X-562-68-388

PREPRINT

SUPPORT SOFTWARE
FOR THE
SPACE ELECTRONICS BRANCH ON-BOARD PROCESSOR

NOVEMBER 1968

GSFC

GODDARD SPACE FLIGHT CENTER

GREENBELT, MARYLAND

X-562-68-388

SUPPORT SOFTWARE

FOR THE

SPACE ELECTRONICS BRANCH ON-BOARD PROCESSOR

A. Merwarth
T. Taylor

Space Electronics Branch
Information Processing Division

November 1968

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

SUPPORT SOFTWARE FOR THE SPACE ELECTRONICS BRANCH ON-BOARD PROCESSOR

A. Merwarth
T. Taylor
Space Electronics Branch
Information Processing Division

ABSTRACT

This paper describes the support software package which exists for the On-Board Processor (OBP) being developed by the Space Electronics Branch, GSFC.

CONTENTS

ILLUSTRATIONS

TABLES

INTRODUCTION

The support software package described in this paper was developed for use with the On-Board Processor (OBP) - an 18 bit, stored program digital computer designed for spacecraft application. For a complete description of the OBP refer to document X-562-68-387, "A GENERAL PURPOSE ON-BOARD PROCESSOR FOR SPACECRAFT APPLICATION," dated October 1968.

The guiding philosophy for software development was to make a powerful hardware system easy to use with the aid of software. To this end a support software system has been developed which allows a programmer to use either existing English verbs, or define his own set of mnemonics. Other objectives in the development of the software system were to allow independent programming by OBP users at different locations, and by its simplicity allow efficient user program construction and debugging.

SUPPORT SOFTWARE SYSTEM

The support software system consists of an assembler, a relocatable loader, a simulator, a library of math subroutines, and a set of CPU diagnostics. The system was written in Fortran as a step towards machine independence so that OBP programs could be developed at different locations.

At present, the system has been run and checked out on SDS 920 and UNIVAC 1108 computers at GSFC, and on a GE 635 computer at General Electric Company, Valley Forge, Pennsylvania. At GSFC a special interface unit between a SDS 920 computer and an engineering model of the OBP allows object programs to be loaded into OBP memory and executed. It also permits the transfer of data between OBP memory and the SDS 920 I/O devices.

The support software system is contained on one magnetic tape. The system requires a card reader, a CPU, a line printer, and three magnetic tape drives (one for the system and two scratch). The system which was written for the SDS 920 computer contains 21 links due to core memory limitations.

The OBP assembler accepts punched cards as input, assembles program segments into relocatable binary code and data and writes the program segments on a magnetic tape. The binary segments can then be selectively loaded onto a complete image of OBP memory and written on tape. The memory image tape can be used to either load the OBP memory or to serve as input to a functional OBP simulator, a unit of the support software system.

A call to a subprogram not found in any assembled program segment will result in an automatic library search. The library now contains the following math functions: square root, sine, cosine, arc tangent, exponential, and logarithm.

Assembler

A program to be assembled must be on punched cards and must be preceded by an assemble control card. The format of the assemble control card with an explanation of the various options is covered in the section on control cards.

In order to gain a degree of self-documentation and make the generation of symbolic code as simple as possible, the OBP assembler has the following unique features:

- English language structure - a set of verb phrases have been defined rather than the usual set of three letter mnemonics.
- Flexibility - if a particular programmer prefers a certain set of mnemonics, this set may be "augmented" into the assembler very easily.
- Free form - there are no restrictions on where specific fields are aligned on the input cards. More than one machine instruction may be on one card, and text may be freely continued from one card to the next with no special indication.
- Punctuation - free use of standard English punctuation is permitted since the punctuation characters are treated as blanks. Comments within parentheses are ignored and are not restricted to one card in length.

The above features make possible the following two ways of writing OBP code:

EXAMPLE 1

```
Let X times X yield X squared.  Let Y
times Y plus X squared transformed by
square root yield Norm.  If Norm is
not greater than 1 then go to compute
inside the unit circle; otherwise....
```

EXAMPLE 2

```
LDA     X
MUL     X
STA     X2
LDA     Y
MUL     Y
ADD     X2
BRM     SQRT
STA     NORM
LET     1
BRC     COMP
```

In EXAMPLE 1 each line represents an input card, and in EXAMPLE 2 the mnemonics which have been added to the GSFC facility are used. In both cases, it is assumed that a scale register in the OBP had been set to 17 resulting in an automatic shift following each multiply.

As seen from the above examples, the assembler recognized LET as a symbol for the operation "load accumulator," the word TIMES as a symbol for the operation "multiply the operand by the accumulator, leaving the result after shifting in the accumulator," etc. Another feature of the assembler which applies independently of mnemonics is that the operand symbol may be of any length, because the assembler treats everything between two consecutively recognized operation symbols as an operand symbol.

Registers - Registers which can be affected by program execution are listed in Table 1.

TABLE 1

Central Processor Unit Functional Registers

Register	Symbol	Length (bits)	Function
Accumulator	A	18	Used for operand storage.
Extended Accumulator	EA	18	Holds least significant half of a double length operand in multiply/ divide operations.

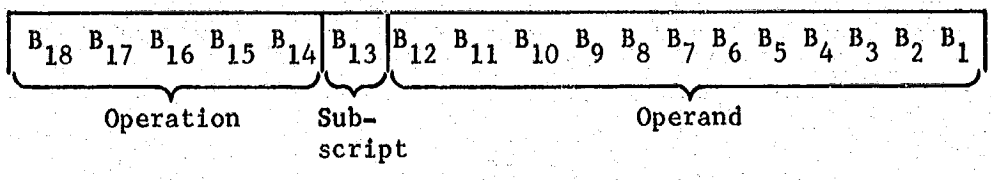
TABLE 1 (Continued)

Central Processor Unit Functional Registers

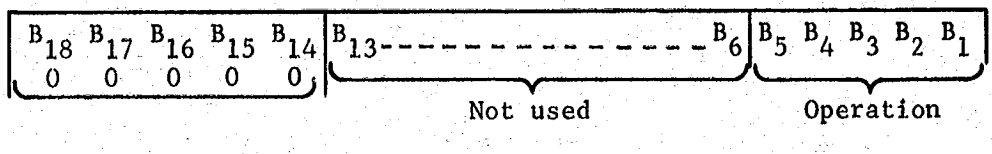
Register	Symbol	Length (bits)	Function
Storage limit	SL	18	Controls where writing into memory is permitted.
Subscript	SS	18	Added to address to form effective address if subscript bit in instruction word is set.
Scale	SC	6	Represents location of binary point for numbers.
Page	P	4	Appended to high order of 12 bit address field to form 16 bit address.
Carry	C	1	Stores a carry out of bit 17 of the parallel adder.
Decision	D	1	Is conditionally set or reset when executing test and conditional transfer instructions.
OR/AND	O/A	1	Determines whether the result of tests are to be ORed or ANDed into the D register.
Overflow	OV	1	Stores the overflow condition.

Instruction Set - The OBP has 50 instructions, 30 of which require a memory access. The other 20 instructions have a minor operation code in the operand field of the instruction word. The formats are as follows:

MEMORY ACCESS



NON-MEMORY ACCESS



With 12 bits of address available, 4,096 memory words are directly addressable. A memory size as large as 65,536 words requires a 4-bit page register which can be loaded and stored under program control and which is appended as four high-order bits to the 12-bit address field to form a full 16-bit address. If the subscript bit is set the low order 16-bits of the subscript register are added to the address to form an effective address, and the execution time is increased by 2.5 usec. The thirty instructions which require a memory access may be indexed. Indexing is specified by using either SUBSCRIPTED, or XED between the verb or mnemonic and the noun or memory location. For example, either LET SUBSCRIPTED NUMBER or LDA XED NUMBER will result in a load accumulator instruction with bit 13 set so that the subscript register will be added to the address field at execution time to form the effective address.

In the following description, the English verb and assembly language mnemonic which exist in the GSFC 920 assembler are shown.

LET noun, IF noun
LDA noun

2 0	n o u n

The content of storage at the effective address is placed in the accumulator.
Registers altered: accumulator
Timing: 6.25 usec

LET LOCATION OF noun
LDL noun

4 0	n o u n

The effective address is placed in the accumulator.
Registers altered: accumulator
Timing: 5.0 usec

YIELD noun
STA noun

6 0	n o u n

The content of the accumulator is stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs.
Registers altered: none
Timing: 7.5 usec

SET EXTENSION WITH noun
LDE noun

5 2	n o u n

The content of storage at the effective address is placed in the extended accumulator.
Registers altered: Extended accumulator
Timing: 6.25 usec

SAVE EXTENSION IN noun
STE noun

1 0	n o u n

The content of the extended accumulator is stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs.
Registers altered: none
Timing: 7.5 usec

PLUS noun
ADD noun

0 4	n o u n

The content of storage at the effective address is added to the content of the accumulator and the sum is retained in the accumulator. If a carry occurs at the input of the 18th stage of the two's complement adder, then the carry register is set to one. Otherwise, the carry register is reset to zero. Overflow can occur when two numbers of the same sign are added. Overflow causes the 18th bit of the sum to remain in the sign position and the overflow register to be set to one.
Register altered: Accumulator
Carry register
Overflow register (conditionally)
Timing: 6.25 usec

TIMES noun
MUL noun

4 4	n o u n

The content of storage at the effective address is multiplied by the content of the accumulator. The high-order 17 bits and sign of the product are retained in the accumulator. The low-order 17 bits and sign of the product are retained in the extended accumulator. The double length product is automatically scaled by arithmetically shifting the accumulator and the 17 bits of the product in the extended accumulator the number of bit positions indicated by the content of the scale register. The sign bit of the extended accumulator is not shifted. If the content of the scale register is negative, then the shift is right a and the content of the sign fills positions vacated on the left so that no overflow is possible. If the content of the scale register is positive, then the shift is to the left, with zeros filling positions vacated on the right. The overflow register is set to one if the sign bit of the accumulator is changed during the shift.

Registers altered: Accumulator
Extended accumulator
Overflow register (conditionally)

Timing: If content of scale register 17, 31.25 + (# of 1's in multiplier + /scale/) x 1.25 usec
If content of scale register 17, 31.25 + (# of 1's in multiplier + scale -15) x 1.25 usec

OVER noun, DIVIDED BY noun
DIV noun

6 4	n o u n

The content of the accumulator and extended accumulator are automatically scaled by shifting them the number of bit positions indicated by the content of the scale register. The sign of the extended accumulator is ignored and not shifted. If the content of the scale register is negative, then the shift is left with zeros filling positions vacated on the right and if the sign bit of the accumulator is changed during the shift, the overflow register is set to one. If the content of the scale register is positive, then the shift is to the right and the content of the sign fills positions vacated on the left so that overflow is impossible. The scaled accumulator and extended accumulator form the dividend that is divided by the content of storage at the effective address. The overflow register is set if the content of the accumulator is greater than or equal to the content of storage. The signed quotient is retained in the accumulator and the signed remainder is retained in the extended accumulator.

The remainder has the same sign as the dividend and has a magnitude less than the divisor.

Registers altered: Accumulator
Extended accumulator
Overflow register (conditionally)

Timing: If content of scale register 17, 87.5 + /scale/ x 1.25 usec
If content of scale register 17, 87.5 + (scale -15) x 1.25 usec
In either case, add 7.50 usec if divident 0

PLUS CARRY
ADC

0 0	0 0	0 6

The content of the carry register is added to the content of the accumulator and the sum is retained in the accumulator. If a carry occurs at the input of the 18th bit of the two's complement adder, then the carry register is set to one. Otherwise, the carry register is reset to zero. Overflow can occur and will cause the 18th bit of the sum to remain in the sign position and the overflow register to be set to one.

Registers altered: Accumulator
Carry register
Overflow register (conditionally)

Timing: 5.0 usec

NEGATED
NEG

The content of the accumulator is replaced by its two's complement. Negating all zeros yields a result of zero and sets the carry register to one. Negating the number that has zeros in all bit positions except the sign yields the same number as a result and sets both the carry register and the overflow register to one. Other than these two special cases, the carry register to reset to zero.

Registers altered: Accumulator
Carry register
Overflow register (conditionally)

Timing: 5.0 usec

ANDED WITH noun
ETR noun

3 0	n o u n

The content of storage at the effective address is anded with the content of the accumulator. The result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the

result only if the accumulator and storage both contained a one in that bit position.
Registers altered: Accumulator
Timing: 6.25 usec

ORED WITH noun
MRG noun

5 0	n o u n

The content of storage at the effective address is ORed with the content of the accumulator. The result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the result if either the accumulator or storage contained a one in that bit position.
Registers altered: Accumulator
Timing: 6.25 usec

EORED WITH noun, EXCLUSIVELY ORED WITH noun
EOR noun

7 0	n o u n

The content of storage at the effective address is exclusively ORed with the content of the accumulator. The result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the result if either the accumulator or storage, but not both, contain a one in that bit position.
Registers altered: Accumulator
Timing: 6.25 usec

COMPLEMENTED
CMP

0 0	0 0	1 0

The content of the accumulator is complemented and the result is retained in the accumulator. The 18 bits of the result are computed independently with a one occurring in a bit position of the result only if the accumulator contained a zero in that position.
Registers altered: Accumulator
Timing: 5.0 usec

SHIFTED BY noun
SHF noun

1 4	n o u n

The low-order 6 bits of the content of storage at the effective address is used as a two's complement shift count. If the count is negative, then the accumulator is shifted right the number of positions specified by the count, with the content of the accumulator sign replacing vacated positions on the left. If the count is positive,

then the accumulator is shifted left the number of positions specified by the count with zeros filling vacated positions on the right. The overflow register is set to one if the sign bit of the Accumulator is changed during the shift.

