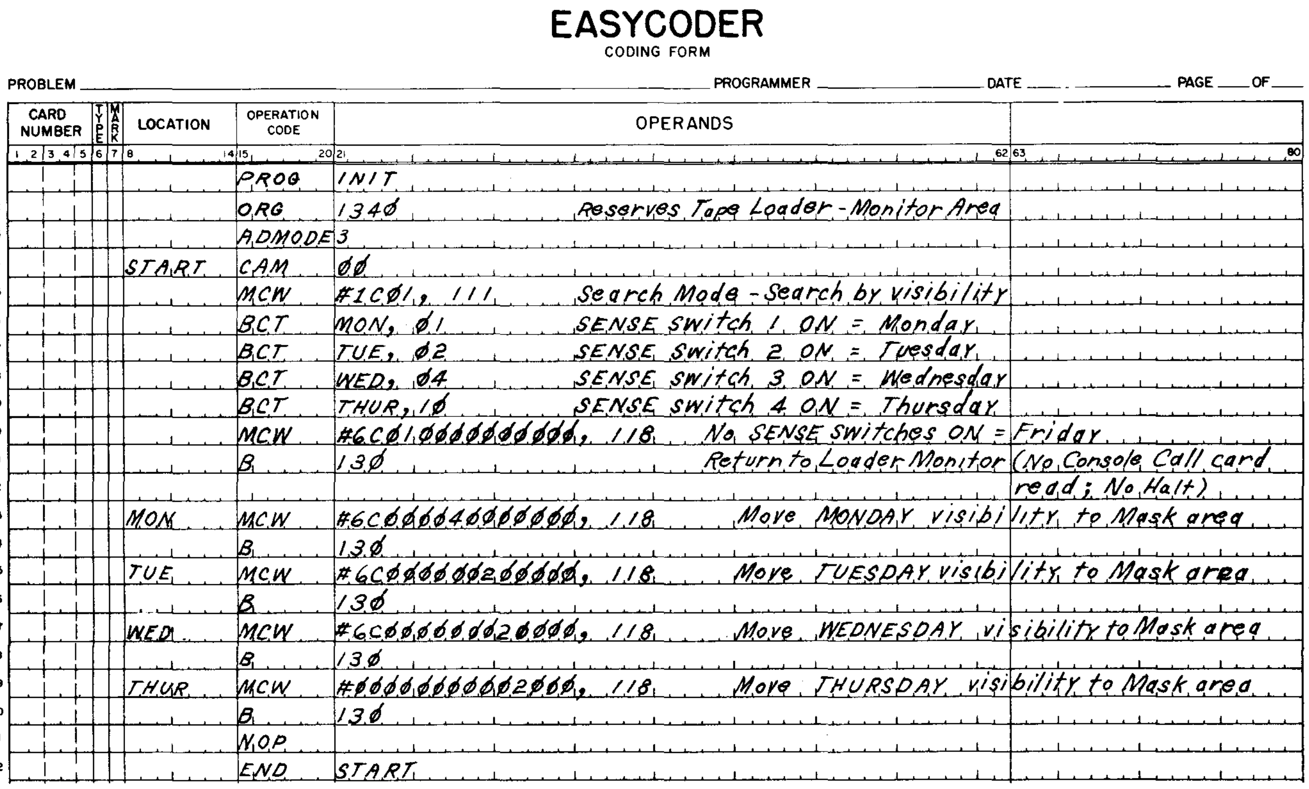
DAY	CODE	VISIBILITY MASK (OCTAL)
MONDAY	M	00 00 40 00 00 00
TUESDAY	T	00 00 00 20 00 00
WEDNESDAY	W	00 00 00 02 00 00
THURSDAY	Z	00 00 00 00 20 00
FRIDAY	F	01 00 00 00 00 00

The appropriate visibilities must now be assigned to the programs according to the codes assigned to each day above. This could have been done at assembly time; however, since the programs are already assembled, this can be accomplished through Update and Select C.

PROGRAM NAME	VISIBILITY CODES (ACCORDING TO DAYS OF THE WEEK)	VISIBILITY (OCTAL) TO BE ASSIGNED
PROGAA	T, F	01 00 00 20 00 00
PROGBB	M, W	00 00 40 02 00 00
PROGCC	T, F	01 00 00 20 00 00
SORTC	T, F	01 00 00 20 00 00
PROGDD	F	01 00 00 00 00 00
PROGEE	ALL	01 00 40 22 20 00
COLLATE C	W, F	01 00 00 02 00 00
PROGFF	ALL	01 00 40 22 20 00

In addition to the programs listed, the user must write a short initialization program, assemble it, and place it at the beginning of the BRT directly after the Tape Loader-Monitor. Called in by a Console Call card, this program initializes the loader communication area in memory to search and load by visibility and places the appropriate day-of-week code in the visibility mask area according to the SENSE switch settings. See Table A-1, page A-5, for the general layout of the loader communication area.

The initializing program (see Figure 6-5) sets the Search Mode field for searching by visibility and, by testing the SENSE switches, moves the proper code constant to the Visibility Mask field. It then branches to the return address for a normal call. The loader-monitor searches forward on the tape and loads the first program having a visibility corresponding to the one indicated by the visibility mask.


EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

```
         PROG    INIT
         ORG     134Ø                     Reserves Tape Loader-Monitor Area
         ADMODE  3
START    CAM     ØØ
         MCW     #1CØ1, 111               Search Mode - Search by visibility
         BCT     MON, Ø1                  SENSE switch 1 ON = Monday
         BCT     TUE, Ø2                  SENSE switch 2 ON = Tuesday
         BCT     WED, Ø4                  SENSE switch 3 ON = Wednesday
         BCT     THUR, 1Ø                 SENSE switch 4 ON = Thursday
         MCW     #6CØ1ØØØØØØØØØØ, 118     No SENSE switches ON = Friday
         B       13Ø                      Return to Loader Monitor (No Console Call card
                                                                   read; No Halt)
MON      MCW     #6CØØØØ4ØØØØØØØ, 118     Move MONDAY visibility to Mask area
         B       13Ø
TUE      MCW     #6CØØØØØØ2ØØØØØ, 118     Move TUESDAY visibility to Mask area
         B       13Ø
WED      MCW     #6CØØØØØØØØ2ØØØ, 118     Move WEDNESDAY visibility to Mask area
         B       13Ø
THUR     MCW     #ØØØØØØØØØØØ2ØØØ, 118    Move THURSDAY visibility to Mask area
         B       13Ø
         NOP
         END     START
```

