HONEYWELL EDP

SOFTWARE BULLETIN

SERIES 200

TRANSITION TO THE MOD 1 OPERATING SYSTEM

GENERAL SYSTEM:	SERIES 200/PROGRAMMING SYSTEMS
SUBJECT:	Transition procedures to convert programs written for the Basic Programming System to run in the Mod 1 Operating System.
DATE: April 22, 1966	FILE NO. 122.0005.0000.0-317*

8865
8466
Printed in U.S.A.

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

FOREWORD

Programs operating in the Basic Programming System may have been coded directly in the Honeywell Easycoder Symbolic Language, or they may have been translated from SPS or Autocoder programs by Easytran 1401 or Easytran Symbolic Translator B. This bulletin considers only programs that have been coded directly in Easycoder. Translated programs present other problems and for them there is a Honeywell-supplied program, Easytran Program Modifier C, that will make the appropriate modifications. Alternatively, the original SPS or Autocoder program may be retranslated with Easytran Symbolic Translator C or D. Appropriate procedures will be found in the manuals for these programs — Easytran Program Modifier C Manual, Order No. 147; Easytran Symbolic Translator B and C Reference Manual, Order No. 035; or Easytran Symbolic Translator D Manual, Order No. 220.

The reader of this bulletin is assumed to be familiar with the Honeywell Series 200 Programmers' Reference Manual, Order No. 139 and the Easycoder B Assembly System Manual, Order No. 011. Other relevant documents include the Easycoder Assemblers C and D Manual, Order No. 041; the Tape Loader-Monitor C Manual, Order No. 221; and the Floating Tape Loader-Monitor C Manual, Order No. 005.

TABLE OF CONTENTS

		Page
Section I	Introduction	1-1
	General	1-1
	Systems Description	1-1
	Basic	1-1
	Mod 1	1-2
Section II	Procedures	2-1
	General	2-1
	Relocation	2-2
	Address Mode	2-3
	Assembly Control Statements	2-4
	Program Header — PROG	2-5
	Segment Header — SEG	2-6
	Set Address Mode and ADMODE	2-7
	Clear — CLEAR	2-7
	Origin — ORG	2-7
	Literal Origin — LITORG	2-7
	Control Equals — CEQU	2-8
	Memory Dump — HSM	2-8
	Execute — EX	2-9
	End — END	2-10
	Machine Instructions	2-12

LIST OF ILLUSTRATIONS

Figure 2-1.	Memory Maps Showing Basic Loaders	2-1
Figure 2-2.	Memory Maps Showing Mod 1 Loaders	2-2
Figure 2-3.	Makeup of the Symbolic Decks	2-4
Figure 2-4.	PROG Cards for Basic and Mod 1	2-5

LIST OF TABLES

Table 1-1.	Comparison of Systems Programs	1-3
Table 2-1.	PROG Card Format for Assembler C	2-5

SECTION I
INTRODUCTION

GENERAL

This bulletin will help users of the Series 200/Basic Programming System change over to the Mod 1 Operating System to take advantage of more integrated and efficient methods of operation. As computer jobs get larger and more complex, it becomes more economical to integrate and automate operation — processing by jobs rather than by individual programs. This greatly reduces idle system time resulting from operator intervention during operation. By establishing automatic job control, using a deck of prearranged action directors or an internal communication scheme between program units, a job or a series of jobs, each consisting of several programs, can be processed automatically without the delays of manual control. The operating system increases throughput, makes operation faster, provides for more systematic and orderly growth, and permits multiprogramming.

In Mod 1, control and service programs are provided to perform some of the routine supervisory functions that would otherwise be performed by the operator. A user may select from the available system software whatever program preparation and maintenance, data transcription and editing, and utility programs his job requires. Then, using monitor and control programs, he processes the job automatically thereby reducing idle system time as well as chances for human error. The jobs may be arranged for either stacked job or batched processing, depending on which consideration is more important — fast turnaround time, or optimum machine utilization.

In moving from the Basic Programming System to Mod 1, there are significant changes in the operating environment. Most differences are resolved automatically by reassembly, but there are a few changes that must be made to a program written to be run in the Basic Programming System before it can be run in the Mod 1 Operating System.

SYSTEMS DESCRIPTION

Basic

The Basic Programming System is card oriented. It is designed to run on a small computer with 4K to 12K characters of memory, so it must make the best use of limited memory, foregoing resident control functions. Programs are individually assembled and independently executed. Each program is an independent entity with its own facilities for self-loading, data manipulation and specialization, and diagnostic analysis. Jobs are scheduled and executed on a demand basis. If tapes are used, they are self-loading — still unit-record oriented.

Mod 1

The Mod 1 Operating System is tape oriented and designed for systems with 12K to 65K characters of memory, three to six tape units, and perhaps a mass storage file. Programs can be operated individually as in the Basic Programming System, or in continuous semi-centralized jobs. The programs that make up the jobs, although for the most part independently generated, are joined and coordinated in the operating system.