Registers altered: Accumulator
Overflow register (conditionally)

Timing: 6.25 usec + 1.25 usec per position shifted

DOUBLE SHIFTED BY noun
DSH noun

3 6	n o u n

The low-order 6 bits of the content of storage at the effective address is used as a two's complement shift count. The accumulator and the extended accumulator are shifted together. The extended accumulator is to the right of the accumulator and its sign bit is not shifted. If the count is negative, then the accumulators are shifted right the number of positions specified by the count with the content of the accumulator sign replacing vacated positions on the left. If the count is positive, then the accumulators are shifted left the number of positions specified by the count with zeros filling vacated positions on the right. The overflow register is set to one if the sign bit of the accumulator is changed during the shift.

Registers altered: Accumulator
Overflow register (conditionally)

Timing: 6.25 usec + 1.25 usec per position shifted

CYCLED BY noun
CYC noun

3 4	n o u n

The low-order 6 bits of the content of storage at the effective address is used as a two's complement shift count. If the count is negative, then the content of the accumulator is shifted cyclically right the number of positions specified by the count, with bits leaving the low-order position entering the sign position. If the count is positive, then the content of the accumulator is shifted left the number of positions specified by the count with bits leaving the sign position entering the low-order position.

Timing: 6.25 usec + 1.25 usec per position shifted

DOUBLE CYCLED BY noun
DCY noun

5 6	n o u n

The low-order 6 bits of the content of storage at the effective address is used as a two's complement shift count. If the count is negative, then the content of the accumulator and extended accumulator is shifted cyclically right the number of positions specified by the count with bits leaving the low-order position of the extended accumulator entering the sign of the accumulator and bits leaving the low-order position of the accumulator entering the sign of the extended accumulator. If the count is positive, then the content of the accumulator and extended accumulator is shifted left the number of positions specified by the count with bits leaving the sign of the entended accumulator entering the low-order position of the accumulator and bits leaving the sign position of the accumulator entering the low-order position of the extended accumulator.

Registers altered: Accumulator
Extended accumulator

Timing: 6.25 usec + 1.25 usec per position shifted

NORMALIZED
NORM

0 0	0 0	1 4

The content of the accumulator and extended accumulator is shifted left until the 17th and 18th bits of the accumulator are different. The sign bit of the extended accumulator is not shifted. Bits leaving the 17th bit of the extended accumulator enter the low-order position of the accumulator. Zeros fill the positions vacated on the right. A count of the number of positions shifted is retained as a 6-bit positive number in the scale register. If the content of the accumulator and positions 1 through 17 of the extended accumulator are zero, then the scale register is set to zero.

Registers altered: Accumulator
Extended accumulator
Scale register

Timing: 5.0 usec + 1.25 usec per position shifted or 26.25 usec if accumulator and extended accumulator are both zero.

CLOSE EXTENSION WITH DECISION
LDD

0 0	0 0	1 3

The content of the accumulator and extended accumulator is shifted left one position. The sign of the extended

accumulator is not shifted and the vacated, low-order position of the extended accumulator is filled with the content of the decision register. The overflow register is not altered.
Registers altered: Accumulator
Extended accumulator
Timing: 3.75 usec

TRANSPOSED
FLP

0 0	0 0	2 2

The contents of the accumulator are reversed. The $(19-n)^{th}$ and n^{th} bits are exchanged for n=1,2,...9.
Registers altered: Accumulator
Timing: 3.75 usec

USE SUBSCRIPT noun
LDX noun

5 4	n o u n

The content of storage at the effective address is placed in the subscript register.
Registers altered: Subscript register
Timing: 5.0 usec

SAVE SUBSCRIPT IN noun
STX noun

7 4	n o u n

The content of the subscript register is stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs.
Registers altered: none
Timing: 7.5 usec

STEP SUBSCRIPT BY noun
ADX noun

0 2	n o u n

The content of storage at the effective address is added to the content of the subscript register. The 18-bit result of the two's complement addition is retained in the subscript register.
Registers altered: Subscript register
Timing: 6.25 usec

THEN GO TO noun
BRC noun

4 2	n o u n

If the content of the decision register is zero, then the next sequential instruction is executed. If the content of the decision register is one, then the content of storage at the effective address is placed in the instruction counter and execution proceeds from the address specified by the instruction counter.
The decision register and OR/AND register are reset to zero.
Registers altered: Decision register
OR/AND register
Timing: 5.0 usec

GO TO noun, RETURN FROM noun
BRU noun

6 2	n o u n

The content of storage at the effective address is placed in the instruction counter and execution proceeds from the address specified by the instruction counter.
Registers altered: none
Timing: 5.0 usec

TRANSFORMED BY noun, PERFORM noun
BRM noun

0 6	n o u n

The content of the instruction counter plus one is stored at the effective address unless that address is protected by the storage limit registers. If storage is protected, no write into memory occurs. The content of one location greater than the effective address is placed in the instruction counter and execution proceeds from the address specified by the instruction counter.
Registers altered: none
Timing: 10.0 usec

AND

0 0	0 0	1 1

The OR/AND register is set to one.
Registers altered: OR/AND register
Timing: 3.75 usec

OR

0 0	0 0	1 5

The OR/AND register is set to zero.
Registers altered: OR/AND register
Timing: 3.75 usec

IS LESS THAN noun
ILT noun

2 6	n o u n

If the content of the accumulator is less than the content of storage at the effective address, then the test condition is set to one. Otherwise, it is zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the decision register.
Registers altered: Decision register
Timing: 6.25 usec

IS EQUAL TO noun
IET noun

4 6	n o u n

If the content of the accumulator is equal to the content of storage at the effective address then the test condition is set to one. Otherwise it is zero. The OR/AND register being zero specifies the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies the test condition is to be ANDed with the content of the decision register and the result to be retained in the decision register.
Registers altered: Decision register
Timing: 6.25 usec

IS GREATER THAN noun
IGT noun

6 6	n o u n

If the content of the accumulator is greater than the content of storage at the effective address the test condition is set to one. Otherwise, it is zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the

decision register.
Registers altered: Decision register
Timing: 6.25 usec

IF OVERFLOW
TOV

0 0	0 0	0 1

If the content of the overflow register is one, then the test condition is set to one. Otherwise, it is zero. The overflow register is reset to zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the decision register.
Registers altered: Decision register
Overflow register
Timing: 3.75 usec

IF PARITY ODD
IOP

0 0	0 0	0 5

If the number of ones in the 18-bit accumulator is odd, then the test condition is set to one. Otherwise, it is zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the decision register.
Registers altered: Decision register
Timing: 26.25 usec

IS POSITIVE
IGZ

0 0	0 0	0 3

If the sign position, bit 18, of the accumulator contains a zero, then the test condition is set to one. Otherwise, it is zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the decision register.
Registers altered: Decision register
Timing: 3.75 usec

IS EQUAL TO ZERO
IEZ

0 0	0 0	2 1

If the value of the contents of the accumulator is equal to zero, then the test condition is set to one. Otherwise, it is zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the decision register.
Registers altered: Decision register
Timing: 3.75 usec

IF SUBSCRIPT IS NOT GREATER THAN noun
XNGT noun

2 2	n o u n

If the content of the subscript register is less than or equal to the content of storage at the effective address, then the test condition is set to one. Otherwise, it is zero. The OR/AND register being zero specifies that the test condition is to be ORed with the content of the decision register and the result is to be retained in the decision register. The OR/AND register being one specifies that the test condition is to be ANDed with the content of the decision register and the result is to be retained in the decision register.
Registers altered: Decision register
Timing: 6.25 usec
8.75 usec if subscripted

IS FALSE
CPD

0 0	0 0	1 7

The Decision register is complemented
Registers altered: Decision register
Timing: 3.75 usec

PASS
NOP

0 0	0 0	0 2

No operation is performed other than the automatic incrementing of the instruction counter.
Registers altered: none
Timing: 3.75 usec

HALT
HLT

0 0	0 0	0 0

The processor stops indefinitely. An initiate signal must be supplied from an external source to start the processor.
Registers altered: none
Timing: none

EXECUTE noun
EXU noun

1 2	n o u n

The content of storage at the effective address is used as the address of the instruction to be executed. The instruction counter is incremented by one unless it is changed by the execution of a transfer-type instruction. If the machine attempts to execute an EXECUTE noun, the program proceeds with no operation being performed.
Registers altered: those associated with the instruction executed.
Timing: 5.0 usec + executed instruction timing.

SET SCALE WITH noun
LDS noun

3 2	n o u n

The low-order 6 bits of the content of storage at the effective address is placed in the scale register.
Registers altered: Scale register
Timing: 5.0 usec

LET SCALE
SSA

0 0	0 0 2 0

The content of the scale register is placed in the low-order 6 bits of the accumulator. The high-order 12 bits of the accumulator are set to zero.
Registers altered: Accumulator
Timing: 3.75 usec

SET PAGE
LDP

0 0	0 0	1 2

The content of bits 13 through 16 of the accumulator are placed in the page register.
Registers altered: Page register
Timing: 3.75 usec.

RESET OVERFLOW
ROV

0 0	0 0	0 7

The content of the overflow register is set to zero.
Registers altered: Overflow register
Timing: 3.75 usec

RESET DECISION
RED

0 0	0 0	2 3

The contents of the decision register is set to zero.
Registers altered: Decision register
Timing: 3.75 usec

EXIT

0 0	0 0	1 6

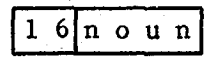
This instruction initiates interrupt number 16 and uses locations octal 200 through 207. Upon completion, execution proceeds normally using the new value in the instruction counter.
Registers altered: Limit register
Interrupt priority register
Page register
OR/AND register
Overflow register
Carry register
Decision register
Scale register
Timing: 35.0 usec

RESUME FROM noun
TIN noun

7 2	n o u n

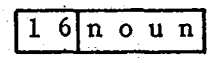
The content of storage at the effective address is used as the starting address of an interrupt storage area. This instruction reloads the registers that were saved at the occurrence of an interrupt. Upon completion, execution proceeds normally using the new value in the instruction counter.
Registers altered: Limit Register
Interrupt priority register
Page register
OR/AND register
Overflow register
Carry register
Decision register
Scale register
Timing: 21.25 usec

CONNECT TO noun
IO noun



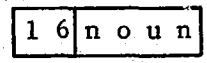
The content of storage at the effective address is sent to the I/O unit as a command. This word must have a zero in bits 17 and 18 to denote the establishment of a cycle steal channel. Bits 1 through 16 specify the data channel and block length. The content of the accumulator is stored at location seven. The content of location seven is then output as a starting memory address to the I/O unit.
Registers altered: none
Timing: 11.25 usec

LET FUNCTION TO noun
IO noun



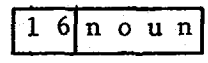
The content of storage at the effective address is sent to the I/O units as a command. This word must have a zero in bit 18 and a one in bit 17 to denote that bits 1 through 16 are a function code.
Registers altered: none
Timing: 5.0 usec

OUTPUT TO noun
IO noun



The content of storage at the effective address is sent to the I/O unit as a command. This word must have a one in bit 18 and a zero in bit 17. Bits 1 through 16 denote the data channel for an output device. The content of the accumulator is stored at location seven. The content of location seven is then output as data to the I/O unit.
Registers altered: none
Timing: 11.25 usec

LET INPUT FROM noun
IO noun



The content of storage at the effective address is sent to the I/O unit as a command. This word must have a one in bits 17 and 18. Bits 1 through 16 denote the data channel for an input device. The I/O unit stores one word of data at location seven. The content of location seven is then placed in the accumulator
Registers altered: Accumulator
Timing: 11.25 usec

Assembler Directives

Assembler directives are used to pass information to the assembler concerning a particular program to be assembled. The assembler directives are loaded in with the source program and enjoy the same freedom from fixed card format as does the symbolic program statements. These directives have affect only for the program with which they are assembled and their affect begins when they are encountered during the assembly process. For convenience, the assembler directives can be delineated into four categories according to their usage or function in the program. These categories are:

a. Control of storage allocation

- START THE FOLLOWING INSTRUCTION AT LOCATION_____
- START THE FOLLOWING DATA AT LOCATION_____
- ASSIGN_____LOCATIONS FOR_____
- ALLOCATE_____LOCATIONS FOR_____
- END OF THIS PROGRAM SEGMENT

b. Data Word Generation

- PREDEFINE_____AS THE VALUE_____
- PRESET_____VALUES OF_____TO THE CONSTANTS_____
- THE RADIX FOR NUMBERS IS DECIMAL
- THE RADIX FOR NUMBERS IS OCTAL
- THE SCALE FACTOR FOR NUMBERS IS_____
- THE SCALE FOR NUMBERS IS_____
- _____OCTAL_____
- _____DECIMAL_____
- _____SCALED_____ _____

c. Label Definition and Subscripting

- DEFINING_____
- DEFINING FOR EXTERNAL USE

- SUBSCRIPTED

d. Connectives

- LET IT
- IF IT
- BE
- OTHERWISE
- AND ALSO

Control of Storage Allocation -

START THE FOLLOWING INSTRUCTIONS AT LOCATION number
START THE FOLLOWING INSTRUCTIONS AT LOCATION noun

This directive allows the user to alter the assembler's automatic assignment of storage by specifying the origin of the following group of instructions. This allows specific arrangements of various sequences of instructions both in relation to themselves and to the memory capacity available to the particular program. The origin is the location in memory from which the sequence of instructions will be executed. If number is used, then the origin will be the absolute memory address equal to the number specified. If noun is used, then the origin will be absolute provided the noun in question has been previsouly assigned a numeric value. Otherwise, the origin is subject to relocation by the loader. Once this directive has been encountered, the verbs are assembled as having addresses beginning with the value of noun or number and increasing by one for each succeeding instruction until either an END OF THIS PROGRAM SEGMENT or another START THE FOLLOWING INSTRUCTIONS AT LOCATION is encountered.