Figure 6-5. Application 3: Initializing Program

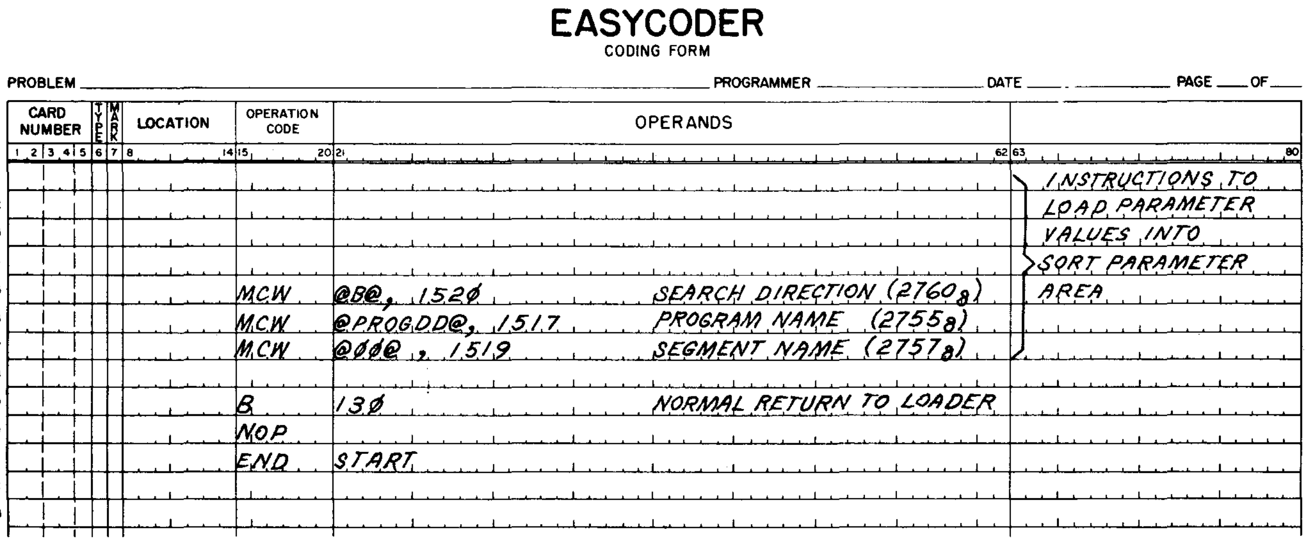
Program Termination

In general, each program must terminate with a branch to the Normal Return Address (B/130). The loader then searches for the next program according to the parameters entered in the loader communication area.

Tape Sort C Programs

The Tape Sort C program is called twice, once after PROGCC and the second time after PROGDD. The parameters (size and location of key fields, record lengths, blocking factors, work tapes, etc.) required by the sort process must be supplied either by parameter cards in the input deck or by MCW instructions in the program prior to each sort. Since the Sort program changes the Search Mode parameter to cause the loader to search by program and segment name, the parameters supplied to the sort must include the search direction, program name, and segment name to be used in searching for the next unit to be loaded and executed after the sort.

PROGCC is the program prior to the first execution of the Sort program and can be used to load the required parameter values into the sort parameter area (locations 2477_8 through 2760_8). These include the parameters which indicate to the loader the search direction ("B" = forward), program name (PROGDD), and segment name (00), to be used in searching for the next program to be loaded and executed after the sort. The coding for moving this information into the sort parameter area is shown in Figure 6-6.


EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1						INSTRUCTIONS TO
2						LOAD PARAMETER
3						VALUES INTO
4						SORT PARAMETER
5				MCW	@B@, 152Ø SEARCH DIRECTION (2760_8)	AREA
6				MCW	@PROGDD@, 1517 PROGRAM NAME (2755_8)	
7				MCW	@ØØ@ , 1519 SEGMENT NAME (2757_8)	
8						
9				B	13Ø NORMAL RETURN TO LOADER	
10				NOP		
11				END	START	
12						
13						

Figure 6-6. Application 3: PROGCC Termination Routine

Once the Tape Sort C program is loaded and initialized, it modifies the loader communication area to direct the loader-monitor to load by program and segment name. When the sorting is completed, the Sort program moves the program and segment names from the sort parameter area to the loader communication area and makes a normal return to the loader-monitor. The loader then searches forward for PROGDD00.