The Mod 1 System comprises a group of related programs which handle language processing, file maintenance, loading, interrupt processing, input/output operations, and program testing. The language processing and file maintenance programs are used to create executable files and store and maintain them on binary run tapes (BRT). Programs may be called from these executable files in any sequence as they are needed in processing a job. In addition to the executable files, the Mod 1 System also maintains files of source programs and library routines, in both symbolic and machine-language form, on symbolic program tapes (SPT). Executable BRT files may be created from these SPT's, without each time going through the assembly process. To update the SPT's, only the corrections need be supplied for assembly — it is not necessary to resubmit the entire source program deck as it is in Basic.

The loader-monitor is the only program permanently in memory during a run, and as such it has a central function in the system. The communication area of the loader-monitor is used to control the sequencing of successive programs in a job. Using this communication area, a user can load and execute programs in any order he desires. Operation directors, which may be in the form of cards, operator entries, or programmed instructions, are used to control the loader-monitor and other systems programs. These directors designate which programs will be read into memory and executed and provide the parameters these programs need.

Pre-execution specialized input/output routines provide standardized and coordinated peripheral operation. They read and write files, block and unblock records, label files, and check for errors.

A subsystem of coordinated program test routines facilitates the checkout of programs by providing diagnostic information before, during, and after execution. Each of the constituent routines can be used separately or as an integrated part of the subsystem. The subsystem can process either individual programs or a job, automatically producing the documentation necessary to evaluate the programs.

Table 1-1 provides a comparison of representative programs and routines of both systems.

Table 1-1. Comparison of Systems Programs

Basic	Mod 1
Language Processing	**Language Processing**
Easycoder Assemblers A and B	Easycoder Assemblers C and D
Easycoder Assembler A(P)	COBOL Compilers D and H
Library Processor B	Fortran Compiler D
COBOL Compiler B	Library Processors C and D
	Analyzer C
Program Editing and Maintenance	**Program Editing and Maintenance**
Condense A	Update and Select C and D
Update A	SPT Merge C
	BRT Punch C
	Drum Program Store C
Operation Control	**Operation Control**
Card Loader-Monitor B	Card Loader-Monitor B
Tape Loader/Search A	Drum Bootstrap-Loader C
	Drum Monitor C
	Tape Loader-Monitor C
	Floating Tape Loader-Monitor C
	Interrupt Control D
Input/Output Control	**Input/Output Control**
Tape I/O Translator A	Standard I/O Calls C
1/2" Tape I/O A	1/2" Tape I/O C
1/2" Tape I/O B	1/2" Tape and Terminal I/O C
1/2" Tape and Terminal I/O B	Drum I/O C
Console I/O B	Console I/O C
Program Test	**Program Test**
Memory Dump A	Program Test System C
Tape Handling Routines A and B	Initializer C
	List Comments C
	Test Data Generator C
	Memory Dump Control C
	Memory Dump C
	Octal Correction C
	Tape Dump C
	Emergency Dump C
	End C

SECTION II
PROCEDURES

GENERAL

Transition from Basic to Mod 1 must be made at the source language level. All programs must be reassembled with Easycoder Assembler C (or D). If the usual good programming conventions had been observed in writing the original programs, so that they can easily be relocated, most of the differences between the two operating environments are resolved simply by reassembly. Only a few changes in some of the assembly control statements need to be made in the source program.

Mod 1 uses a resident loader-monitor and the most important considerations in the transition concern the location and use of that monitor. Programs obviously may not use memory locations reserved for the monitor, which means that many programs will have to be relocated. The memory areas reserved for the various loaders are shown in Figures 2-1 and 2-2.