Restrictions: number should be positive and smaller than the memory size. It is the user's responsibility to be sure that the same absolute locations are used only once.
noun should have been previously used with the directive DEFINING noun or with the directive ASSIGN noun TO LOCATION number.
The storage allocation of relocatable sequences of instructions and relocatable sequences of data words in independent during assembly and loading. No distinction

is made during loading between instructions and data words with absolute addresses.

Examples:
START THE FOLLOWING INSTRUCTIONS AT LOCATION 4095
START THE FOLLOWING INSTRUCTIONS AT LOCATION OCTAL 3777
START THE FOLLOWING INSTRUCTIONS AT LOCATION LABEL 2

START THE FOLLOWING DATA AT LOCATION number
START THE FOLLOWING DATA AT LOCATION noun

This directive allows the user to alter the assembler's automatic assignment of storage by specifying the origin of the succeeding data addresses. This allows specific arrangements of various sequences of data words or data addresses in relation to themselves and to the memory capacity available to the particular program. The origin is the location in memory where data words or data addresses will begin at execution time. If number is used, then the origin will be the absolute memory address equal to the number specified. If noun is used, then the origin will be absolute provided the noun in question has been previously assigned a numeric value. Otherwise the origin is subject to relocation by the loader. Once this directive has been encountered, data words generated by PREDEFINE and PRESET and nouns specified in ALLOCATE and ASSIGN noun will be assigned data addresses beginning with the value of noun or number and increased by one for each succeeding address until either an END OF THIS PROGRAM SEGMENT or another START THE FOLLOWING DATA AT LOCATION _____ is encountered.

Restrictions:
number must be positive and smaller than 4096. The relocatable data origin of the support software loader may be altered by the user to cause relocatable data to be loaded anywhere in the On-Board Processor's memory.
noun should have been previously used in an ASSIGN, ALLOCATED, PREDEFINE, or PRESET assembler directive.
The storage allocation of relocatable sequences of instructions and relocatable sequence of data words is independent during assembly and loading. No distinction is made during loading between instructions and data words with absolute addresses.

Certain data addresses are not assigned until the END OF THIS PROGRAM SEGMENT directive is encountered.
These include: the data address for constants used as the object of machine verbs; the data address of the indirect address for the noun used as the object of GO TO, THEN GO TO, TRANSFORMED BY, PERFORM, and RETURN FROM machine verbs; the data address of nouns used as the objects of machine verbs that did not appear in ASSIGN, ALLOCATE, PREDEFINE, or PRESET directives.

Examples:
START THE FOLLOWING DATA AT LOCATION 3999
START THE FOLLOWING DATA AT LOCATION OCTAL 100
START THE FOLLOWING DATA AT LOCATION MATRIX

ASSIGN noun 1
ASSIGN noun 1 TO LOCATION number
ASSIGN noun 1 TO LOCATION noun 2

This assembler directive provides a method of assigning nouns which represent data to specific memory locations. It is not necessary to assign all nouns storage locations because the assembler will automatically assign one location for each noun that is used only as the object of a machine verb. The first case causes noun 1 to be assigned the next available data location and causes the data location counter to be incremented by one. The second case causes noun 1 to be assigned the absolute memory address specified by number. The data location counter is not changed. The third case causes noun 1 to be assigned to this same memory address as noun 2. If noun 2 has previously been assigned an address then the data location counter is not incremented. If noun 2 is unassigned when this directive is encountered, then both noun 1 and noun 2 are assigned the next available data location and the data location counter is incremented by one.

Restrictions: Multiple assignment of noun 1 is allowed but only the last assignment is used as the address to be placed in machine instructions.
number must be positive and less than 4096.
When the next available data location is used, the noun is relocatable or absolute according to the most recent data origin that has been established.

Neither noun 1 nor noun 2 may be used as the object of GO TO, THEN GO TO, TRANSFORMED BY, PERFORM or RETURN FROM machine verbs. Nouns for these verbs refer indirectly to a position in the instruction sequence and must be defined by the DEFINING or DEFINING FOR EXTERNAL USE assembler directives.

Examples:
```
ASSIGN ALPHA
ASSIGN L TO LOCATION LONGITUDE
ASSIGN COMMON DATA TO LOCATION OCTAL 7000
```

ALLOCATE number LOCATIONS FOR noun

This directive is used to reserve data storage for a vector or array whose name is designated by noun. The number of locations reserved for the noun vector is specified by number. The noun is assigned the next available data location when this directive is encountered. The data location counter is then incremented by number. In a program, noun will usually appear preceded by SUBSCRIPTED. During execution the subscript register would typically contain a value between 0 and number -1.

Restrictions: noun must not have appeared previsouly in an ASSIGN, PRESET, or PREDEFINE assembler directive. However, noun may appear in a subsequent PRESET or PREDEFINE assembler directive and retain the address assigned when allocated. If noun appears in a subsequent ASSIGN statement, its address will be changed.

Examples:
```
ALLOCATE 100 LOCATIONS FOR TEN BY TEN
MATRIX
ALLOCATE OCTAL 77 LOCATIONS FOR TABLE
```

END OF THIS PROGRAM SEGMENT

This directive is used to inform the assembler of the end of a program. Upon encountering this directive, each machine instruction has been assigned either a relocatable or absolute instruction address. After this directive is encountered, nouns that have not appeared in ASSIGN, PREDEFINE, PRESET, or ALLOCATE assembler directives and constants used with machine verbs will be assigned data storage.

Restrictions: This directive must appear physically as the last sentence of input to the assembler. The next input card is expected to be a control card with a semicolon in column one.

Data Word Generation -

PREDEFINE noun AS THE VALUE literal

This assembler directive may be used to initialize a data location to a fixed constant. If noun has previously been assigned a data address via a PREDEFINE, PRESET, ALLOCATE, or ASSIGN directive, then the literal will be loaded into memory at the assigned address. Otherwise, noun is assigned the next available data location and the data location counter is incremented by one. The literal will be loaded into memory at the assigned address reserved for noun. The value of noun may be modified during execution, by machine instructions that stored into memory.

Restrictions: The memory address where the literal is initialized is not changed if noun is subsequently assigned another data location. If the same memory location is initialized more than once, then the last literal value will be in memory when the program is executed.
The literal constant will be converted to an 18-bit, two's complement data word. If the literal contains a decimal point and the radix is decimal, then the literal is multiplied by the scale factor for numbers. Otherwise the literal is taken as it appears. In both cases only the integral part of the result is used and it must be less than 2^{17} in magnitude.

Examples:
```
PREDEFINE COUNT AS THE VALUE 10
PREDEFINE APERATURE AS THE VALUE -27.3127
PREDEFINE MASK AS THE VALUE OCTAL 770077
PREDEFINE PI AS THE VALUE SCALED 32768
3.1416
```

PRESET number VALUES OF noun TO THE CONSTANTS literals

This assembler directive may be used to initialize successive data locations to a set of fixed constants. If noun has been previously assigned a data address via

a PREDEFINE, PRESET, ALLOCATE, or ASSIGN directive, then the first literal will be loaded at the assigned address and the data location counter is unchanged. Otherwise noun is assigned the next available data location. Storage is allocated for number locations by incrementing the data location counter by number. The sequence of literals will be loaded into successive memory locations starting at the address assigned to noun. In a program, noun will usually appear preceded by SUBSCRIPTED. During execution the subscript register would typically contain a value between 0 and number -1. The preset values may be modified during execution by machine instructions that store into memory.

Restrictions: The memory addresses where the sequence of literals is initialized is not changed if noun is subsequently assigned another data location. If the same memory location is initialized more than once, then the last literal value will be in memory when the program is executed.
The literal constants will be converted to 18-bit, two's complement data words. Each literal that contains a decimal point and has a decimal radix is multiplied by the scale factor for numbers. Other literals are used as they appear. In both cases, only the integral part of the result is used and it must be less than 2^{17} in magnitude.
number must be equal to the number of constants. If noun was previsouly assigned a data location, then no check is made to determine if the number of constants is larger than the storage allocated to noun.

Examples: PRESET 5 VALUES OF TABLE -1 TO THE CONSTANTS 10, -27.3127, OCTAL 770077, SCALED 32768, 99

THE RADIX FOR NUMBERS IS DECIMAL

This assembler directive is used in conjunction with the directive THE RADIX FOR NUMBERS IS OCTAL, and directs the assembler to use 10 as the base or radix in the conversion of numbers appearing in the program to their binary equivalents. This base is used from the occurrence of this directive until either the directive END OF THIS PROGRAM SEGMENT or THE RADIX FOR NUMBERS IS OCTAL appears. The

assembler will interpret all numbers appearing in the program as decimal numbers if no radix directive is used.

THE RADIX FOR NUMBERS IS OCTAL

This assembler directive is provided for those situations in which a large number of octal constants are used. It directs the assembler to use eight as the base or radix in the conversion of numbers appearing in the program to their binary equivalents. This base is used from the occurrence of this directive until either the directive END OF THIS PROGRAM SEGMENT or THE RADIX FOR NUMBERS IS DECIMAL appears. The assembler will interpret all numbers appearing in the program as decimal numbers if no radix directive is used.

Restrictions: Numbers converted using octal radix may not contain a radix point and must contain only the digits 0 through 7.

THE SCALE FACTOR FOR NUMBER IS number

This assembler directive changes the scale factor that the assembler uses for converting numbers, which contain a decimal point, to binary. The whole and fractional part of a number containing a decimal point are multiplied by the current assembler scale factor, number. The assembler uses a scale factor of 1.0 if no scale factor directive occurs. The scale factor number is used until either the directive THE SCALE FACTOR FOR NUMBERS IS or END OF THIS PROGRAM SEGMENT appears.

Restrictions: number is always converted to binary with a scale factor of 1.0.
Octal numbers that contain a radix point will not be scaled.

Example: THE SCALE FACTOR FOR NUMBERS IS 1000.
LET .307 (METERS) YIELD MILLIMETERS PER FOOT.

THE SCALE FOR NUMBERS IS number

This assembler directive accomplishes the same function as the previous directive. The difference is that the scale factor for converting numbers is computed as 2 exp (number - 17); i.e., numbers will be scaled for the scale register setting specified by number.

verb OCTAL number

This assembler directive modifies the base or radix to eight for converting number to binary. This directive supersedes the directive THE RADIX FOR NUMBERS IS DECIMAL for conversion of this one number. The verb may be most machine verbs and most assembler directives. The form OCTAL number may be used anywhere number could be used in the assembly language.

Restrictions: number may not contain a radix point and must contain only the digits 0 through 7.

Example: LET OCTAL 37777 PLUS OCTAL -101

verb DECIMAL number

This assembler directive modifies the base or radix to 10 for converting number to binary. This directive supersedes the directive THE RADIX FOR NUMBERS IS OCTAL for conversion of this one number. The verb may be most machine verbs and most assembler directives. The form DECIMAL number may be used anywhere number could be used in the assembly language.

Restrictions: If the directive THE RADIX FOR NUMBERS IS OCTAL has not been used, then the radix will automatically be decimal.

Example: LET DECIMAL 99 TIMES DECIMAL 3.14.

verb SCALED number literal

This assembler directive causes number to be used as the scale factor for converting literal to binary. The whole and fractional part of literal are multiplied by number. The integer part of the product is the resulting binary number. This directive supersedes the directive THE SCALE FACTOR FOR NUMBERS IS for conversion of this one literal.

Restrictions: number is always converted with a scale factor of 1.0.
literal must contain a decimal point and the current radix must be decimal.

Examples: LET SCALED 1000 .297
PLUS DECIMAL SCALED 100 47.25

Label Definition and Subscripting -

DEFINING noun

This assembler directive is used to define a position in a sequence of machine instructions. The noun should appear somewhere in this program segment as the object of a GO TO, THEN GO TO, RETURN FROM, TRANSFORMED BY, PERFORM, or EXECUTE machine verb. A data location will be assigned for noun when the END OF THIS PROGRAM SEGMENT directive is encountered. This data location will be preset with a value equal to the instruction address where the DEFINING directive occurred.

Restrictions: The only other assembler directive that noun may occur with is START THE FOLLOWING INSTRUCTIONS AT LOCATION noun.
When using noun as the object of LET-, PLUS-, and YIELD-type machine verbs, the data location referred to by noun is expected to contain an instruction address.
If noun is used as the object of TRANSFORMED BY or PERFORM machine verbs, then two successive locations are preset to the instruction address where the DEFINING directive occurred.

DEFINING FOR EXTERNAL USE noun

This assembler directive is used to define a position in a sequence of machine instructions that may be referred to by other separately assembled programs or subroutines. The noun may appear as the object of GO TO, THEN GO TO, RETURN FROM, TRANSFORMED BY, PERFORM, or EXECUTE machine verbs in this program and other programs. A data location will be assigned for noun when the END OF THIS PROGRAM SEGMENT directive is encountered. Two data locations are reserved and the second is preset with a value equal to the instruction address where the DEFINING FOR EXTERNAL USE directive occurred.

Restrictions: The only other assembler directive that noun may occur with is START THE FOLLOWING INSTRUCTIONS AT LOCATION noun.
noun may be used as the object of LET-, PLUS-, and YIELD-type machine verbs only in the program where it is defined. The data location referred to by noun is expected to contain an instruction address.

verb SUBSCRIPTED noun

This directive is used between a machine verb and its object noun. Bit position 13 of the machine instruction is set to 1. When the machine instruction is exe uted, the data address referred to in memory is that of noun plus the content of the subscript register.