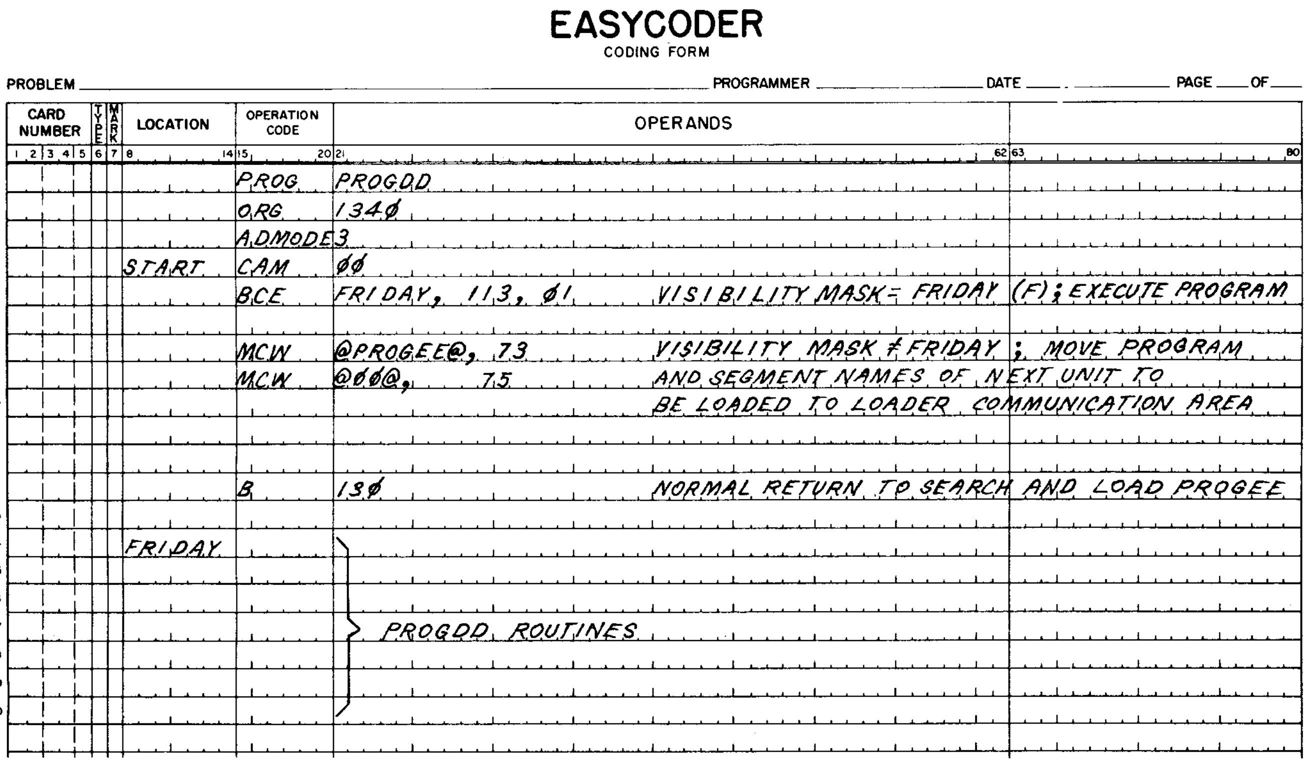
Although the first sort is executed on both Tuesday and Friday, PROGDD is scheduled for Friday only. Therefore, PROGDD must begin with a routine which checks whether the program should be executed. If not, the program and segment name (PROGEE00) of the next unit to be loaded should be moved to the loader communication area and a normal return made to the loader. The Search Mode parameter is still set to search by program and segment name. This initialization routine is shown in Figure 6-7.

The terminating routine for PROGDD must:

1. Move the required parameter values to the sort parameter area. These must include the program name (PROGEE), segment name (00), and search direction ("B" = forward) for the unit to be loaded and executed following the termination of the sort.

2. Set the loader communication area Search Direction, Program Name, and Segment Name fields to direct the loader to search backwards for the Tape Sort C program (AADS2Δ00).

The coding is shown in Figure 6-8.

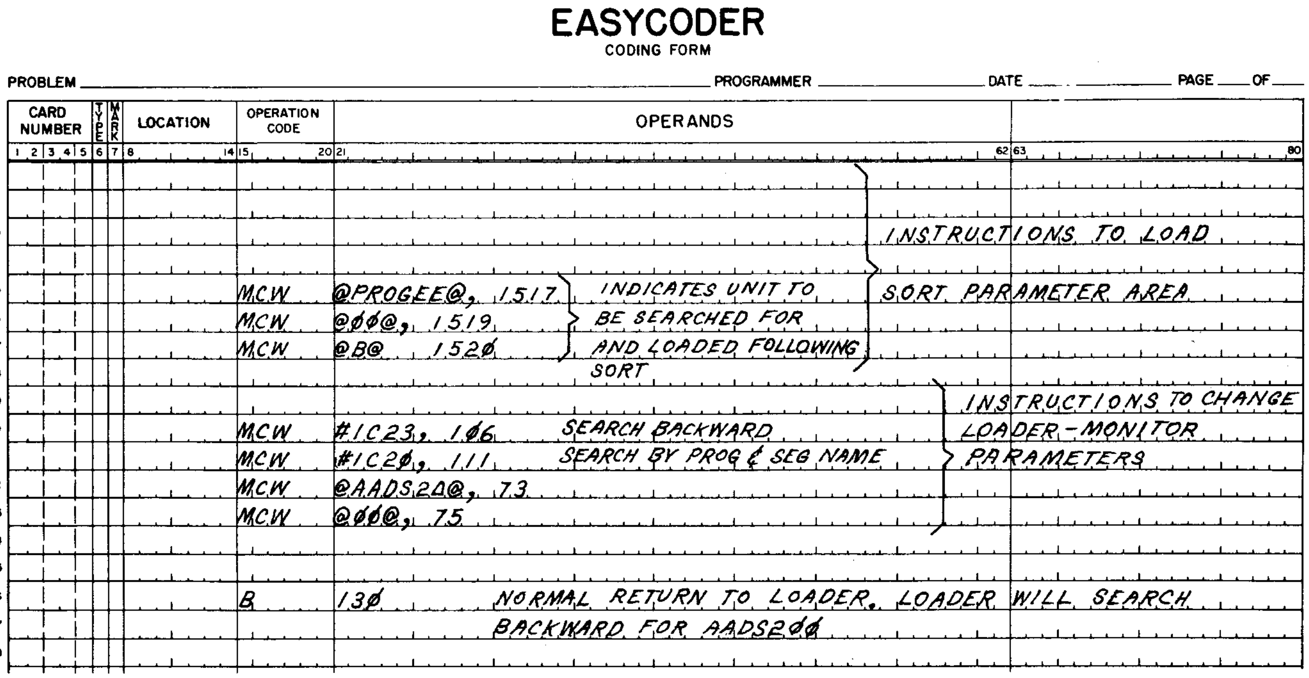


EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

```
CARD    T M
NUMBER  Y A  LOCATION  OPERATION  OPERANDS
        P R            CODE
        E K
1 2 3 4 5 6 7 8     14 15      20 21                                  62 63            80

                       PROG       PROGDD
                       ORG        134Ø
                       ADMODE     3
               START   CAM        ØØ
                       BCE        FRIDAY, 113, Ø1      VISIBILITY MASK = FRIDAY (F); EXECUTE PROGRAM

                       MCW        @PROGEE@, 73         VISIBILITY MASK ≠ FRIDAY ; MOVE PROGRAM
                       MCW        @ØØ@,     75         AND SEGMENT NAMES OF NEXT UNIT TO
                                                       BE LOADED TO LOADER COMMUNICATION AREA


                       B          13Ø                  NORMAL RETURN TO SEARCH AND LOAD PROGEE

               FRIDAY             ⎫
                                  ⎪
                                  ⎪
                                  ⎬ PROGDD ROUTINES
                                  ⎪
                                  ⎪
                                  ⎭
```