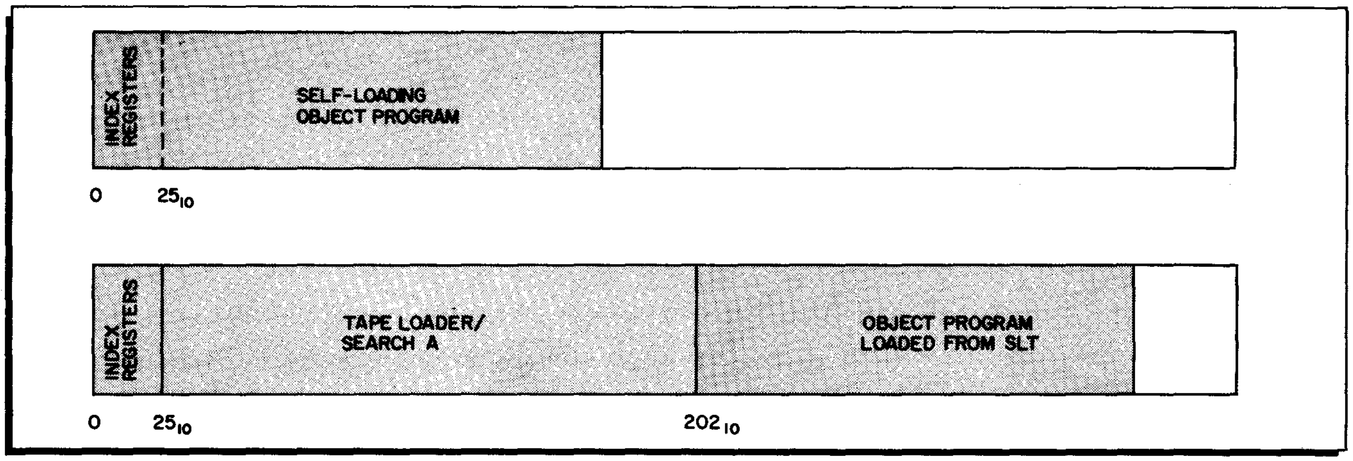

Figure 2-1. Memory Maps Showing Basic Loaders

In Mod 1, memory locations 64_{10} through 189_{10} are reserved as a communication area for the loader-monitor, and any program that uses these locations must be relocated.

Programs that do not use locations below 190_{10} can be operated under the control of Floating Tape Loader-Monitor C without relocation, provided that sufficient space is left in upper memory for the loader-monitor (1,400 to 2,750 locations, depending on the configuration). This includes any programs that are presently loaded from a self-loading tape (SLT), which do not overlay Tape Loader/Search A during execution. The Floating Tape Loader-Monitor can be assembled to reside in any portion of memory above 8K and can be relocated at execution time to the end of any 4K module except banks 0 and 1.

If, instead of Floating Tape Loader-Monitor C, one of the other loader-monitors is used, programs will have to be relocated higher. For Card Loader-Monitor B, to $1,000_{10}$ or above; for Tape Loader-Monitor C or Drum Monitor C, to 1340_{10} or above.

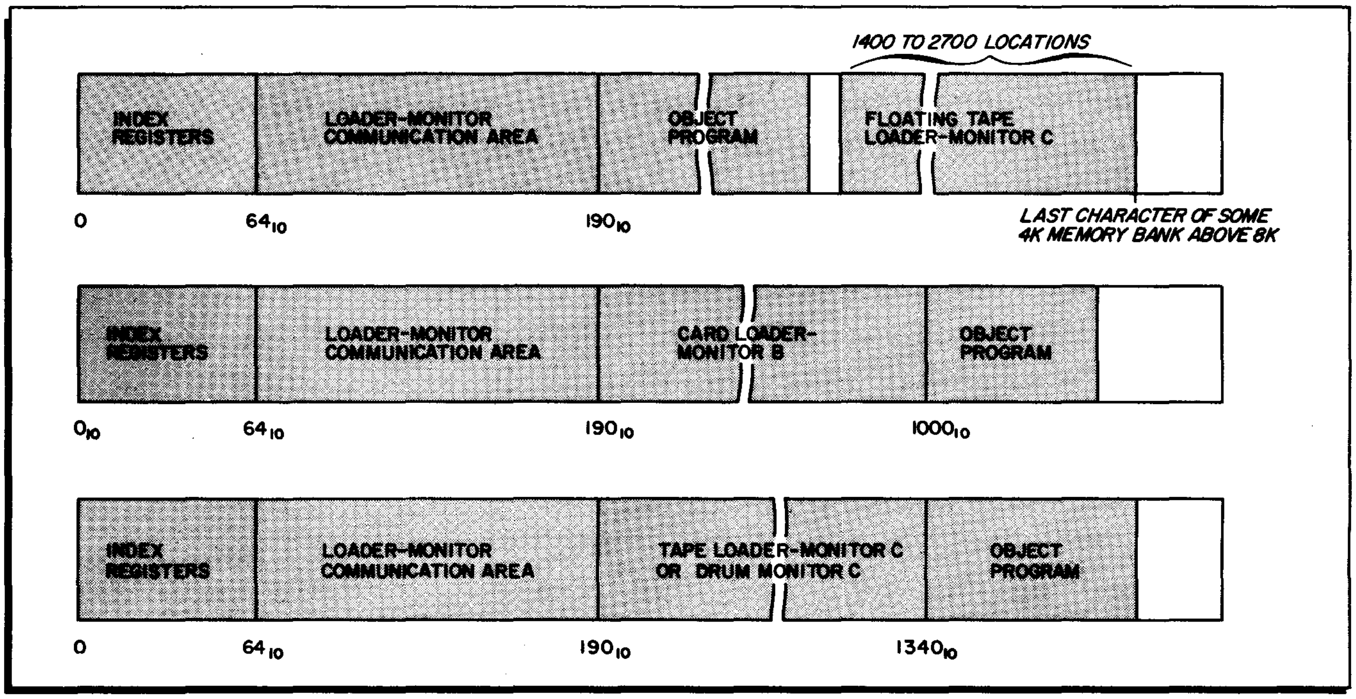
1400 TO 2700 LOCATIONS
INDEX REGISTERS
LOADER-MONITOR COMMUNICATION AREA
OBJECT PROGRAM
FLOATING TAPE LOADER-MONITOR C
0
64₁₀
190₁₀
LAST CHARACTER OF SOME 4K MEMORY BANK ABOVE 8K
INDEX REGISTERS
LOADER-MONITOR COMMUNICATION AREA
CARD LOADER-MONITOR B
OBJECT PROGRAM
0₁₀
64₁₀
190₁₀
1000₁₀
INDEX REGISTERS
LOADER-MONITOR COMMUNICATION AREA
TAPE LOADER-MONITOR C OR DRUM MONITOR C
OBJECT PROGRAM
0
64₁₀
190₁₀
1340₁₀