Restrictions: verb must be a machine verb that requires a noun.
noun will normally appear in an ALLOCATE, PRESET, or ASSIGN assembler directive.

Example: LET SUBSCRIPTED VECTOR TIMES SUBSCRIPTED VECTOR YIELD PARTIAL DOT PRODUCT

Connectives -

Connectives do not generate machine instructions or assembler directives but function only as a means of making the assembly language text more readable, while preserving the verb-noun syntactic pattern. Upon recognition of a connective, the assembler continues with the processing of the next syntactic element. The following is a list of the connectives:

- LET IT
- IF IT
- BE
- OTHERWISE
- AND ALSO

Connectives may be used anywhere a verb may be used. They may not be used as nouns.

Example:
```
LET A PLUS B YIELD C.  LET IT PLUS D YIELD E.
IF IT IS LESS THAN 5 THEN GO TO ALPHA.
OTHERWISE GO TO BETA.
SET EXTENSION WITH X2 AND ALSO LET X1 BE.
RETURN FROM THIS ROUTINE.
```

Relocatable Loader

The assembler produces relocatable code and data except when it encounters a directive, such as START THE FOLLOWING DATA AT LOCATION

number, which uniquely specifies where this code or data will be located. This means that code or data addresses are relative to the beginning code or data address assigned by the loader such that programs and data sets will be automatically stacked in core without overlap and without unused storage locations. The beginning bias for data and code is presently 210 and 4000 octal respectively, where locations 0 to 210 are used for input/output and interrupt storage, and 4000 octal is the mid-point of a 4K memory module. As will be seen in the control card description, these starting biases may be altered. Figure 1 and Figure 2 illustrate the assembly of a program to calculate the combinations of n things taken k at a time and how the assembled program and data would be loaded into memory for execution. This example shows two assemblies - a main calling program and a subroutine which returns n! Figure 1 lists the two programs to be assembled, COMBINATIONS and FACTORIAL. Figure 2 indicates how these two programs would be loaded into OBP memory where the data for COMBINATIONS occupies locations 210 through 221 and the data for FACTORIAL is between location 222 and 234, all numbers being in octal. The code for COMBINATIONS and FACTORIAL are loaded into locations 4000 - 4023 and 4024 - 4046 respectively. Another feature of the loader is that a binary tape can be built up which contains many assembled program segments and certain of these assemblies can be selectively loaded.

Simulator

The functional simulator reads the absolute memory image tape created by the loader into core and simulates the execution of that program. When a HALT is encountered, the simulator prints out statistics concerning simulated running time and frequency of instruction usage. By means of various control cards, the simulator may be made to give selective tracing and/or dumping in either an octal or decimal mode. Control cards are also available for specifying periodic interrupts, simulation of input from the I/O unit, and such miscellaneous capabilities as halts treated as no-ops and restarting after a halt has been executed.

Interrupt Simulation - At the completion of each simulated instruction, the interrupt processor senses if any of the 15 external interrupts appeared during the simulation of the previous instruction. When an interrupt appears, the comment: INTERRUPT n HAS APPEARED AT time is written on the printer. If the interrupt cannot be honored immediately, it is saved and the word SAVED is printed on the same line as the above comment. If an interrupt is currently being saved and another of the same number appears, it is lost and the word LOST is printed. When an interrupt is honored, the comment: INTERRUPT n HONORED AT time is printed.

Input/Output Simulation - When an I/O instruction is simulated, the content of storage at the effective address is sent to the simulated I/O unit as a command. The simulated I/O unit interprets bits 17 and 18 of this function word to determine the function code and interprets bits 1 to 16 to identify the I/O device.

```
;  ASSEMBLE COMBINATIONS PRINT

ASSEMBLER FOR THE ON-BOARD PROCESSOR
STANDARD VERBS AVAILABLE

                    THE RADIX FOR NUMBERS IS OCTAL
                    PREDEFINE NUMBER OF COMBINATIONS AS THE VALUE 400001
                    PREDEFINE NUMBER OF ITEMS PER GROUP AS THE VALUE 600001
                    PREDEFINE NUMBER OF ITEMS AS THE VALUE 500001
                    THE RADIX FOR NUMBER IS DECIMAL.
                    SET SCALE WITH 17.
                    LET INPUT FROM NUMBER OF ITEMS YIELD N.  LET N TRANSFORMED BY
                    FACTORIAL YIELD N FACTORIAL.  LET INPUT FROM NUMBER OF ITEMS
                    PER GROUP YIELD M.  LET M TRANSFORMED BY FACTORIAL YIELD
                    M FACTORIAL.  LET N MINUS M TRANSFORMED BY FACTORIAL TIMES
                    M FACTORIAL YIELD DENOMINATOR.  LET N FACTORIAL OVER DENOMINATOR
                    BE OUTPUT TO NUMBER OF COMBINATIONS.
                    HALT
                    END OF THIS PROGRAM SEGMENT

;  ASSEMBLE FACTORIAL PRINT

ASSEMBLER FOR THE ON-BOARD PROCESSOR
STANDARD VERBS AVAILABLE

                    DEFINING FOR EXTERNAL USE FACTORIAL.  IF IT IS LESS THAN Ø THEN GO TO
                    (RETURN FROM) FACTORIAL.  IF IT IS EQUAL TO Ø THEN GO TO ANSWER EQUALS ONE:
                    OTHERWISE LET IT YIELD N AND ALSO YIELD RESULT.
                    DEFINING JAIL.  LET N MINUS 1 YIELD N.  IF IT IS LESS THAN 2 THEN GO TO
                    ANSWER;  OTHERWISE, LET RESULT TIMES N YIELD  RESULT.  GO TO JAIL.
                    DEFINING ANSWER.  LET RESULT BE.  RETURN FROM FACTORIAL.
                    DEFINING ANSWER EQUALS ONE, LET 1 BE.  RETURN FROM FACTORIAL.
                    END OF THIS PROGRAM SEGMENT.
```

Figure 1. Computer Listing of Main Program (COMBINATIONS) and Subroutine (FACTORIAL).

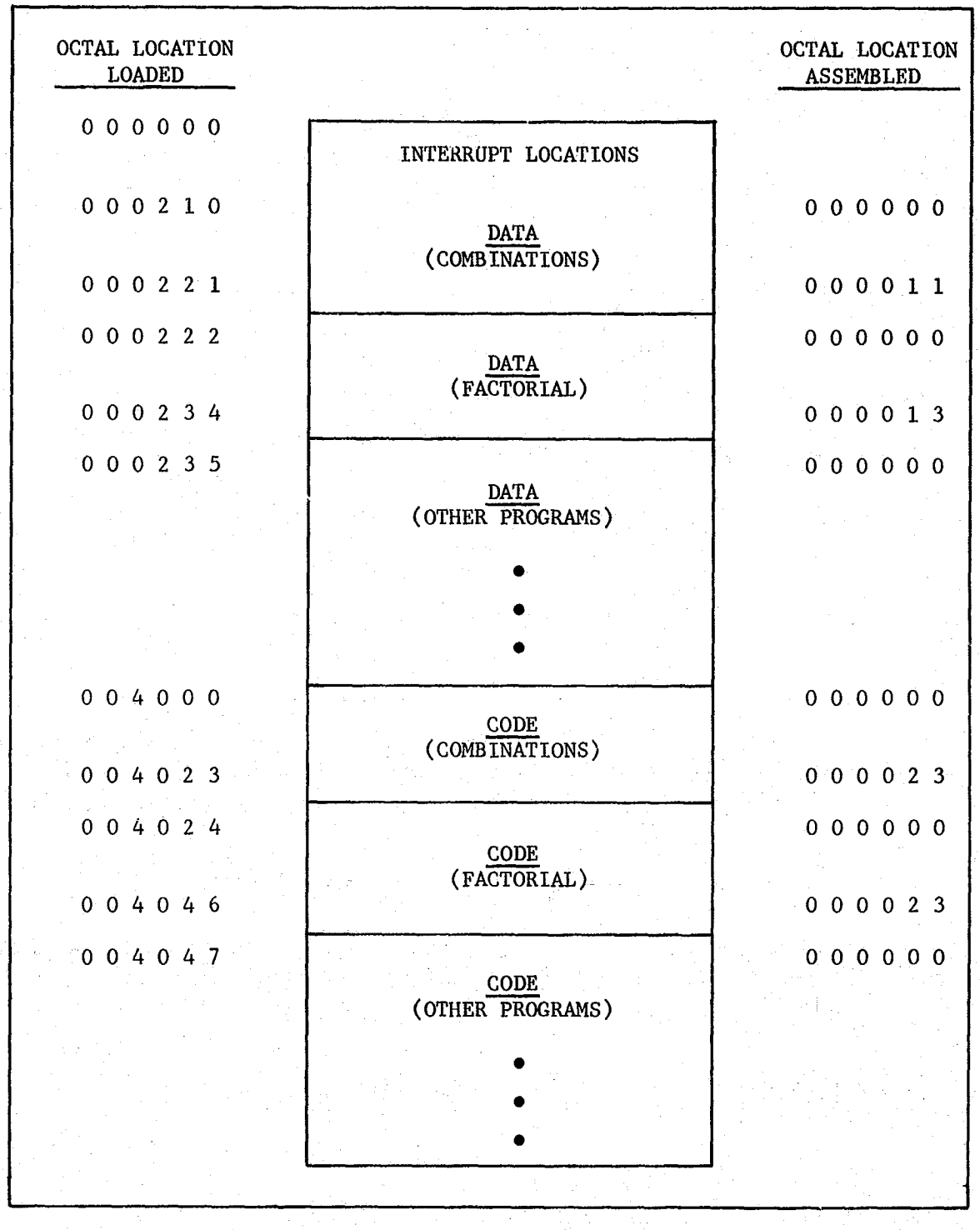


Figure 2. Core Image Diagram of Automatic Code and Data Relocation

When the I/O instruction is simulated, the OBP simulator writes the following comment on the printer:

**** contents of effective address IS SENT TO I/O UNIT.

When the I/O simulator interprets the function word and successfully carries out the operation, it continues this comment by writing one of the following phrases on the same line.

Function Code	Phrase
0	No phrase
1	No phrase
2	THE DATA WORD contents of the accumulator IS OUTPUT TO I/O DEVICE Bits 1 to 16 of function word.
3	THE DATA WORD next available data word for the designated I/O device IS INPUT FROM I/O DEVICE Bits 1 to 16 of function word.

NOTE

The current simulation time is also printed with the above comments. It should be noted that the I/O comment is printed regardless of whether or not the simulation is being traced.

Function codes 0, 1, or 2 may specify an I/O device identification ranging from 0 to $2^{16}-1$. However, the I/O device identification is limited to 0, 1, 2, or 3 for function code 3 for simulation purposes. If any other device identification is specified, the comment

ILLEGAL I/O DEVICE SPECIFIED

will be written on the printer and the I/O operation will be considered complete with no further action.

If function code 3 specifies a legal I/O device, say n, which has no more data, the comment

OUT OF DATA FOR I/O DEVICE n

will be written on the printer and the I/O operation will be considered complete with no further action. When the input/output simulation is completed, the next sequential instruction is simulated.

Description of Dumps and Traces - A dump is a printout of the contents of memory and generally comprises two parts: data and code. By means of the several simulator control cards, the user can control the dump to suit his needs. A dump may be decimal or octal; the entire memory may be dumped or just a specific segment. A dump may be printed upon simulation of a HALT instruction or at any specific point in the program execution. Likewise a trace, the printout of register contents during execution, has almost as many optional forms as the dump and is also user controlled to suit specific needs. Examples of dumps and traces with their various columns explained are discussed in the following paragraphs.

Decimal Dump - A decimal dump is a printout of memory within specified limits at a particular time (refer to Figure 3). The first column on the left is the decimal address. The entire dump is broken into several sections, the first of which is always a printout of the content of the interrupt locations. The remainder of the dump comprises a scaled decimal printout of the content of memory at the data locations occupied by the various programs and subroutines located within the specific limits of the dump, and a verb-noun listing of the code locations. The name of the program or subroutine is given as a heading and is followed by the initial and final addresses of the segment of memory which it occupies. Each program or subroutine is printed out in two parts: data (a decimal listing of the data words and their addresses) and code (a verb-noun listing of the code words and their addresses).