Figure 6-7. Application 3: PROGDD Initialization Routine



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

```
CARD    T M
NUMBER  Y A  LOCATION  OPERATION  OPERANDS
        P R            CODE
        E K
1 2 3 4 5 6 7 8     14 15      20 21                                  62 63            80

                                                                    ⎫
                                                                    ⎪
                                                                    ⎪ INSTRUCTIONS TO LOAD
                                                                    ⎬
                       MCW        @PROGEE@, 1517 ⎫ INDICATES UNIT TO ⎪ SORT PARAMETER AREA
                       MCW        @ØØ@, 1519     ⎬ BE SEARCHED FOR   ⎪
                       MCW        @B@   152Ø     ⎭ AND LOADED FOLLOWING ⎭
                                                   SORT
                                                                         ⎫ INSTRUCTIONS TO CHANGE
                       MCW        #1C23, 1Ø6     SEARCH BACKWARD         ⎪ LOADER-MONITOR
                       MCW        #1C2Ø, 111     SEARCH BY PROG ¢ SEG NAME ⎬ PARAMETERS
                       MCW        @AADS2Δ@, 73                           ⎪
                       MCW        @ØØ@, 75                               ⎭


                       B          13Ø          NORMAL RETURN TO LOADER. LOADER WILL SEARCH
                                               BACKWARD FOR AADS2ØØ
```

Figure 6-8. Application 3: PROGDD Termination Routine

When PROGDD makes the normal return to the loader (B/130), the loader searches backwards, finds the first segment of the Tape Sort C program, and loads and executes it. At the completion of the sorting, the Sort program moves the search direction, program name, and segment name values supplied by the PROGDD coding from the sort parameter area to the loader communication area and makes a normal return to the loader. The loader then searches forward for PROGEE00.

The termination routine for PROGEE must:

1. Load the parameter values for Collate C into the collate parameter area in case the Tape Collate C program is to be executed after PROGEE. These parameter values must include the search direction, program name, and segment name to be used in loading the next unit to be executed following the Collate program.
2. Change the Search Mode field (which has been altered by Sort C) back to 01_8.

This coding is shown in Figure 6-9.

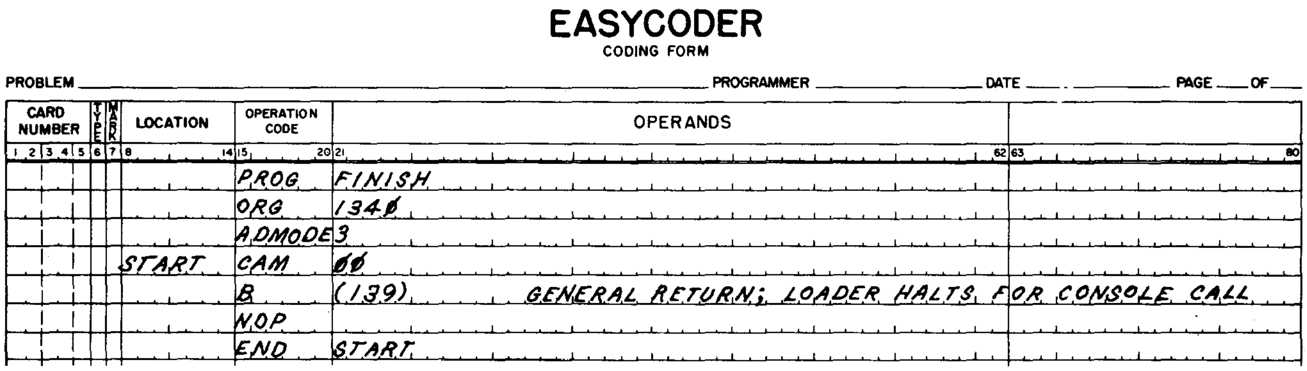
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8–14	OPERATION CODE 15–20	OPERANDS 21–62	63–80
1							
2							INSTRUCTIONS TO
3							LOAD PARAMETER
4							VALUES INTO COLLATE
5					MCW	@PROGFF@, 1499 MOVE PROGRAM NAME	PARAMETER AREA
6					MCW	@ØØ@, 15Ø1 MOVE SEGMENT NAME	
7					MCW	@B@ , 15Ø2 MOVE SEARCH DIRECTION	
8							
9							
10					MCW	#1CØ1, 111 RESTORE SEARCH MODE TO	Ø1 (SEARCH BY
11						VISIBILITY)	
12							
13					B	13Ø NORMAL RETURN TO LOADER	
14					NOP		
15					END	START	
16							

Figure 6-9. Application 3: PROGEE Termination Routine

If more programs were to be loaded by visibility following PROGFF, the Search Mode field would again have to be restored to 01_8 following Collate C (Collate C, like Sort C, searches by program name and segment name). In this example, PROGFF should terminate with an indirect branch to the General Return Address of the loader — B/(139). This directs the loader-monitor to halt until the operator manually intervenes. If PROGFF were not always the last program to be executed in the series, a dummy program might be written as shown in Figure 6-10. This program would be visible to all visibility codes.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8–14	OPERATION CODE 15–20	OPERANDS 21–62	63–80
1					PROG	FINISH	
2					ORG	134Ø	
3					ADMODE	3	
4				START	CAM	ØØ	
5					B	(139) GENERAL RETURN; LOADER HALTS F	OR CONSOLE CALL
6					NOP		
7					END	START	

Figure 6-10. Application 3: Run Termination Program (Visible to All Visibilities)

Input Run Deck

Since in this application the loader is directed to load by visibility, or in some cases by a program and segment name placed in the communication area via programmed instructions, the input deck is a very simple one.