Figure 2-2. Memory Maps Showing Mod 1 Loaders

RELOCATION

When a program is relocated, references to absolute addresses must of course change. This is no problem if symbolic addressing has been used throughout the source coding — re-assembly will make the required adjustments. But absolute addresses must be individually changed. You should check the following areas of the coding for any absolute addressing:

1. Location fields
2. ORG and LITORG statements
3. DSA and EQU statements
4. CLEAR, EX, and END statements
5. Instruction Address fields
6. Programming techniques that assume fixed locations

To avoid the expense and trouble of hand tailoring when relocating a program, you should in your coding consistently observe the following conventions:

1. Use absolute addressing only in the first ORG Statement and when referring to permanently reserved areas like the index registers and the loader-monitor communication area. (Even here, it is a good idea to write EQU's and use symbolic tags.)
2. Do all other addressing with (or relative to) symbolic tags.
3. Avoid programming techniques that assume fixed locations or a particular address mode.

You need, then, only change one ORG statement (and possibly some ADMODE statements) to relocate the entire program. Modifying programs to conform to these conventions, if this has not been done in the past, will make them easily relocatable at any time in the future.

Address Mode

If a program that normally uses two-character addressing is relocated into a higher memory bank, it usually should be reassembled to use three-character addressing. A program may use two-character addressing only if it is kept entirely within one 4K memory bank.

When the address mode of a program is changed, any programming practices that assume two-character addressing must also change. For instance, since instruction operand lengths increase, any absolute relative addressing on instructions must be modified. Also, since the longer instructions require more memory, you may have to change some ORG or LITORG statements.

The Mod 1 Loaders all operate in three-character mode. If it is desired to still run the program in two-character mode, you must insert CAM instructions (and ADMODE statements) wherever control is transferred between the program and the loader. And when you CAM from three- to two-character mode, you must be sure that the bank bits in the A- and B-address registers are set to the proper bank. So in each segment of a two-character program, the first instruction (or if there is an initial SCR, the next instruction after it) must set the bank bits of the A- and B-address registers and be followed by a CAM to two-character mode. And the last instruction before the branch to the loader must be a CAM to three-character mode.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER (1–5)	TYPE (6)	MARK (7)	LOCATION (8–14)	OPERATION CODE (15–20)	OPERANDS (21–62)	(63–80)
				ADMODE	3	
			STARTX	SCR	EXIT,70	
				SW	* SET UP A and B REGISTER BANK BI	TS
				CAM	20	
				ADMODE	2	
				≀	≀	
				≀	≀	
				ADMODE	3	
				CAM	00	
			EXIT	B	0	
				EX	STARTX	

ASSEMBLY CONTROL STATEMENTS

Transition from Basic to Mod 1 requires that programs written to be assembled by Assemblers A or B be reassembled by Assembler C. Some of the assembly control statements for Assembler C differ from their equivalents for Assemblers A and B, and before reassembly, you must appropriately modify these few statements (see Figure 2-3).