Octal Dump - An octal dump is a printout of memory within specified limits at a particular time. There are two columns of initial octal addresses and 16 columns of octal code or data words (refer to Figure 4). Starting from the left, column 1 and column 10 list the initial addresses of the following eight memory locations. Each set of eight columns (columns 2 through 9 and columns 11 through 18) presents a printout of the content of memory at that particular location. The entire dump is broken into several parts, the first of which is always a printout of the content of the interrupt locations. The remainder of the dump comprises a printout of the contents of memory at the locations occupied by the various programs and subroutines located within the limits of the dump. The name of the program or subroutine is given as a heading and is followed by the initial and final addresses of the segment of memory which it occupies. Each program or subroutine is dumped in two parts: data (an octal listing of the data words and their addresses) and code (an octal listing of the code words and their addresses).

```
INTERRUPT LOCATIONS

    5            .0000000000            .0000000000           10.0000000000            .0000000000            .0000000000
  135            .0000000000

   COMBINATIONS (DATA)    136-   145

  136       -65535.0000000000       -65535.0000000000           16.9999990000           5.0000000000          119.9999990000
  141            3.0000000000            6.0000000000           12.0000000000            .0000000000            .0000000000

   COMBINATIONS (CODE)    2048-   2067

2048 SET SCALE WITH   17                    INPUT/OUTPUT FRO    NUMBER OF IT        YIELD               N
2051 LET (IF)         N                     TRANSFORMED BY      FACTORIAL           YIELD               NFACTORIAL
2054 INPUT/OUTPUT FRO NUMBER OF IT          YIELD               M                   LET (IF)            M
2057 TRANSFORMED BY   FACTORIAL             YIELD               MFACTORIAL          LET (IF)            N
2060 MINUS            M                     TRANSFORMED BY      FACTORIAL           TIMES               MFACTORIAL
2063 YIELD            DENOMINATOR           LET (IF)            NFACTORIAL          OVER                DENOMINATOR
2066 INPUT/OUTPUT FRO NUMBER OF CO          HALT

   FACTORIAL (DATA)      146-   153

 146         2062.0000000000         2068.0000000000            .0000000000           2.0000000000         2069.9999000000
 151            1.0000000000         2080.0000000000            .0000000000

   FACTORIAL (CODE)      2068-   2081

2068 YIELD            N                     YIELD               RESULT              LET (IF)            N
2071 MINUS            1                     YIELD               N                   LET (IF)            N
2074 IS LESS THAN     1                     THEN GO TO          ANSWER              LET (IF)            RESULT
2077 TIMES            N                     YIELD               RESULT              GO TO  (RETURN      JAIL
2080 LET (IF)         RESULT                GO TO  (RETURN      FACTORIAL

REMAINING CORE IN DATA REGION

  2044            .0000000000            .0000000000            .0000000000            .0000000000

REMAINING CORE IN CODE REGION
```

Figure 3. Computer Printout of Decimal Dump

```
INTERRUPT LOCATIONS

000000  000000  000000  000000  000000  000000  000000  000000  000012    000010  000000  000000  000000  000000  000000  000000  000000  000000
000200  000000  000000  000000  000000  000000  000000  000000  000000

    COMBINATIONS (DATA)  000210-000221

000210  600001  600001  000021  000005  000170  000003  000006  000014    000220  000000  000000

    COMBINATIONS (CODE)  004000-004023

004000  320212  160211  600213  200213  060222  600214  160210  600215    004010  200215  060222  600216  200213  240215  060222  440216  600217
004020  200214  640217  160220  000000

    FACTORIAL (DATA)  000222-000231

000222  004016  004024  000000  000002  004026  000001  004040  000000

    FACTORIAL (CODE)  004024-004041

004024  600224  600225  200224  240227  600224  200224  260227  420230    004034  200225  440224  600225  620226  200225  620222

REMAINING CORE IN DATA REGION

003772  000000  000000  000000  000000  000000  000000

REMAINING CORE IN CODE REGION
```

Figure 4. Computer Printout of Octal Dump

Decimal Trace (Figure 5)

Column	Explanation
LABEL	This heading refers to the words in the first column that stand alone on a line. These words are the labels used in GO TO, THEN GO TO, TRANSFORMED BY, RETURN FROM, and PERFORM instructions.
INSTRUCTION	This column gives a fully spelled-out listing of the instruction being simulated.
OPERAND	This lists the first 12 characters of the operand name used with the instruction if the name is available.
OPERAND VALUE	This column gives the value of the operand stored in memory converted to a decimal number using the current value of the scale register to position the decimal point.
ACCUMULATED RESULT	This column presents the accumulated result after the simulation of each instruction converted to a decimal number using the current value of the scale register to position the decimal point.
TIME	This column lists the cumulative total time in microseconds for the execution of instructions.

Octal Trace (Figure 6)

Column	Explanation
IC	This column lists the value contained in the Instruction Counter for each simulated instruction. The words that appear in this column but stand alone on a line are the labels used in GO TO, THEN GO TO, TRANSFORMED BY, RETURN FROM, and PERFORM instructions.
INSTR	This column gives an 8-letter abbreviation of the instruction just simulated.
OPERAND	This lists the first twelve letters of the operand name used with the instruction if the name is available.

LABEL	INSTRUCTION	OPERAND	OPERAND VALUE	ACCUMULATED RESULT	TIME
	TIMES	N	2.0000000000	119.9999990000	402.50
	YIELD	RESULT	119.9999990000	119.9999990000	410.00
	GO TO (RETURN FROM)	JAIL			415.00
JAIL					
	LET (IF)	N	2.0000000000	2.0000000000	421.25
	MINUS	1	1.0000000000	1.0000000000	427.50
	YIELD	N	1.0000000000	1.0000000000	435.00
	LET (IF)	N	1.0000000000	1.0000000000	441.25
	IS LESS THAN	1	1.0000000000	1.0000000000	447.50
	THEN GO TO	ANSWER			452.50
	LET (IF)	RESULT	119.9999990000	119.9999990000	458.75
	TIMES	N	1.0000000000	119.9999990000	522.50
	YIELD	RESULT	119.9999990000	119.9999990000	530.00
	GO TO (RETURN FROM)	JAIL			535.00
JAIL					
	LET (IF)	N	1.0000000000	1.0000000000	541.25
	MINUS	1	1.0000000000	.0000000000	547.50
	YIELD	N	.0000000000	.0000000000	555.00
	LET (IF)	N	.0000000000	.0000000000	561.25
	IS LESS THAN	1	1.0000000000	.0000000000	567.50
	THEN GO TO	ANSWER			572.50
ANSWER					
	LET (IF)	RESULT	119.9999990000	119.9999990000	578.75
	GO TO (RETURN FROM)	FACTORIAL			583.75
FACTORIAL					
	YIELD	NFACTORIAL	119.9999990000	119.9999990000	591.25
*** 600001 IS SENT TO THE I/O UNIT.		INPUT DATA (000003)		3.0000000000 FROM DEVICE 1	591.25
	INPUT/OUTPUT FROM	NUMBER OF IT	-65535.0000000000	3.0000000000	602.50
	YIELD	M	3.0000000000	3.0000000000	610.00
	LET (IF)	M	3.0000000000	3.0000000000	616.25
	TRANSFORMED BY	FACTORIAL			626.25
FACTORIAL					
	YIELD	N	3.0000000000	3.0000000000	633.75
	YIELD	RESULT	3.0000000000	3.0000000000	641.25
	LET (IF)	N	3.0000000000	3.0000000000	647.50
	MINUS	1	1.0000000000	2.0000000000	653.75
	YIELD	N	2.0000000000	2.0000000000	661.25
	LET (IF)	N	2.0000000000	2.0000000000	667.50

Figure 5. Computer Printout of Decimal Trace

IC	INSTR	OPERAND	MOR	AC	EA	ACDO	S	MEM	P	SS	X	EAR	SL	TIME	I
JAIL															
004032	LET (IF)	N	000005	000005	000000	0100	21	200226	00	000000	0	000226	000000	311.25	
004033	MINUS	1	000001	000004	000000	0100	21	240231	00	000000	0	000231	000000	317.50	
004034	YIELD	N	000004	000004	000000	0100	21	600226	00	000000	0	000226	000000	325.00	
004035	IS LESS	2	000002	000004	000000	0100	21	260232	00	000000	0	000232	000000	331.25	
004036	THEN GO	ANSWER	004043	000004	000000	0100	21	420233	00	000000	0	000233	000000	336.25	
004037	LET (IF)	RESULT	000322	000322	000000	0100	21	200227	00	000000	0	000227	000000	342.50	
004040	TIMES	N	000004	001510	000000	0100	21	440226	00	000000	0	000226	000000	406.25	
004041	YIELD	RESULT	001510	001510	000000	0100	21	600227	00	000000	0	000227	000000	413.75	
004042	GO TO	JAIL	004032	001510	000000	0100	21	620230	00	000000	0	000230	000000	418.75	
JAIL															
004032	LET (IF)	N	000004	000004	000000	0100	21	200226	00	000000	0	000226	000000	425.00	
004033	MINUS	1	000001	000003	000000	0100	21	240231	00	000000	0	000231	000000	431.25	
004034	YIELD	N	000003	000003	000000	0100	21	600226	00	000000	0	000226	000000	438.75	
004035	IS LESS	2	000002	000003	000000	0100	21	260232	00	000000	0	000232	000000	445.00	
004036	THEN GO	ANSWER	004043	000003	000000	0100	21	420233	00	000000	0	000233	000000	450.00	
004037	LET (IF)	RESULT	001510	001510	000000	0100	21	200227	00	000000	0	000227	000000	456.25	
004040	TIMES	N	000003	004730	000000	0100	21	440226	00	000000	0	000226	000000	520.00	
004041	YIELD	RESULT	004730	004730	000000	0100	21	600227	00	000000	0	000227	000000	527.50	
004042	GO TO	JAIL	004032	004730	000000	0100	21	620230	00	000000	0	000230	000000	532.50	
JAIL															
004032	LET (IF)	N	000003	000003	000000	0100	21	200226	00	000000	0	000226	000000	538.75	
004033	MINUS	1	000001	000002	000000	0100	21	240231	00	000000	0	000231	000000	545.00	
004034	YIELD	N	000002	000002	000000	0100	21	600226	00	000000	0	000226	000000	552.50	
004035	IS LESS	2	000002	000002	000000	0100	21	260232	00	000000	0	000232	000000	558.75	
004036	THEN GO	ANSWER	004043	000002	000000	0100	21	420233	00	000000	0	000233	000000	563.75	
004037	LET (IF)	RESULT	004730	004730	000000	0100	21	200227	00	000000	0	000227	000000	570.00	
004040	TIMES	N	000002	011660	000000	0100	21	440226	00	000000	0	000226	000000	633.75	
004041	YIELD	RESULT	011660	011660	000000	0100	21	600227	00	000000	0	000227	000000	641.25	
004042	GO TO	JAIL	004032	011660	000000	0100	21	620230	00	000000	0	000230	000000	646.25	
JAIL															
004032	LET (IF)	N	000002	000002	000000	0100	21	200226	00	000000	0	000226	000000	[illegible].50	
004033	MINUS	1	000001	000001	000000	0100	21	240231	00	000000	0	000231	000000	[illegible].75	
004034	YIELD	N	000001	000001	000000	0100	21	600226	00	000000	0	000226	000000	666.25	
004035	IS LESS	2	000002	000001	000000	0110	21	260232	00	000000	0	000232	000000	672.50	
004036	THEN GO	ANSWER	004043	000001	000000	0100	21	420233	00	000000	0	000233	000000	677.50	
ANSWER															
004043	LET (IF)	RESULT	011660	011660	000000	0100	21	200227	00	000000	0	000227	000000	683.75	
004044	GO TO	FACTORIAL	004005	011660	000000	0100	21	620222	00	000000	0	000222	000000	688.75	
FACTORIAL															
004005	YIELD	N FACTORIAL	011660	011660	000000	0100	21	600215	00	000000	0	000215	000000	696.25	
*** 600001 IS SENT TO THE I/O UNIT.				INPUT DATA (000003)						3.0000000000 FROM DEVICE			1		696.25
004006	INPUT/OU	NUMBER OF IT	600001	000003	000000	0100	21	160210	00	000000	0	000210	000000	707.50	
004007	YIELD	M	000003	000003	000000	0100	21	600216	00	000000	0	000216	000000	715.00	
004010	LET (IF)	M	000003	000003	000000	0100	21	200216	00	000000	0	000216	000000	721.25	
004011	TRANSFOR	FACTORIAL	004012	000003	000000	0100	21	060222	00	000000	0	000222	000000	731.25	

Figure 6. Computer Printout of Octal Trace

Octal Trace (Figure 6) (Continued)

Column	Explanation
MOR	This is a printout of the value contained in the memory operand register after each simulated instruction.
AC	This column shows the value contained in the accumulator after each simulated instruction.
EA	This column shows the value contained in the extended accumulator after each simulated instruction.
ACDO	This lists the value contained in each of the following one-bit registers: AND/OR, carry, decision, and overflow after each simulated instruction.
S	This column presents the value of the scale register after each simulated instruction.
MEM	This is a printout of the contents of memory at the address indicated by the instruction counter prior to simulation of each instruction.
P	This lists the value of the page register after each simulated instruction.
SS	This column gives the value contained in the subscript register after each simulated instruction.
X	For each instruction, the value listed here is a one if bit 13 of the instruction contains a one, thus designating use of the subscript register to determine the effective address.
EAR	This gives an instruction-by-instruction listing of the contents of the effective address register. The content of this register specifies the memory address of the operand.
SL	This column lists the contents of the storage limit register after each simulated instruction.

Octal Trace (Figure 6) (Continued)

Column	Explanation
TIME	This column presents the cumulative total time in microseconds for the execution of instructions.
I	This is a listing of the interrupt number being processed, if any interrupts have occurred during the simulation.

Control Cards

All control cards must have a ; (11/8/6) punched in column one. The function follows with blanks being disregarded. Thereafter, one or more blanks are used as delimiters. No control function can be continued onto a second card. Anywhere number appears, a decimal radix will be assumed unless OCTAL is specified; i.e., 11 is 11_{10}, but OCTAL 11 is 9_{10}.

```
; DATE characters
```

This card causes the first 12 nonblank (blanks are delimiters) characters to be printed as a part of the heading printed by the major functions (Assembler, Loader, etc.). For example, 7/4/8, 1400 could be used for date and time on the listing for the current run.

```
; ASSEMBLE    name    PRINT    NOLIST    EDIT    AUGMENT
```

This card causes the execution program to call in the assembler to assemble a program designated by name and to inform it of the options desired by the user. The output of the assembler is accumulated on the assembly tape. Only the first twelve characters of name are retained, any remaining characters are ignored. There must be no blanks interspersed in a program name. The remaining control card fields are optional, and if used their format is free form with blanks used as separators.

Options:

PRINT - If this word is present on the control card, the assembler will print out a statement by statement listing of the source language input.