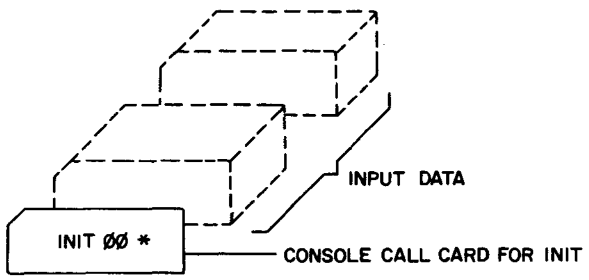


Figure 6-11. Application 3: Input Run Deck

APPENDIX A
FORMATS

For the convenience of the reader, the following layouts pertinent to the Mod 1 Operating System are presented:

1. Symbolic program tape (SPT) format
2. Binary run tape (BRT) format
3. Binary run deck (BRD) format
4. Loader-Monitor communication area layout
5. Equipment Configuration Descriptor (ECD) card format

SYMBOLIC PROGRAM TAPE (SPT) (Figure A-1)

Symbolic program tapes (SPT's) are processed and produced within the Easycoder system and contain the user's programs in both source- and machine-language format. They are input to, and output from, Library Processors C and D, Easycoder Assemblers C and D, and SPT Merge C. Since both the source-language and machine-language elements of each program are present on the tape, either version can be selected and either punched into cards or written onto tape.

A symbolic program tape begins with a beginning-of-file header label. For each program stored on the tape there is a program header record and one or more segment header records each followed by one or more data records. These data records (see Figure A-2) contain the source- and machine-language coding for the segment. At the end of a symbolic program file, there is an end-of-file record, a file of directory records, another end-of-file record, and two end-of-information records.

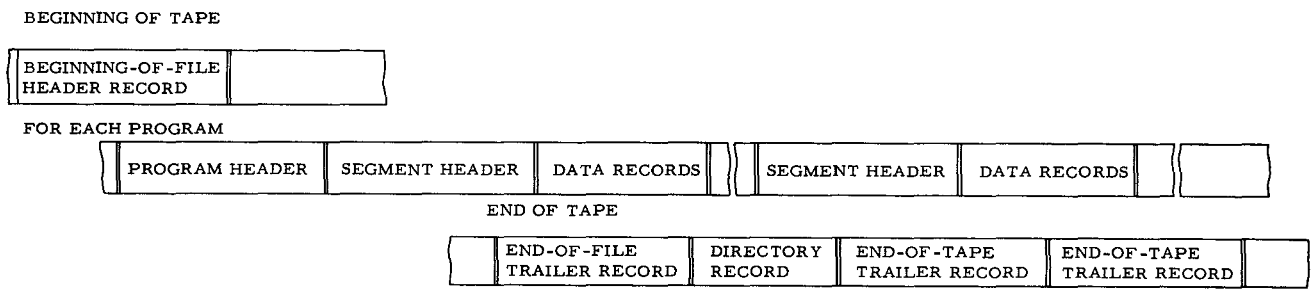
BINARY RUN TAPE (Figure A-3)

A binary run tape (BRT) contains the machine-language coding of one or more object programs. These programs can be loaded directly from the tape into main memory by either Tape Loader-Monitor C or Floating Tape Loader-Monitor C (which can reside at the beginning of the tape) and executed.

At the beginning of each BRT are a beginning-of-file header record, a tape bootstrap routine (which loads the Tape Loader-Monitor), and one or more versions of the Tape Loader-Monitor. For each program on the tape, there are one or more loading units (or segments), each consisting of a segment header record and one or more non-header records. The segment header record identifies the program and the segment and contains the revision number of the segment, the visibilities under which the segment is to run, and part of the machine-language coding of the program. Every non-header record consists of a control field to identify the record and a portion of the program's machine-language coding. Each loading unit contains as many non-header records as are required for the storage of the object coding. The program file is terminated by one end-of-file trailer record and two end-of-information records.

BINARY RUN DECKS (Figure A-4)

Binary run decks (BRD's) can be produced by either Easycoder Assemblers C and D or BRT Punch C and can be loaded for execution by Card Loader-Monitor B. Such program decks are actually the machine-language coding of the programs converted from a binary run tape into punched cards. As in the case of a BRT, the deck begins with a beginning-of-file header record. Each segment or loading unit begins with a segment header record which is followed by one or more non-header records containing the machine-language coding for the program. The format of these records is basically identical to that of the corresponding record types on a BRT with the exception that BRT records have a limit of 250 characters while BRD cards are limited to 80 characters.



BEGINNING-OF-FILE HEADER RECORD

1-5	1HDR
6-10	Unspecified
11-15	SPT revision number
16-20	Unspecified
21-30	ECDR2SPT
31-80	Unspecified

PROGRAM HEADER RECORD

1	Banner character (51_8)
2-4	Number of characters in record (124_8)
5-6	Number of characters in item (120_8)
7-12	Program name
13-15	Program revision number
16-21	Visibilities
22-29	Date of assembly
30-34	SPT number
35-84	Unspecified

SEGMENT HEADER RECORD

1	Banner character (50_8)
2-4	Number of characters in record (124_8)
5-6	Number of characters in item (120_8)
7-12	Program name
13-14	Segment name
15-84	Unspecified

END-OF-INFORMATION RECORDS (2)

1-5	1ERI
6-80	Unspecified

DATA RECORDS
(See Figure A-2)

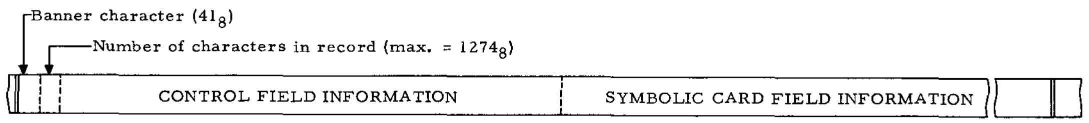
END-OF-FILE TRAILER RECORD

1-5	1EOF
6-80	Unspecified

DIRECTORY RECORD

1	Banner character (43_8)	
2-4	Number of characters in record (764_8 - maximum)	
5-6	Number of characters in item (31_8)	Repeated for each program on the SPT
7-12	Program name	
13-15	Program revision number	
16-21	Visibilities	
22-29	Date of assembly	