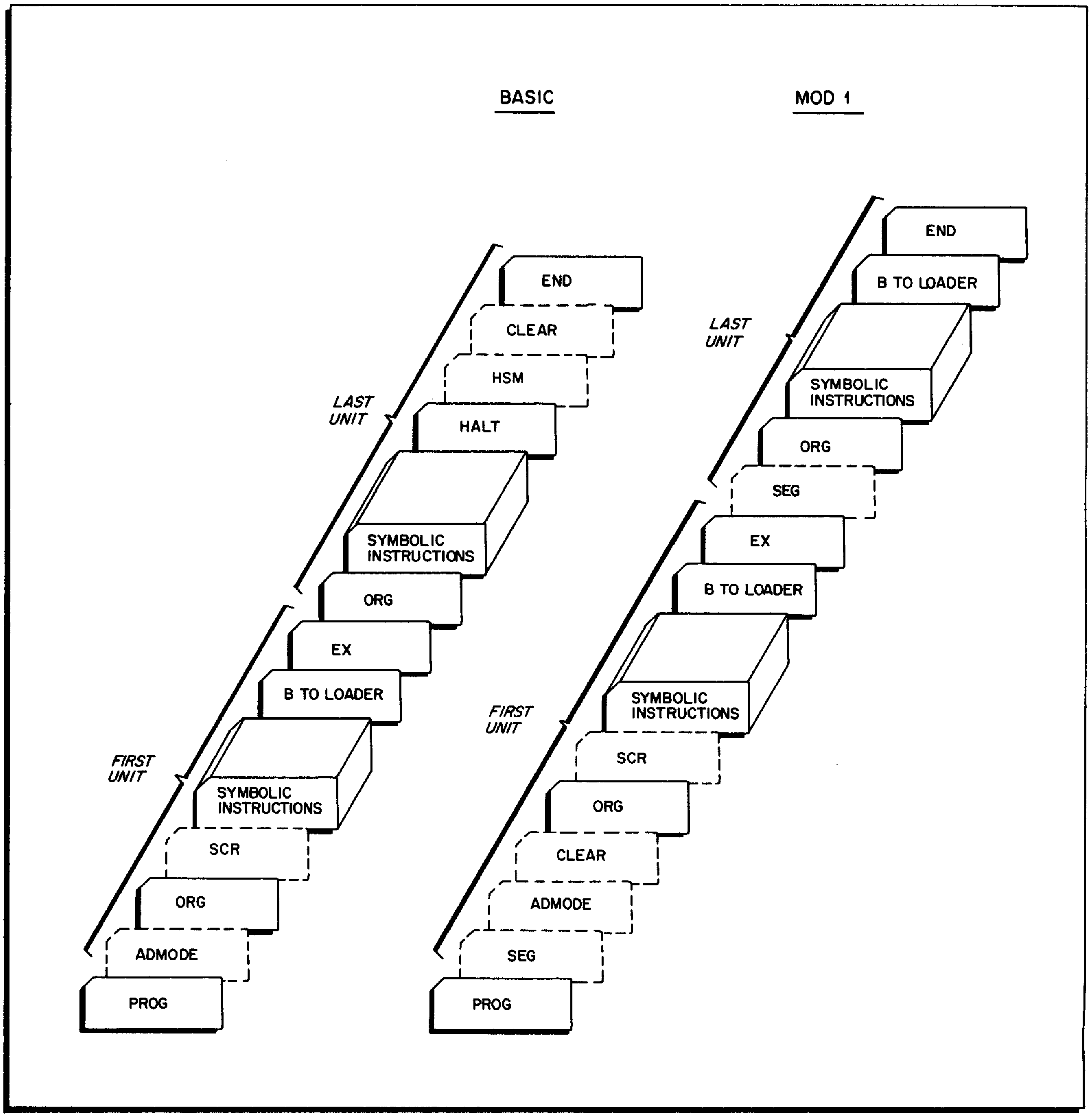

Figure 2-3. Makeup of the Symbolic Decks

Program Header - PROG

Punch a new PROG card in the Assembler C format. In Mod 1, the PROG card is used as a system action director as well as a program header. It requires additional information and is coded slightly differently (see the Easycoder Assemblers C and D Manual, Order No. 041). For the example shown in Figure 2-4, both PROG cards name the program to be assembled and ask for a sequence check of the source deck and an assembly listing.

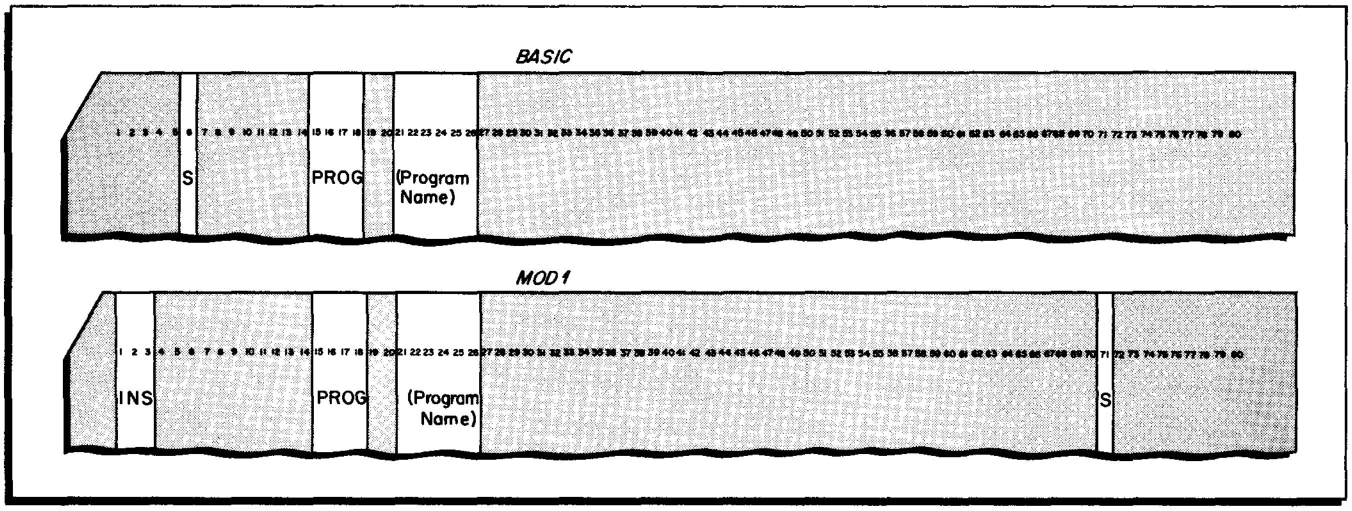

Figure 2-4. PROG Cards for Basic and Mod 1

Table 2-1. PROG Card Format for Assembler C

Columns	Contents	Explanation
1 - 3	Action Directive Code COR - Correct INS - Insert LST - List SEL - Select CPY - Copy DEL - Delete POS - Position END - End	See the Assembler C Manual for the functions of each action director.
8 - 10	Program Revision Number Three- character field that is blank or contains a number from 001 to 999_{10}.	If this field is blank, assembly will assign a revision number to the program. If the field contains a number, that number will be used as the revision number. Revision numbers are assigned in the order of updating, beginning with 001.
15 - 18	PROG	Specifies card as Program Header.