NOLIST - This option directs the assembler to suppress the card image printout. If this option is not present, then a card image listing of the program being assembled will be given.

EDIT - This option allows the user installation to use its own mnemonics in place of the distributed verb phrases. The editing procedure assumes a deck setup as shown in Figure 7. Each card of the editing deck must have the format indicated. In particular, each new verb phrase must be accompanied by the exact number code of the standard verb which it will replace. This option will cause the new verb table to be punched out in the form of data statements representing the internal tree structure of the verb table. The installation must then place this punched output in the subroutine VERB according to the comments given in that subroutine.

AUGMENT - This option allows the user to temporarily include mnemonics of his own choosing for the program which he is assembling. The AUGMENT option assumes the same deck setup as the EDIT option and must precede each source deck. The new verb phrases are valid only for the current assembly.

```
; MEMORY SIZE IS number
```

This card designates the memory size to be loaded, simulated, or listed. The variable number must have a value of either 4096 or 8192. If this control card is missing, a memory size of 4096 is assumed.

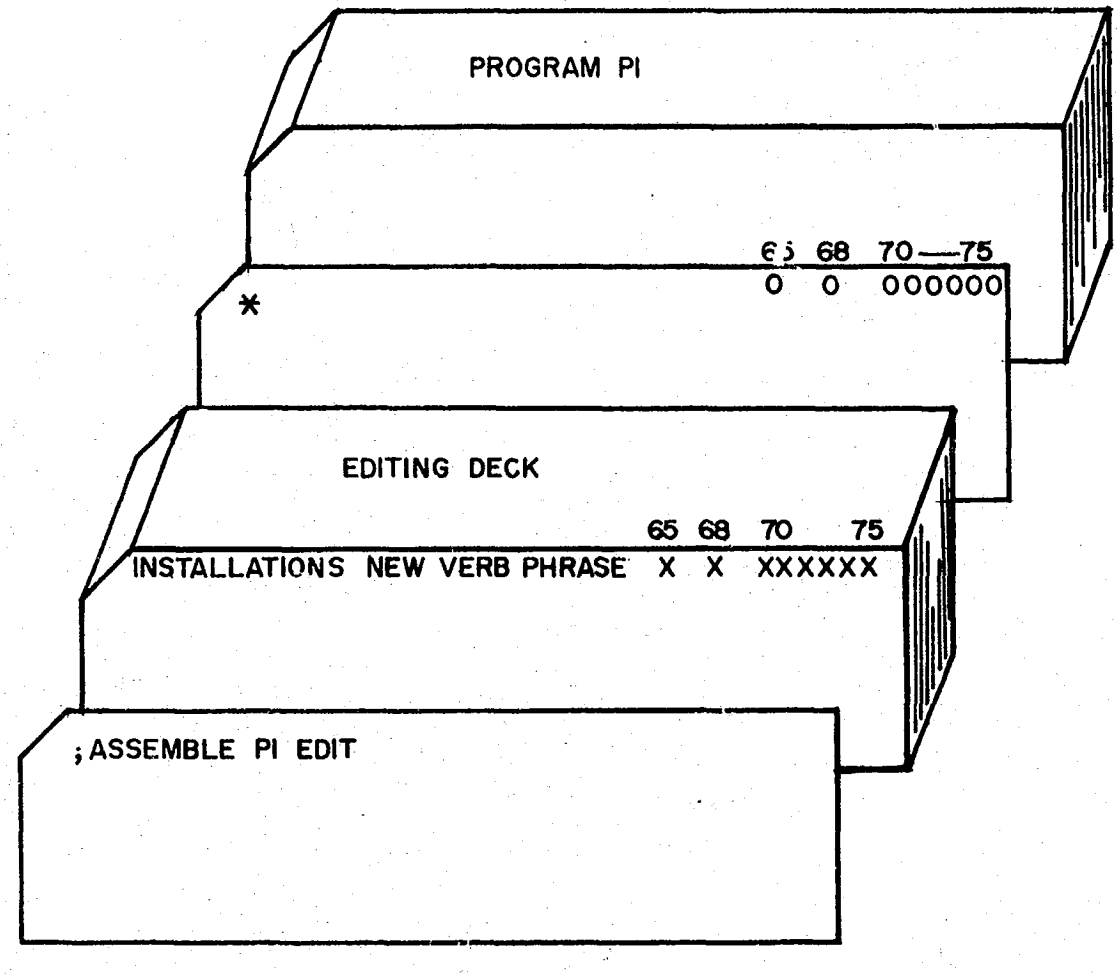


Figure 7. Deck Setup for EDIT and AUGMENT

PROGRAM NAME 1, PROGRAM NAME 2, . . . , PROGRAM NAME n

; LOAD $

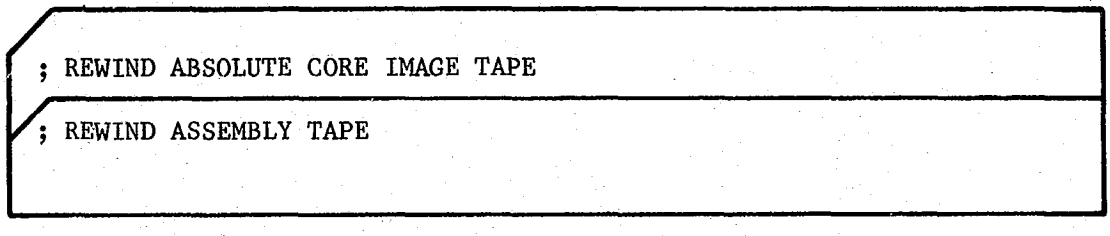
; LOAD program name

; LOAD program name DATA AT number AND CODE AT number

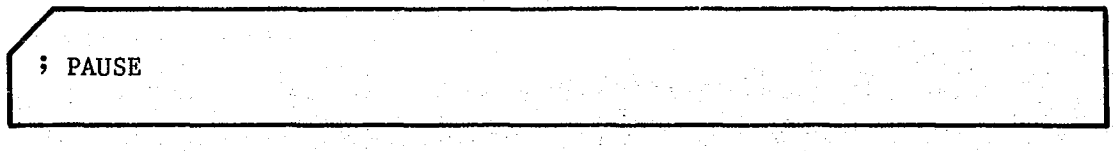
The ; LOAD program name card loads the entire assembly tape into core and then writes the On-Board Processor core image onto the absolute core image tape. The loader will assume a starting location of octal 210 for data.

When an 8K memory is being loaded, a starting location of octal 10000 is assumed for code, whereas octal 4000 is assumed when loading a 4K memory. The relative origins assumed by the loader may be altered by using the optional ; LOAD program name DATA AT number AND CODE AT number card. If the user chooses to alter the assumed origins, care should be exercised to prevent data words from overlapping a block 4096 words. For example, if a program is loaded with data beginning at location 4000 and there are 100 words of data, then the first 96 words must be accessed with a page register setting of zero, and the remaining four words must be accessed with a page register setting of one.

The ; LOAD $ indicates selective loading to the loader. The program names listed on the following cards will be loaded from the assembly tape. There must be a blank in column 1 of the program name cards but as many cards as needed may be used. However, the number of specified programs is limited to 25. The order in which the specified programs are loaded is the order in which they appear on the assembly tape. The relative origins assumed by the loader during selective loading may be altered by using the optional ; LOAD $ DATA AT number AND CODE AT number card.



Either of these cards causes the specified tape to be rewound.



This card will cause the OBP executive routine to pause. This option is included to allow the 920 user to switch tapes, save tapes, or hang tapes if necessary during his run

```
; END OF FILE ON ASSEMBLY TAPE
```

This card causes an end-of-file record to be written on the assembly tape. This is to be used if and only if the file of relocatable programs on the tape is to be used at a later time.

```
; SAVE PREVIOUS ASSEMBLIES
```

This card causes the executive routine to space down the assembly tape until an end-of-file record is read. The assembly tape is then backspaced over the EOF record, thus positioning it for further assemblies.

```
; DELETE name FROM PREVIOUS ASSEMBLIES
```

This card causes the executive routine to search the assembly tape and delete the assembly specified. All other assemblies are preserved. The end-of-file record is removed, and the assembly tape positioned for further assemblies. If a routine is to be reassembled with an assembly tape containing a previous assembly by the same name, the above card should be used to remove the old routine before the new routine is assembled.

```
; LIST
; LIST THE NOUN TABLE ALPHABETICALLY NUMERICALLY
```

This card causes the executive to read in the absolute core image tape prepared by the loader. It then will list the complete noun table of all the loaded programs. The

order of the two options is irrelevant and either one or both may be omitted. If both are omitted, then numeric and alphabetic lists will be given. Both lists may also be obtained with the abbreviated control card ; LIST.

Options: ALPHABETICALLY - This option causes the alphabetically ordered noun table to be printed.
NUMERICALLY - This option yields a printed list of the nouns used ordered on the relocated addresses assigned to the nouns.

```
; LIST THE ABSOLUTE CORE IMAGE TAPE
```

This card causes a complete listing of the absolute core image. The core image, allocation table, and noun table is read from the absolute core image tape produced by the loader. The allocation table is then used to list data and code for each program segment. Data is listed in an octal format. Code is listed as the octal bit pattern for each instruction, with the decoded verb-noun phrases and labels as they were defined in the program. All indirect instructions are flagged with the indirect address.

```
; CHECK PRINT XX
```

This card causes the executive to turn on debugging flags within the OBP software package. This control option is provided as an aid in maintaining the OBP package and should be used only as directed by GSFC personnel.

Simulator Control Cards

The control cards for the simulator are grouped into the following six categories: starting, tracing, dumping, interrupting, inputting, and stopping (and/or restarting).

The format for the simulator control cards is the same as the format for the control cards discussed in Control Cards section, page 42.

Column one must contain a ; (11/8/6 punches) and columns 2 through 80 contain the control information. One or more blanks are used as delimiters. No control function may continue onto a second card.

```
; SIMULATE
; SIMULATE name
```

This control card causes the simulator to be loaded into memory. The simulator sets the instruction counter to the load location of name. If name is omitted, then the instruction counter will be set to the normal, initial load location for instructions. The simulator is then ready to interpret any remaining simulator control cards. This card must be placed between the LOAD control card and the START control card.

```
; START
; START AT number or label
```

This control card, or one of its optional forms, should appear as the last control card. It causes the simulator to commence simulating at the location specified by the SIMULATE function. The optional form, where number or label is specified, causes the simulation to commence at the location specified.

```
; TRACE
```

This card causes the simulator to print tracing information in the decimal mode for each relocatable instruction simulated.

```
; TRACE OCTALLY
```

This control card causes the simulator to print tracing information in the octal mode for each relocatable instruction simulated.

```
; TRACE FROM label or number TO label or number
```

This card causes the simulator to print tracing information for each instruction simulated between the limits specified in the decimal mode by label and/or number.

```
; TRACE OCTALLY FROM label or number TO label or number
```

This card causes the simulator to print octal tracing information for each instruction simulated between the limits specified by label and/or number.

```
; TRACE PROGRAM name
```

This control card causes the simulator to print decimal tracing information where the limits of name are taken from the allocation table.

```
; TRACE OCTALLY PROGRAM name
```

This card causes the simulator to print tracing information in the octal mode where the limits of name are taken from the allocation table.

```
; DUMP AT label or number
```

This control card causes a decimal dump of the entire memory when the specified location is accessed (either code or data).

```
; DUMP OCTALLY AT label or number
```

This card causes an octal dump of the entire memory when the specified location is accessed (either code or data).

```
; DUMP AT label or number FROM label or number TO label or number WITH
  SCALE FACTOR number
```

This card causes a decimal dump of the segment of memory located within the limits specified by the second the third label or number when the location specified by the first label or number is accessed. The user denotes the scale factor to be used in generating the dump.

```
; DUMP OCTALLY AT label or number FROM label or number TO label or number
```

This card causes an octal dump of the segment of memory located within the limits specified by the second and third label or number when the location specified by the first label or number is accessed.

```
; DUMP AT label or number PROGRAM name WITH SCALE FACTOR number
```

When the location specified by label or number is accessed, this function causes a decimal dump of the single program specified by name where the limits of name are taken from the allocation table. The user denotes the scale factor to be used in the dump.

```
; DUMP OCTALLY AT label or number PROGRAM name
```

This control card causes an octal dump of the program specified by name, where the limits of name are taken from the allocation table, when the location specified by label or number is accessed.

```
; DUMP AT HALT
```

This control card causes a decimal dump of the entire memory at the time of simulation of a HALT statement.

```
; DUMP OCTALLY AT HALT
```

This control card causes an octal dump of the entire memory at the time of simulation of a HALT statement.

```
; INTERRUPT number EVERY number MICROSECONDS or MILLISECONDS or SECONDS
  STARTING AT number MICROSECONDS or MILLISECONDS or SECONDS
```

This card causes the specified interrupt (legal interrupts are 1 through 15) to occur at the specified interval beginning at the specified start time. The user must specify the time-measure of the interval length and start time in microseconds, milliseconds or seconds.

```
; INTERRUPT number EVERY number MICROSECONDS or MILLISECONDS or SECONDS
```

This control card causes the specified interrupt (legal interrupts are 1 through 15) to occur at the specified interval beginning at time zero. The user must specify the time-measure of the interval length to microseconds, milliseconds, or seconds.

```
; MAXIMUM TIME IS number MICROSECONDS or MILLISECONDS or SECONDS
```

This card will cause the simulation to cease at the specified simulation time. If this card is omitted, then a value of 5 milliseconds is assumed.

```
; MAXIMUM INSTRUCTIONS IS number
```

This control card causes the simulation to cease after the specified number of instructions has been executed. If this function is omitted, a value of 500 instructions is assumed.