Figure A-1. Symbolic Program Tape (SPT) Format


Banner character (41_8)
Number of characters in record (max. = 1274_8)
CONTROL FIELD INFORMATION
SYMBOLIC CARD FIELD INFORMATION

CONTROL FIELD INFORMATION (31-70 characters in length)

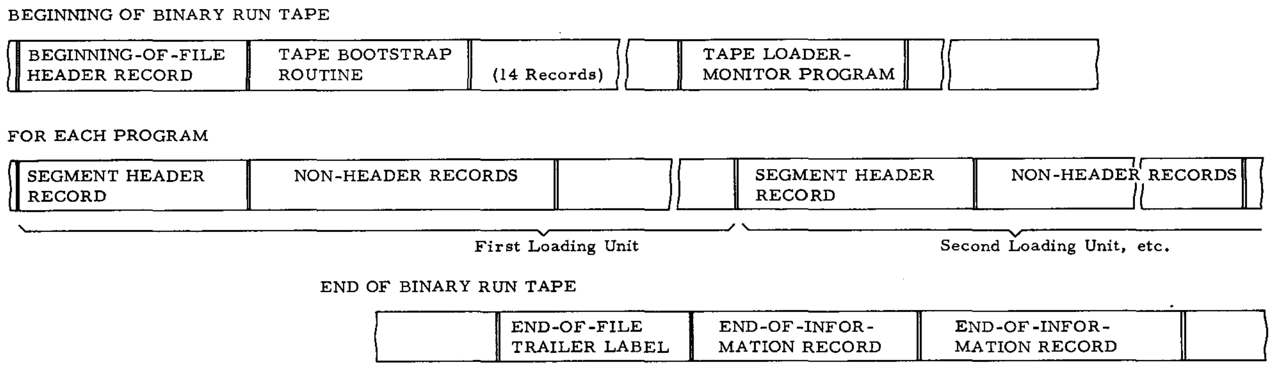
1-2	Number of characters in item
3-4	Number of characters in control information field
5-6	Number of characters in symbolic information field
7	Item type
8-12	Line number of symbolic card
13	Allocation information
14-19	Subfield reserved for future use
20-22	Error code
23	Label information
24	Word mark and item mark information
25	Not used
26-29	Beginning memory address of instruction or constant
30	Length of machine-language entry (binary)
31 -	Machine language

SYMBOLIC CARD FIELD INFORMATION (15-75 characters in length)

1	Type field
2	Mark field
3-9	Location field
10-15	Op code field
16 -	Operands field (0-60 characters)

NOTE: Literal items, repeated items, and generated items do not have Symbolic Card Fields.

Figure A-2. Symbolic Program Tape (SPT) Format: Data Record Layout


BEGINNING OF BINARY RUN TAPE
BEGINNING-OF-FILE HEADER RECORD
TAPE BOOTSTRAP ROUTINE
(14 Records)
TAPE LOADER-MONITOR PROGRAM
FOR EACH PROGRAM
SEGMENT HEADER RECORD
NON-HEADER RECORDS
SEGMENT HEADER RECORD
NON-HEADER RECORDS
First Loading Unit
Second Loading Unit, etc.
END OF BINARY RUN TAPE
END-OF-FILE TRAILER LABEL
END-OF-INFORMATION RECORD
END-OF-INFORMATION RECORD

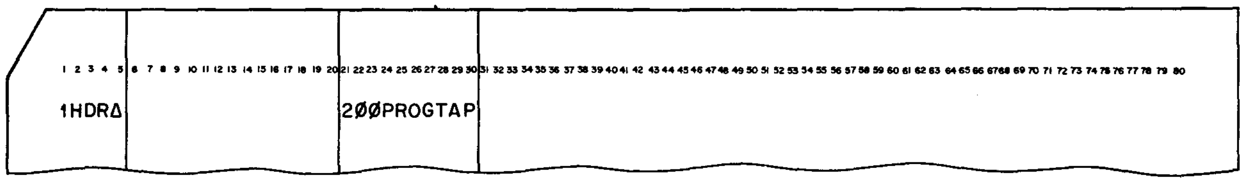
SEGMENT HEADER RECORD

1	Banner character 50_8 - This is the beginning record of a multirecord loading unit 54_8 - This is the beginning and only record of a single-record loading unit
2-4	Number of characters in record (binary)
5-6	Record sequence number (used in backspacing to beginning (segment header) record of load unit
7	Length of identification and control field information (30_8)
8-10	Revision number
11-16	Program name
17-18	Segment name
19-24	Visibilities
25-250	Machine language to be loaded interspersed with control characters

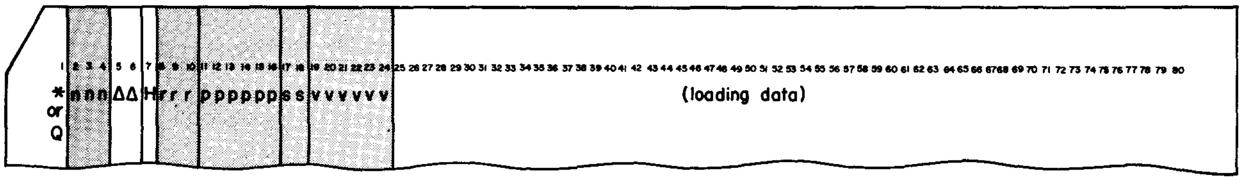
NON-HEADER RECORD

1	Banner character 41_8 - This is not the last record of the current loading unit (segment) 44_8 - This is the last record of the current loading unit
2-4	Number of characters in record
5-6	Record sequence number
7	Length of identification and control field information (07_8)
8-250	Machine language to be loaded interspersed with control characters