Table 2-1 (cont). PROG Card Format for Assembler C

Columns	Contents	Explanation
21 - 26	Program Name 1	Six-character program name.
28 - 33	Program Name 2	Six-character program name used with all action directors except DEL and END.
35 - 70	Visibilities	Used with all action directors except DEL and END. May contain blanks, *, or visibility codes from A-Z, 0-9 in any order.
71 - 80	Options	
71	S	Sequence check.
72	G	Generate line numbers. Can be used with INS, COR or CPY action directors.
	H	Check hash total. This field will be used as an aid to field maintenance.
73	X	Do not list program. Can be used with CPY, COR, or INS action directors.
74	L	List program. Can be used with SEL action director only.
	X	Do not write machine language on the BRT or on punched cards. Can be used with CPY, COR or INS action directors.
	C	Write machine language on punched cards instead of on the BRT. Can be used with CPY, COR, INS, or SEL action directors.
75 - 80	Not used	

Segment Header - SEG

It is not necessary to insert SEG statements — Assembler C will automatically define segments by assigning ascending numeric designations starting with 01. But if you insert any segment headers, then there must be one for every segment.

Set Address Mode -- ADMODE

Include an ADMODE statement at the beginning of the program. If the program occupies more than one 4K memory bank (programs that didn't before may now after being relocated), three-character addressing must be used. Remember that when there is no ADMODE statement, Assemblers A and B assume two-character addressing while Assembler C assumes three-character addressing.

Clear - CLEAR

Move any CLEAR cards in the symbolic program to the front of the deck, right behind the ADMODE card (see Figure 2-1). In Assemblers A and B, all CLEAR cards were placed at the end of the symbolic program and repositioned by assembly to the beginning of the program to be executed before loading. But this is not the case with Assembler C. Here a CLEAR statement is executed during loading at exactly the position it appears in the symbolic deck.

Origin - ORG

When using Floating Tape Loader-Monitor C, you must ORG all programs to location 190_{10} or above; Card Loader-Monitor B, to $1,000_{10}$ or above; Tape Loader-Monitor C or Drum Monitor C, to $1,340_{10}$ or above. All ORG statements after the first should use relative addressing so that the program can be relocated by changing only one ORG.

Check the location field of all ORG statements to be sure that there are no indented tags. Assemblers A and B treat indented tags in the location field of an ORG statement the same as left-justified tags, and assign them to the address specified in the operands field. Assembler C, however, assigns such tags to the location at which the next instruction would have begun if the ORG statement had not been present.

Literal Origin - LITORG

If any executable instructions follow a LITORG statement, you must insert an ORG back to the location following the instruction that preceded the LITORG. This is necessary because Assembler C assigns instructions that follow the LITORG differently than Assembler B. Assembler B assigns these instructions the same as if the LITORG had not been present. But Assembler C assigns subsequent instructions to locations following the literal table. So with Assembler C it is necessary to ORG back to the proper location. To do this, write an indented tag in the location field of the LITORG statement; then, immediately following the LITORG, insert an ORG to that tag.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

```
CARD NUMBER  TYPE  MARK  LOCATION  OPERATION CODE  OPERANDS
                                   ~               ~
                                   ~               ~
                         RET       LITORG          LITTAB
                                   ORG             RET
                                   ~               ~
                                   ~               ~
```

Control Equals - CEQU

Check all CEQU statements to see if any contain more than 4 characters (8 octal digits). Divide each CEQU statement that contains more than 4 characters into two CEQU statements. Then throughout the program, change each instruction that references the original CEQU so it refers also to the second CEQU. This is necessary because, although Assemblers A and B can handle 6 character (12 octal digit) CEQU's, Assembler C can handle only CEQU's of up to 4 characters (8 octal digits).

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

```
CARD NUMBER  TYPE  MARK  LOCATION  OPERATION CODE  OPERANDS
                                                               BASIC

                         INTCR     CEQU            #5CØØ41272122
                                   ~               ~
                                   ~               ~
                                   PCB             *,INTCR

                                                               MOD1

                         INTCR     CEQU            #2CØØ41
                         INTCRA    CEQU            #3C272122
                                   ~               ~
                                   ~               ~
                                   PCB             *,INTCR,INTCRA
```

Memory Dump - HSM

Remove all HSM statements. They are not used in Mod 1 and are treated as illegal statements. You may use, instead, a call to any of the more powerful and flexible program test routines that are available for Mod 1. Refer to the Program Test System C Manual, Order No. 049.