Inputting Data

```
data      data      data      data      data      data      data$
; DATA FOR INPUT DEVICE number USING SCALE REGISTER SETTING OF number
```

This is the control card used for inputting data. Data may be input by one of four different input units numbered zero through three. A total of 200 data words may be specified. The user should denote the setting of the scale register for the input data words. Immediately following this card are the cards of input data for the specified input unit. As many cards as needed may be used and the format of these cards is free form with one or more blanks used as delimiters. Column one must be blank. The last data word must be followed by a $.

During the simulation, when the specified unit is referenced in the input command, the data words, which are converted in accordance with the specified scale register setting, are input one word per call until the buffer is exhausted.

It should be noted that if it is desired to supply the exact octal bit pattern for data words instead of converting a decimal number, then a scale register setting of 17 should be used regardless of the actual scale of the number supplied. For example, 0.5 with a scale register setting of zero may appear at input time as octal 177777, instead of octal 200000, because of truncation errors in the conversion process. However, octal 200000 with a scale register setting of 17 will appear at input time as octal 200000; i.e., 0.5 if the scale register is set to zero.

Stopping and/or Restarting - The normal means of ending a simulation is by simulating a HALT. Any remaining control cards will then be honored.

```
; NO HALT
```

This control card causes all HALT instructions to be simulated as PASS instructions.

```
; RESTART AT HALT
```

This control card causes one HALT instruction to be simulated as a PASS instruction. There must be a RESTART AT HALT card for each HALT that is to be simulated as a PASS instruction.

```
; STOP AT HALT
```

This control card allows the HALT statement to be simulated normally. Its main purpose is to allow proper page skipping between multiple jobs.

Diagnostics

A set of CPU diagnostics exists which is aimed at testing the execution of all machine instructions in order of increasing complexity. The diagnostics consist of seven program assemblies and must be segmented into two parts with a 4K word memory since the total set of diagnostics requires approximately 4,500 words of memory. The diagnostics can be loaded into OBP memory at the GSFC installation using the SDS 920 computer. Then, the results of the diagnostics can be monitored with a program in the SDS 920 version of the support software system. The diagnostics can be set in a mode whereby when a test fails, the accumulator is loaded with a code word which indicates which test failed. Thus, diagnostics can be run independently of the SDS 920 and if a display console is connected to the CPU, test results will be automatically displayed. Future effort will be directed toward the development of a set of system diagnostics in which the input/output unit, memory, and peripheral devices, if possible, are also tested.

Library

The following subroutines have been added to the GSFC 920 system tape.

- ARCTAN

  Entry Points: ARCTAN

  Accuracy: 10^{-5}

Storage: 36 data words + 73 code words

Execution Time: 800 usec

Function: The argument in radians, scaled by the scale register, is received in the accumulator. The arctangent of the argument is computed and returned in the accumulator with the original scale setting. Overflow will be set if the result is larger in magnitude than 2^s where s is the content of the scale register.

- EXPONENTIAL

  Entry Points: E-EXPONENTIAL, 10-EXPONENTIAL, and 2-EXPONENTIAL

  Accuracy: 10^{-4}

  Storage: 43 data words + 84 code words

  Execution Time: 800 usec

  Function: The argument X, scaled by the scale register, is received in the accumulator and is transformed by either e^x, 10^x, or 2^x depending on the entry point. The result returned in the accumulator has the original scale setting. Overflow will be set if the magnitude of the result is greater than 2^s, where s is the contents of the scale register.

- LOGARITHM

  Entry Points: NATURAL LOGARITHM, COMMON LOGARITHM, and LOG BASE - 2

  Accuracy: 10^{-4}

  Storage: 41 data words + 100 code words

  Execution Time: 800 usec

  Function: The argument X, scaled by the scale register, is received in the accumulator and is transformed by either log (base e), log (base 10), or log (base 2), depending on the entry point. The result returned in the accumulator has the original scale setting and overflow will be set for negative arguments or for arguments less than 1 which yield a result larger in magnitude than 2^s where s is the content of the scale register.

- SIN/COS

  Entry Points: SIN, COS, SINF and COSF

  Accuracy: 10^{-4}

  Storage: 35 data words + 59 code words

  Execution Time: 500 usec

  Function: For the entries SIN and COS the argument is assumed to have a scale of zero and is expressed in fractions of PI/2. The argument X as received in the accumulator is transformed by either SIN (X) or COS (X) and returned in the accumulator. The cosine of 0 will be returned as $1-2^{-18}$. For the entries SINF and COSF, the argument X, in radians, with a scale determined by the scale register, will be transformed by either SIN (X) or COS (X) and returned in the accumulator with the original scale setting.

- SQUARE ROOT

  Entry Points: SQUARE ROOT

  Accuracy: 10^{-5}

  Storage: 44 data words + 83 code words

  Execution Time: 800 usec

  Function: The double length argument X, with a scale setting equal to the contents of the scale register, is received in the accumulator; extended accumulator and is transformed by $(X)^{1/2}$ and returned in the accumulator with the original scale setting. Negative arguments will be returned as octal 400000 and overflow will be set.

PROGRAMMING NOTES

The support software which has been developed for the OBP was aimed at allowing independent programming by OBP users at different locations. In keeping with this philosophy, all I/O processing, interrupt processing, data decommutation and formatting, job sequencing, and storage limit control will be programmed by one group so that program integration and checkout problems will be minimized. The items under this section, then, that relate to some of these functions are presented to alleviate curiosity.

It is the current intent to be able to provide data and code linkage between an EXECUTIVE routine and OBP worker programs so that the author of the worker programs can assume his input data will be properly formatted and appear at a known place and he can operate on these numbers, using available mathematical subroutines, and deposit his output numbers in a known segment of storage, leaving all housekeeping duties to the central housekeeping group.

Input/Output

The OBP has two modes of input/output: program-controlled and cycle steal. For program-controlled I/O, either the contents of the accumulator is output to a device or the data from a device is input to the accumulator. For cycle steal operation, there are two cycle steal channels in the input/output unit which control block transfers of data between input/output devices and memory. These data transfers are independent of program execution and the external device supplies the I/O request pulses. The cycle steal channels provide memory addressing and either the gating of data from an external device onto a memory input data bus or select pulses to an external device, since for cycle steal output operations the data is broadcast to all devices. The initialization of a cycle steal channel may be accomplished either under CPU control by executing the appropriate I/O instruction or under external command control by receiving the appropriate ground command sequence. The following information is sent to the I/O unit to initialize a cycle steal channel:

- Starting address (16 bits) - denotes the initial address of the data block to be transferred.
- Device (2 bits) - specifies which device is to be connected to memory. A device will have different numbers for input and output.
- Control channel (1 bit) - specifies which of two control channels is to be initialized. Note that control channel redundancy exists since a device can use either channel.
- Block length (12 bits) - indicates the number of words to be transferred.

Once a channel has been initialized, it will remain active until either the block transfer is complete, at which time a block length = zero interrupt is produced, or until the channel is re-initialized.

The one I/O instruction results in four different actions depending on bits 17 and 18 of the contents of the effective address of the instruction. When the I/O instruction is executed, the content of the effective

address instruction is examined by the I/O unit so that harmony will exist between the I/O and the CPU. The four I/O operations resulting from executing the I/O instruction are shown in Table 2. The I/O unit is designed such that any device connected to the OBP can transfer data in and out of the accumulator by executing the I/O instruction. For input, 1,1 in bits 18, 17 and device number in bits 13-16 of the cell at the effective address is required. For output, 1,0 in bits 18, 17 and device number in bits 13-16 of the cell at the effective address is required. A subset of all devices, depending on the application, may be connected to memory through cycle steal control. Again, to effect this connection by program execution, the I/O instruction must be set up such that the contents of memory at the effective address has 0,0 in bits 18, 17; block length in bits 1-12, device number in bits 13-15 and cycle steal channel selection is made in bit 16. For initializing a cycle steal channel the accumulator must be loaded prior to executing the I/O instruction with the block starting address right justified. The fourth I/O action which occurs when bits 18, 17 of the content of the effective address are 0,1 will depend on the application since the remaining 16 bits of that memory word are available for any use by the I/O unit.

Factors to be considered in the choice of program controlled versus cycle steal I/O are:

- Hardware - additional gating is required to add the capability of connecting devices to memory through a cycle steal channel.

- Timing - the fastest program controlled I/O rate is limited by an I/O sequence consisting of:

```
DEFINING I/O ROUTINE

LET INPUT FROM DEVICE

YIELD SUBSCRIPTED BLOCK

STEP SUBSCRIPT BY 1 IF

SUBSCRIPT IS NOT GREATER

THAN BLOCK LENGTH THEN GO TO I/O ROUTINE
```

This routine for input and a similar one for output requires 37.75 usec to execute plus the time required to process interrupts needed to synchronize the data with program execution. Cycle steal I/O rates as high as 400 (10^3) words/second are possible. Generally, program control of I/O is restricted to 1) devices for which the data transfers are program dependent or 2) very low frequency devices if interrupt of the program is necessary for synchronization.

TABLE 2

(M) INPUT/OUTPUT FORMAT

CONNECT TO M

	M						A		
18	17	16	15	14	13	12---------1	18	17	16---------1
0	0	CC	X	Device #		Block Length	X	CC	Starting ADD

CC = 0 Channel A is selected
CC = 1 Channel B is selected

LET FUNCTION TO M

	M			A	
18	17	16---------------1		18	1
0	1	Bits to be defined		X	X

OUTPUT TO M

	M			A	
18	17	16----13	12----------1	18	1
1	0	Device #	X X	Any Data to be output	

See note

LET INPUT FROM M

	M			A	
18	17	16 13	12----------1	18	1
1	1	Device #	X X	Holds incoming word	

See note

NOTE: Device #'s 0000_2 to 0011_2 are identical to Device #'s 00_2 to 11_2 for the "CONNECT TO" instruction.

Interrupts

There is a 15 bit register in the I/O unit which stores interrupt requests. As each request is serviced, the corresponding bit in the register will be reset. There is another 15 bit register in the I/O, called the interrupt priority register, where each stage indicates whether or not an interrupt request at that level is to be locked out. An interrupt is sent to the CPU when the request bit is set and the corresponding bit of the interrupt priority register is zero. Should two allowable interrupts simultaneously request interrupts, there is a hard-wired 15 level priority circuit in the I/O unit (interrupt 1 is of highest priority) to determine which interrupt request is to be honored first. The interrupt priority register is set either when an interrupt is sent from the I/O unit to the CPU, or when the CPU executes an EXIT instruction, or when the CPU executes a RESUME instruction, the instruction normally executed at the termination of an interrupt routine. The interrupt priority is set by the contents of a memory word so that the determination of which interrupts are to be allowed can be dynamically changed. The one exception is that interrupt 1 has top priority and cannot be locked out. This interrupt will be used to initiate program execution. When an interrupt is received by the CPU, the instruction being executed will be completed and then an automatic sequence is entered in which the address of the next instruction to be executed, the contents of the storage limit register, the miscellaneous registers (page, scale, D, O/A, OV, and C), and the current status of the interrupt priority register are stored in a bank of four memory locations, N0 to N3. Then the same registers will be loaded from the four memory locations N4 to N7. It is the programmer's responsibility to initialize locations N4 to N7, where N is the interrupt number, with the desired interrupt priority, miscellaneous register settings, storage limit setting and starting address of interrupt routine N. At the conclusion of interrupt routine N there must be a RESUME FROM N instruction which will result in returning control to the interrupted program and restoring the interrupt priority, storage limit, and miscellaneous registers. Of course, any addressable register which may be altered during the interrupt routine should be tucked away and uncovered upon entering and leaving the interrupt program.

Scale Register

The 6 bit scale register can be loaded from memory with the SET SCALE WITH noun instruction and the instruction LET SCALE maps the scale register onto bits 1-6 of the accumulator with the remaining accumulator bits being set to zero. The scale register is used to control the length of automatic double-length left shifts following multiply operations and right shifts preceding divide operations. The value of the scale register denotes the number of places to the right of the sign bit at which the binary point is located.

Fractional numbers have a scale of 0, and integers have a scale of 17. Another convenient way of visualizing the scale register is if the contents of the scale register is s and the unscaled accumulator contents is X' where $0 \leq X' < 1$, then the scaled number in the accumulator $X=2^sX'$. If two numbers are multiplied, the result should have the same scale as the arguments. That is, if $X=2^sX'$ and $Y=2^sY'$, $X \cdot Y = 2^{2s} \cdot X'Y'$. What is needed is for the product to be of the form $2^s(Z)$, thus $Z=2^sX'Y'$ which is a fractional multiplication of X times Y shifted left s places, exactly what occurs in hardware with the scale register. Similarly, for division $X/Y = 2^sX'/2^sY' = X'/Y'$ but, to maintain the same scale, the quotient should be of the form 2^sZ where $Z=2^{-s}X'/Y'$ which indicates a shift right by X by s places prior to division, again what is accomplished in hardware with the implementation of the scale registers. It is noteworthy that if two numbers with different scales, s1 and s2, are multiplied with a scale register setting of s1 then the product will be scaled by s2. That is, for $X = X'2^{s1}$ and $Y = Y'2^{s2}$, $X \cdot Y = 2^{s1+s2}(X'Y')$, but what the hardware produces is $2^{s1}(X'Y')$ for a scale register setting of s1. Therefore, $X \cdot Y = 2^{s1+s2}(X'Y') = 2^{s2}(2^{s1}X'Y')$ which is the product scaled at s2.