Figure A-3. Binary Run Tape (BRT) Format

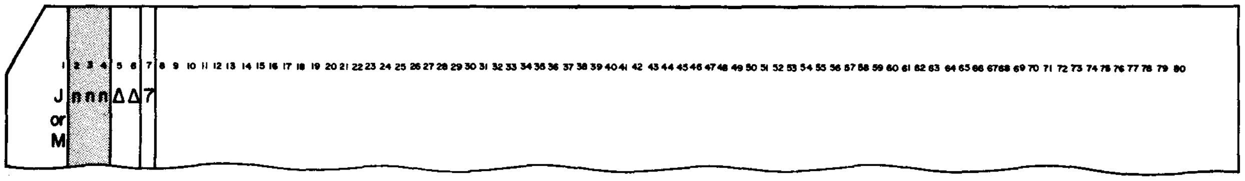


Header Label Record



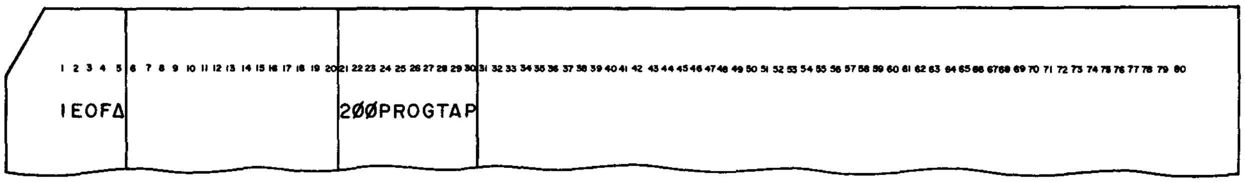
Segment Header Record

Col. 1 -	* (54_8) = Last record of load unit Q (50_8) = Not last record of load unit
Col. 2-4	Card sequence number
Col. 5-6	Blanks
Col. 7-24	Equivalent to characters 7 through 24 of a BRT segment header record
Col. 25-80	Machine language to be loaded interspersed with control information.



Non-Header Record

Col. 1 -	M (44_8) = Last record of loading unit J (41_8) = Not last record of loading unit
Col. 2-4	Card sequence number
Col. 5-6	Blanks
Col. 7	Number of control field characters (7)
Col. 8-80	Machine language to be loaded interspersed with control information.



Trailer Label Record

Figure A-4. Binary Run Deck (BRD) Format

LOADER-MONITOR COMMUNICATION AREA (Table A-1)

The loader-monitor communication area, consisting of 92 locations in main memory, is utilized by all of the loader-monitors in the Mod 1 Operating System as a control link between one program run and the following program run(s) and between the loader-monitor routine and the operator. Table A-1 shows the fields within this area which are most likely to be referenced by the user.

The communication area contains:

1. Parameters which control the searching, loading, and starting operations of the loader-monitors.
2. Entry points for transferring control back to the loader-monitors.
3. Exit and return points for own-coding routines to be executed during loading.
4. Parameters provided for use by other programs.

All fields, except for Program Name, Segment Name, and Halt Name, are initially set to certain standard values. All fields can be modified either by programmed instruction or by control panel entries. The Program Name, Segment Name, and Tape Unit Address fields can be modified by a Console Call card. Some fields are automatically reset to their initial value by a console call, special call, or the loading of a program unit.

Table A-1. Loader-Monitor Communication Area (Basic Fields)

Field	Contents[1]	Locations	
		Decimal	Octal
METHOD OF CONSOLE CALL ENTRY	00_8 = Card 01_8 = Manual	64	100
PROGRAM NAME[2]		68-73	104-111
SEGMENT NAME[2]		74-75	112-113
TAPE UNIT ADDRESS OF BRT[2]	00_8	76	114
FIXED START 0	(Manual Return to Loader-Monitor for Next Console Call)	86-89	126-131
SEARCH DIRECTION[3]	22_8 = Forward 23_8 = Backward	106	152
RELATIVE POSITION[3]	(Used With Search Mode 01) 01_8	110	156
SEARCH MODE[3]	20_8 = Program and Segment Name 01_8 = Visibility and Relative Position 00_8 = Segment Name Within Current Program 60_8 = Program, Segment, and Visibility 40_8 = Segment, Visibility Within Current Program	111	157

Table A-1 (cont). Loader-Monitor Communication Area (Basic Fields)

Field	Contents[1]	Locations Decimal	Locations Octal

START MODE[3]	N = Begin Execution at Address Specified in EX Command
	S = Begin Execution at Address Stored in Special Start Location
	R = After Loading, Return to Point Immediately After that Where Exit to Loader Was Made.
		112	160

VISIBILITY MASK	(Initial Value = Visibility "A")	113-118	161-166
SPECIAL START LOCATION	(Used With Start Mode "S")	119-121	167-171
RETURN ADDRESS FOR NORMAL CALL	(To Load Another Unit Without Halting)	130-138	202-212
GENERAL RETURN ADDRESS	(Halts for Console Call)	139-141	213-215
CURRENT DATE		142-146	216-222
TRAPPING MODE	00_8 = Off 04_8 = On	147	223
ALTERNATE RETURN ADDRESS	(Reads Next Console Call Card Without Halting)	148-150	224-226
ECD FIELD	JJ0# - ECD Entered From Card Reader	151-154	227-232

CONSOLE TYPEWRITER AVAILABILITY	IM = Not Available
	WM = Available
		155	233

NOTES: 1. Initial or Reset Value is First Value Shown

2. Can be taken from Console Call card

3. Reset by Fixed Start 0 or General Return console call

EQUIPMENT CONFIGURATION DESCRIPTOR (ECD) CARD FORMAT (Table A-2)

Systems programs (Easycoder Assemblers, COBOL Compilers, etc.) require that the user indicate the equipment configuration available for their execution. This may be done in one of two ways: either by punching an Equipment Configuration Descriptor (ECD) card and placing it after the Console Call card in the card reader, or by indicating that one of the ten standard equipment configuration descriptors automatically loaded into memory with the systems program is to be used.

ECD Card

The user punches an ECD card (see Table A-2). The specific meaning and content of the file media fields are discussed in the software manual for the particular systems program. The ECD field (151-154) of the loader communication area is initially set to cause the ECD information to be accepted from the card reader (JJ0#).

Standard ECD Entries

Normally, the user can specify that one of the standard equipment configurations loaded in conjunction with the systems program is to be used. He indicates this choice to the system by manually entering the following information into the ECD field (151-154) of the loader communication area. This field is reset only by another manual entry.