Execute - EX

Check to be sure that the operands field contains a starting address. This may be left blank (to load and halt) in Basic, but not in Mod 1. To load and halt in Mod 1, you must enter the program and segment name into the Halt Name parameter (locations 77 through 84) of the loader-monitor communication area.

The coding preceding an EX statement must of course provide a return to the loader. A method of returning to the loader that is compatible with both programming systems (the Special Call) is to incorporate, as the first instruction of the segment to be executed, an SCR instruction that moves the contents of the B-address register into the A-address field of a return Branch instruction at the end of the segment. This sets up a branch back to the instruction in the loader that follows the loader's starting branch to the segment. The loaders in both programming systems interpret this as a call for the next segment. (If the program is to run in 2-character mode, see page 2-3.)

EASYCODER
CODING FORM

PROBLEM Special Program Call PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1				STARTX	SCR	EXIT,7Ø	
2					⋮	⋮	
3					⋮	⋮	
4					⋮	⋮	
5				EXIT	B	Ø	
6					EX	STARTX	
7							
8							

However, in Basic the return to the loader may have been a branch to a read routine in the bootstrap area or, with an SLT, to location 111_{10} of Tape Loader/Search A. For Mod 1, you must change this to a special program call (above) or to a normal program call, which, after entering the name of the next segment into locations 74_{10} and 75_{10} of the loader-monitor communication area, branches to location 130_{10}.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1						BASIC	
2							
3				STARTX	〰	〰	
4					⋮	⋮	
5					⋮	⋮	
6					B	BOOT	
7					EX	STARTX	
8							

9							
10				STARTX	〰	〰	
11					⌇	⌇	
12					⌇	⌇	
13					B	111	
14					EX	STARTX	
15							
16						MOD 1	
17							
18					SEG	ØX	
19				STARTX	〰	〰	
20					⌇	⌇	
21					⌇	⌇	
22					MCW	SGNAME,75	
23					B	13Ø	
24				SGNAME	DCW	@Ø3@	
25					EX	STARTX	
26							
27							
28							
29							
30							

End - END

Check to be sure that the operands field contains a starting address. This may be left blank (to load and halt) in Basic, but not in Mod 1. To load and halt in Mod 1, you must enter the program and segment name into the halt name locations (77_{10} through 84_{10}) of the loader-monitor communication area.

As with the EX statement, the coding preceding an END statement should provide a return to the loader-monitor. In Basic, the last executable instruction of the program may have been a halt. To obtain the equivalent in Mod 1, you should replace the halt with an indirect branch to location 139_{10} of the loader-monitor communication area, which causes a console call halt. (If the program is to run in 2-character mode, see page 2-3.)

EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1						BASIC	
2				STARTL	〰	〰	
3					⌇	⌇	
4					H	A,B	
5				BOOT	END	STARTL	
6						MOD1	
7					SEG	LL	
8				STARTL	〰	〰	
9					⌇	⌇	
10					⌇	⌇	
11					B	(139) (CONSOLE CALL HALT)	
12					END	STARTL	

The Basic program, if loaded from an SLT, may have used a normal program call — moving the name of the next program into loader locations 101_{10} through 106_{10} and branching to location 86_{10}. To obtain the equivalent in Mod 1, you should move the program and segment names of the next program into locations 68_{10} through 73_{10} and 74_{10} and 75_{10} of the loader-monitor communication area, and then branch to location 130_{10}.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

```
LOCATION  OPERATION  OPERANDS
          CODE
                                    BASIC

STARTL    ~~~~       ~~~~
            ⋮          ⋮
            ⋮          ⋮
          MCW        PRNAME,1Ø6
          B          86
PRNAME    DCW        @PROCES@
BOOT      END        STARTL

                                    MOD 1

          SEG        LL
STARTL    ~~~~       ~~~~
            ⋮          ⋮
            ⋮          ⋮
          MCW        PRNAME,73
          MCW        SGNAME,75
          B          13Ø
PRNAME    DCW        @PROCES@
SGNAME    DCW        @AA@
          END        STARTL
```

The Basic program may have called for the next program on the tape by simply branching to location 111_{10}. The only way this can be done in Mod 1 is to use the special program call that was described for the EX statement. This method can be used only if the programs are arranged on the tape in exactly the order they will always be run.

An entry in the location field of the END statement, which in Basic specifies the bootstrap area, has no significance in Mod 1.

MACHINE INSTRUCTIONS

The only machine instructions that could present any problems for reassembly by Assembler C are (1) those containing area-defining literals, and (2) those with both an absolute decimal address in the location field and an asterisk (*) in the operands field.

In Basic, after any LITORG or EX statement, the tag of any area-defining literal must be redefined. But Easycoder Assembler C keeps the original assignment throughout the program, and redefining the tag will cause a duplicate symbol error. So in any instruction that contains a redefinition of an area defining literal, you should use a new symbol to define a different area.