Locations		Contents
Decimal	Octal	
151	227	Blank
152	230	Standard configuration number (0-9) desired
153-154	231-232	Highest memory bank available (if blank, will use memory size indicated in ECD)

Table A-2. Equipment Configuration Descriptor (ECD) Card Format

Column(s)	Contents	Interpretation
1-5	Blanks	
6	E	Identifies Equipment Configuration Descriptor (ECD) card
7	Blank	
8	11,9	Read/write channel assignment for RWC1
9	8,2	Read/write channel assignment for RWC2
10	8,3	Read/write channel assignment for RWC3
11-15	Blanks	
16-17	00	Lowest memory bank usable
18	Blank	
19-20		Highest memory bank available
21-80		File media fields (three columns per file)
		First character position: Type of device
	Blank	File absent
	0	Unspecified
	1	Type 204B Magnetic Tape Unit
	2	Control Panel
	3	Type 204A Magnetic Tape Unit
	4	Type 270 Drum Storage Unit
	5	Type 220 Console
	6	Main Memory
	-	Printer
	J	Type 227 Card Reader
	K	Type 227 Card Punch
	L	Type 209 Paper Tape Reader
	M	Type 210 Paper Tape Punch
	N	Type 223, 214-2, or 224 Card Reader with Series 200 Card Reader Control

Table A-2 (cont). Equipment Configuration Descriptor (ECD) Card Format

Column(s)	Contents	Interpretation
21-80 (cont)	O	Type 214-1, 214-2, or 224 Card Punch with Series 200 Card Punch Control
	R	Type 123 Card Reader (or Type 214-2 or 224 Card Reader/Punch used as card reader only) with Model 120 Integrated Card Control
	S	Type 214-1 Card Punch (or Type 214-2 or 224 Card Reader/Punch used as card punch only) with Model 120 Integrated Card Control
		Second character position: Peripheral Address (Control character C_2 of PDT instruction)
		Third character position: Tape drive number (Control character C_3 of PDT instruction)

APPENDIX B

MOD 1 OPERATING SYSTEM PUBLICATIONS

This appendix contains a current listing of the Honeywell publications associated with the Mod 1 Operating System. The order numbers shown in parentheses should be used in ordering these publications.

GENERAL INTRODUCTION

Introduction to Series 200/Operating System - Mod 1 (258)

OPERATING PROCEDURES

Operating System - Mod 1 Operating Procedures Summaries (069)

LANGUAGE PROCESSING

Honeywell Series 200 (Model 120) Programmers' Reference Manual (141)
Honeywell Series 200 (Models 200/1200/2200) Programmers' Reference Manual (139)
Library Processors C and D (051)
Easycoder Assemblers C and D (041) ✓
Transition to Easycoder - A Programmer Text (238)
Programming with Easycoder - A Programmed Text (008)
Analyzer C (019)
COBOL Compilers D & H (065)
COBOL Compiler D - Volume 1 - A Programmed Text (083)
COBOL Compiler D - Volume 2 - A Programmed Text (091)
COBOL Compiler D - Volume 3 - A Programmed Text (294)
Study Guide: COBOL Programming (A three-volume set) (259, 260, 261)
Classroom Workbook - COBOL Programming
Fortran Compiler D Reference Handbook (027)
Fortran Compiler D Generated Object Code (003)
Fortran Conversion Techniques (002)
Fortran D Action Session (114)
Easytran Symbolic Translators B and C (035)
Easytran Symbolic Translator D (220)
Easytran Program Modifier C (147)

UTILITY PROGRAMS

Tape Handling Routine B (applicable to Tape Handling Routine C) (017)
Data Conversion A and C (231)
Simultaneous Media Conversion A and C (021)
Report Generator A, B, and C (080)
Tape Sort C and Collate C (017)
Own Coding Routines for Tape Sort C (026)
Sort C (V) and Collate C (V) (207)
Drum Sort C (157)
Simultaneous Sort and Print (201)

Statistics Package D (159)
Linear Programming Package D (276)

PROGRAM EDITING AND MAINTENANCE

SPT Merge C (152)
Update and Select C and D (025)
BRT Punch C (020)
Drum Program Store C (DSI-411)

OPERATION CONTROL

Tape Loader-Monitor C (221)
Floating Tape Loader-Monitor C and Interrupt Control D (005)
Card Loader-Monitor B (154)
Drum Bootstrap-Loader C (DSI-415)
Drum Monitor C (DSI-408)
List Comments (DSI-353)

INPUT/OUTPUT CONTROL

1/2-Inch Tape I/O B and C (010)
1/2-Inch Tape and Terminal I/O C (167)
Drum I/O C (DSI-405)
Console I/O C (TYRO 2) (DSI-413)
Communications I/O C (202)

PROGRAM TEST FACILITIES

Program Test System C (049)
Memory Dump C and Tape Dump C (469)

COMPUTER-GENERATED INDEX

Cut Along Line

HONEYWELL EDP TECHNICAL PUBLICATIONS
USERS' REMARKS FORM[1]

TITLE: SERIES 200 INTRODUCTION TO SERIES 200/OPERATING SYSTEM - MOD 1 (TAPE RESIDENT) SOFTWARE MANUAL

DATED: AUGUST, 1966

FILE NO: 123.0005.001C.1-258

ERRORS NOTED:

SUGGESTIONS FOR IMPROVEMENT:

Fold

FROM: NAME ____________________ DATE __________

COMPANY ____________________

TITLE ____________________

ADDRESS ____________________

____

[1] Please restrict remarks to the publication itself. Comments concerning hardware/software difficulties and improvement requests should be submitted through the channels established for that purpose.

FIRST CLASS
PERMIT NO. 39531
WELLESLEY HILLS
MASS.

BUSINESS REPLY MAIL

No postage stamp necessary if mailed in the United States

POSTAGE WILL BE PAID BY

HONEYWELL

ELECTRONIC DATA PROCESSING DIVISION

60 WALNUT STREET

WELLESLEY HILLS, MASS. 02181

ATT'N: TECHNICAL COMMUNICATIONS DEPARTMENT

Cut Along Line

Honeywell

ELECTRONIC DATA PROCESSING