In any instruction with an absolute decimal address in the location field, an asterisk in the A- or B-address field should be replaced with the proper decimal value. Otherwise, Assembler C, instead of assigning to the asterisk the value in the location field, will assign the location following the previous instruction.

```
ADDRESS MODE, 2-3
      SET ADDRESS MODE AND ADMODE, 2-7
ADMODE
      SET ADDRESS MODE AND ADMODE, 2-7
ASSEMBLER
      PROG CARD FORMAT FOR ASSEMBLER C, 2-5
ASSEMBLY CONTROL STATEMENTS, 2-3
BASIC, 1-1
    " LOADERS,
          MEMORY MAPS SHOWING BASIC LOADERS, 2-1
      PROG CARDS FOR BASIC AND MOD 1, 2-5
CARD FORMAT
      PROG CARD FORMAT FOR ASSEMBLER C, 2-5
CARDS
      PROG CARDS FOR BASIC AND MOD 1, 2-5
CEQU
      CONTROL EQUALS - CEQU, 2-8
CLEAR
    " - CLEAR, 2-7
COMPARISON OF SYSTEMS PROGRAMS, 1-3
CONTROL
    " EQUALS - CEQU, 2-8
    " STATEMENTS,
          ASSEMBLY CONTROL STATEMENTS, 2-3
DECKS
      SYMBOLIC DECKS,
          MAKEUP OF THE SYMBOLIC DECKS, 2-4
DESCRIPTION
      SYSTEMS DESCRIPTION, 1-1
DUMP
      MEMORY DUMP - HSM, 2-8
END
    " - END, 2-9
EQUALS
      CONTROL EQUALS - CEQU, 2-8
EX
      EXECUTE - EX, 2-9
EXECUTE - EX, 2-9
FORMAT
      PROG CARD FORMAT FOR ASSEMBLER C, 2-5
GENERAL, 1-1, 2-1
HEADER
      PROGRAM HEADER - PROG, 2-5
      SEGMENT HEADER - SEG, 2-6
HSM
      MEMORY DUMP - HSM, 2-8
INSTRUCTIONS
      MACHINE INSTRUCTIONS, 2-12
INTRODUCTION, 1-1
LITERAL ORIGIN - LITORG, 2-7
LITORG
      LITERAL ORIGIN - LITORG, 2-7
LOADERS
      BASIC LOADERS,
          MEMORY MAPS SHOWING BASIC LOADERS, 2-1
      MEMORY MAPS SHOWING MOD 1 LOADERS, 2-2
MACHINE INSTRUCTIONS, 2-12
MAKEUP OF THE SYMBOLIC DECKS, 2-4
MAPS
      MEMORY MAPS SHOWING BASIC LOADERS, 2-1
      MEMORY MAPS SHOWING MOD 1 LOADERS, 2-2
MEMORY
    " DUMP - HSM, 2-8
    " MAPS,
          MEMORY MAPS SHOWING BASIC LOADERS, 2-1
          MEMORY MAPS SHOWING MOD 1 LOADERS, 2-2
MOD
      MEMORY MAPS SHOWING MOD 1 LOADERS, 2-2
      PROG CARDS FOR BASIC AND MOD 1, 2-5
    " 1, 1-2
MODE
      ADDRESS MODE, 2-3
      SET ADDRESS MODE AND ADMODE, 2-7
ORG
      ORIGIN - ORG, 2-7
ORIGIN
    " - ORG, 2-7
      LITERAL ORIGIN - LITORG, 2-7
PROCEDURES, 2-1
PROG
    " CARD FORMAT FOR ASSEMBLER C, 2-5
    " CARDS FOR BASIC AND MOD 1, 2-5
      PROGRAM HEADER - PROG, 2-5
PROGRAM HEADER - PROG, 2-5
PROGRAMS
      SYSTEMS PROGRAMS,
          COMPARISON OF SYSTEMS PROGRAMS, 1-3
RELOCATION, 2-2
SEG
      SEGMENT HEADER - SEG, 2-6
SEGMENT HEADER - SEG, 2-6
SET ADDRESS MODE AND ADMODE, 2-7
STATEMENTS
      ASSEMBLY CONTROL STATEMENTS, 2-3
SYMBOLIC DECKS
      MAKEUP OF THE SYMBOLIC DECKS, 2-4
SYSTEMS
    " DESCRIPTION, 1-1
    " PROGRAMS,
          COMPARISON OF SYSTEMS PROGRAMS, 1-3
```

Cut Along Line

HONEYWELL EDP TECHNICAL PUBLICATIONS USERS' REMARKS FORM

TITLE:	SERIES 200 TRANSITION TO THE MOD 1 OPERATING SYSTEM SOFTWARE BULLETIN	DATED:	APRIL, 1966
		FILE NO:	122.0005.0000.0-317

ERRORS NOTED:

Fold

SUGGESTIONS FOR IMPROVEMENT:

Fold

FROM: NAME ______________________ DATE __________

COMPANY ______________________

TITLE ______________________

ADDRESS ______________________

____

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 A [illegible]g Line

Honeywell

ELECTRONIC DATA PROCESSING