The scale register is also used to store the shift length of the NORMALIZE instruction. The NORMALIZE instruction performs a double-length shift left for up to 34 places until bits 1 and 2 of the accumulator are different. If the accumulator and extended accumulator contain all zeros, zero will be set into the scale register following a NORMALIZE. Since the NORMALIZE alters the scale register and multiply/divide operations always use the scale register, care should be taken when mixing these operations.

Program Linkage

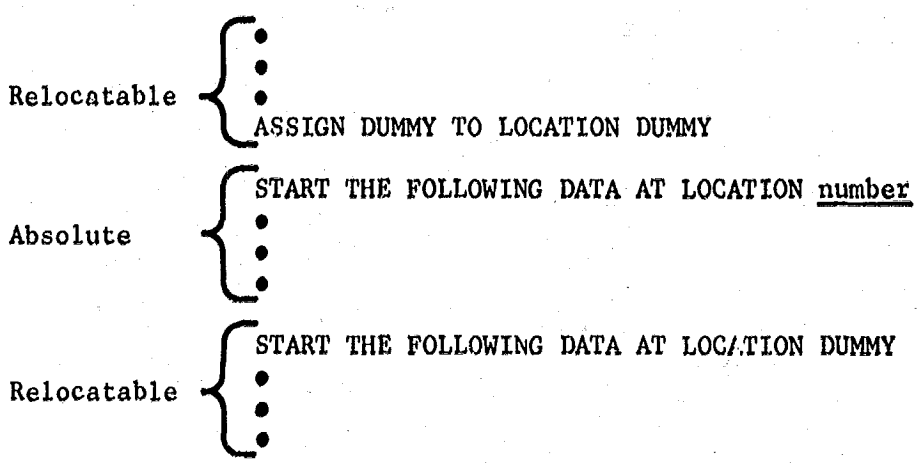
The DEFINING label/noun assembler directive makes label/noun available as a transfer point within an assembled program segment. Each transfer, GO TO, THEN GO TO, RESUME FROM, and TRANSFORMED BY is indirect and the DEFINING statement makes the pointer available for loading into the referenced memory location when a transfer instruction is encountered. For program linkage between separate assemblies, a DEFINING FOR EXTERNAL USE statement makes that definition external and, during load time, if another assembly had the same undefined symbol, linkage is then made. All subroutines, then, must have a DEFINING FOR EXTERNAL USE preceding each entry point for providing linkage information. Two cells are reserved for subroutine entry points with the first used to hold the address to which control will return and the second holds the pointer to the entry point. When a TRANSFORMED BY instruction is executed the current instruction address +1 is written into the first word of the two word address block corresponding to the defined entry point X and the instruction counter is set to X, the contents of the second word. Each return point in a subroutine, then, should be a RETURN FROM (GO TO) X which automatically returns control to the calling program. If a routine has more than one entry point with a common return point, a procedure which

may be used is that each entry point should store the return address in a dummy return point so that one return statement RETURN FROM DUMMY RETURN may be used. As an example, suppose that a SINE routine had three entry points corresponding to the input argument in radians, degrees, or fractions of pi/2. If the polynomial expansion assumed an argument in fractions of pi/2, the following code in the subroutine to allow correct linkage could be used:

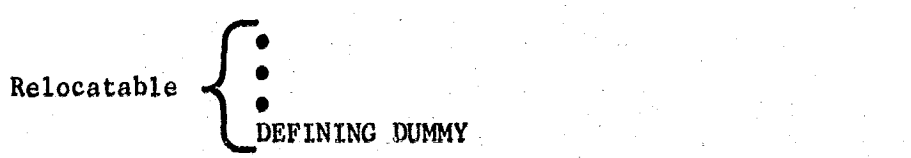
```
                                DEFINING FOR EXTERNAL USE SIN 1 LET IT
TIMES 2 OVER 3.1416 YIELD ARGUMENT. LET SIN 1 YIELD SUBROUTINE.
GO TO COMPUTE. DEFINING FOR EXTERNAL USE SIN 2. LET IT OVER 90
YIELD ARGUMENT. LET SIN 2 YIELD SUBROUTINE. GO TO COMPUTE.
DEFINING FOR EXTERNAL USE SIN 3. LET IT YIELD ARGUMENT. LET SIN
3 YIELD SUBROUTINE.
DEFINING COMPUTE.....
RETURN FROM SUBROUTINE.
DEFINING SUBROUTINE. END OF THIS PROGRAM SEGMENT.
```

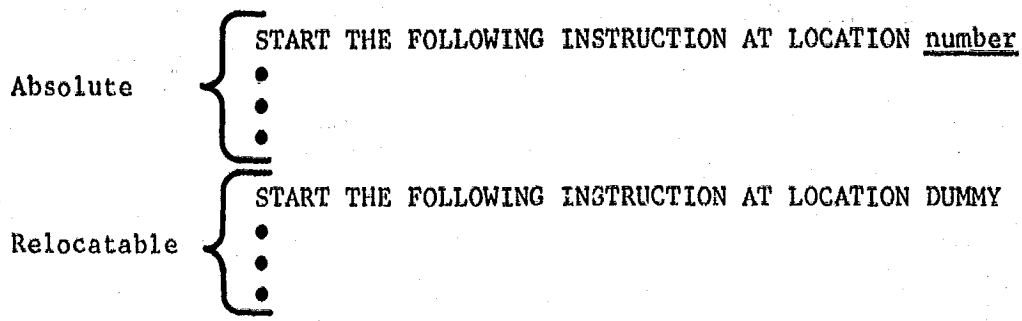
When writing separate program segments, it is advisable to leave code and data relocatable and let the loader provide linkage and absolute memory assignments. However, if it is necessary to make a block of code or data absolute, the following sequence permits the return to relocatable mode at the completion of the absolute segment.

- Relocatable to absolute to relocatable data storage assignment:



- Relocatable to absolute to relocatable code storage assignment:





Storage Limit Register

The OBP has an 18 bit storage limit register which is used to enable blocks of memory into which writing is permitted. Those instructions which require writing into memory, and therefore consult the storage limit register, are YIELD, TRANSFORMED BY, SAVE EXTENSION IN, and SAVE SUBSCRIPT IN. The register is broken into two 9 bit fields - A and B - where A = (B1-B9) and B = (B10-B18), BI = the I^{th} bit of the storage limit register numbered from the right. A and B represent upper and lower limits on the 9 high order bits of a 16 bit effective address between which writing will be permitted. Stated symbolically, if C = (B8-B16) of the effective operand address for one of the four instructions listed above and A and B are the two fields of the storage limit register, then if $B \leq C \leq A$ - write permitted, otherwise, write will not be permitted. Note that if A = B then one 128 word block is enabled whereas if A = (1,1.....1) and B = (0,0.....0) then all of memory is enabled.

A Programming Example

The example which follows was run on the GSFC 920 system. This example is the combinations routine which computes the number of combinations of n things taken k at a time and calls the routine factorial, which takes the factorial of positive entegers. The first control card is a DATE card which results in the date being listed tor assembler/simulator printouts. The next control card is the ASSEMBLE card for COMBINATIONS using the print option which causes a line-by-line listing of assembly language statements which follows the card image listing. A synopsis of storage allocation data follows each assembly listing. The assembly language deck for COMBINATIONS followed the ASSEMBLE COMBINATIONS control card. The first two pages of the listing are associated with the program COMBINATIONS. The next set of cards is the ASSEMBLE control card for the program FACTORIAL followed by the source deck for that program. The assembler again produces the printout shown on the third and fourth pages of listing for FACTORIAL. If the ASSEMBLE control card did not have PRINT following the program name, the line-by-line listing of source statements would not appear. The next input card was the LOAD control card which produces the printout on the fifth page of the example and, of course, performs a loading of the two assembled programs. A LIST THE ABSOLUTE CORE IMAGE TAPE control card followed the LOAD. This listing

shows the initial values of the data cells and decodes the program segments with all transfers flagged to show the indirect pointer. A sequence of control cards to provide a simulation and octal trace with input data followed. The octal trace continued until the program was finished, and the last part of the example shows the frequency of instruction using statistics which the simulator provides upon normal termination of a simulation.

```
; DATE 12/7/57

; ASSEMBLE COMBINATIONS PRINT

ASSEMBLER FOR THE ON-BOARD PROCESSOR 12/7/57


     THE RADIX FOR NUMBERS IS OCTAL
     PREDEFINE NUMBER OF ITEMS PER GROUP AS THE VALUE OCTAL 600001
     PREDEFINE NUMBER OF ITEMS AS THE VALUE 600001
     PREDEFINE NUMBER OF COMBINATIONS AS THE VALUE 400001
     THE RADIX FOR NUMBERS IS DECIMAL.
          SET SCALE WITH 17.
          LET INPUT FROM NUMBER OF ITEMS YIELD N.
          LET N TRANSFORMED BY FACTORIAL YIELD N FACTORIAL.
          LET INPUT FROM NUMBER OF ITEMS PER GROUP YIELD M.
          LET M TRANSFORMED BY FACTORIAL YIELD M FACTORIAL.
          LET N MINUS M TRANSFORMED BY FACTORIAL TIMES M FACTORIAL
            YIELD DENOMINATOR.
          LET N FACTORIAL OVER DENOMINATOR BE OUTPUT TO NUMBER OF COMBINATIONS.
          HALT.
     END OF THIS PROGRAM SEGMENT.
```

*** ASSEMBLY LANGUAGE STATEMENTS FOR COMBINATIONS ***

```
1.                          THE RADIX FOR NUMBERS IS OCTAL
2.                          PREDEFINE NUMBER OF ITEMS PER GROUP
3.                          AS THE VALUE OCTAL 600001
4.                          PREDEFINE NUMBER OF ITEMS
5.                          AS THE VALUE 600001
6.                          PREDEFINE NUMBER OF COMBINATIONS
7.                          AS THE VALUE 400001
8.                          THE RADIX FOR NUMBERS IS DECIMAL
9.                      1.  SET SCALE WITH 17
10.                     2.  LET INPUT FROM NUMBER OF ITEMS
11.                     3.  YIELD N
12.                     4.  LET N
13.           *         5.  TRANSFORMED BY FACTORIAL
14.                     6.  YIELD N FACTORIAL
15.                     7.  LET INPUT FROM NUMBER OF ITEMS PER GROUP
16.                     8.  YIELD M
17.                     9.  LET M
18.           *        10.  TRANSFORMED BY FACTORIAL
19.                    11.  YIELD M FACTORIAL
20.                    12.  LET N
21.                    13.  MINUS M
22.           *        14.  TRANSFORMED BY FACTORIAL
23.                    15.  TIMES M FACTORIAL
24.                    16.  YIELD DENOMINATOR
25.                    17.  LET N FACTORIAL
26.                    18.  OVER DENOMINATOR
27.                         BE
28.                    19.  OUTPUT TO NUMBER OF COMBINATIONS
29.                    20.  HALT
30.                         END OF THIS PROGRAM SEGMENT

END OF LISTING.    0 *DIAGNOSTIC* MESSAGE(S).

EXTERNAL DEFINITIONS   DATA ADDRESS ASSIGNED

UNDEFINED SYMBOLS

FACTORIAL

STORAGE ASSIGNMENT FOR NOUNS AND LITERALS (LOCATION, TYPE/I=INDIRECT, A=ABSOLUTE LOC., B=I AND A/,NAME)

0000    NUMBER OF IT 0001   NUMBER OF IT 0002   NUMBER OF CO 0003   17           0004   N
0005    N FACTORIAL  0006   M            0007   M FACTORIAL  0010   DENOMINATOR

THE FOLLOWING SYMBOLS WERE MENTIONED BUT NOT USED IN ANY MACHINE INSTRUCTION

        600001              400001
```

```
; ASSEMBLE FACTORIAL PRINT

ASSEMBLER FOR THE ON-BOARD PROCESSOR 12/7/57


     DEFINING FOR EXTERNAL USE FACTORIAL.
          IF IT IS LESS THAN 0 THEN GO TO (RETURN FROM) FACTORIAL.
               IF IT IS EQUAL TO 0 THEN GO TO ANSWER EQUALS ONE;
             OTHERWISE LET IT YIELD N AND ALSO YIELD RESULT.
     DEFINING JAIL.  LET N MINUS 1 YIELD N.
          IF IT IS LESS THAN 2 THEN GO TO ANSWER; OTHERWISE,
             LET RESULT TIMES N YIELD RESULT.
          GO TO JAIL.
     DEFINING ANSWER.  LET RESULT BE.  RETURN FROM FACTORIAL.
     DEFINING ANSWER EQUALS ONE, LET 1 BE.  RETURN FROM FACTORIAL.
     END OF THIS PROGRAM SEGMENT.
```

*** ASSEMBLY LANGUAGE STATEMENTS FOR FACTORIAL ***

```
1.  FACTORIAL                   DEFINING FOR EXTERNAL USE FACTORIAL
2.                              IF IT
3.                          1.  IS LESS THAN 0
4.              *           2.  THEN GO TO FACTORIAL
5.                              IF IT
6.                          3.  IS EQUAL TO 0
7.              *           4.  THEN GO TO ANSWER EQUALS ONE
8.                              OTHERWISE
9.                              LET IT
10.                         5.  YIELD N
11.                             AND ALSO
12.                         6.  YIELD RESULT
13. JAIL                        DEFINING JAIL
14.                         7.  LET N
15.                         8.  MINUS 1
16.                         9.  YIELD N
17.                             IF IT
18.                        10.  IS LESS THAN 2
19.             *          11.  THEN GO TO ANSWER
20.                             OTHERWISE
21.                        12.  LET RESULT
22.                        13.  TIMES N
23.                        14.  YIELD RESULT
24.             *          15.  GO TO JAIL
25. ANSWER                      DEFINING ANSWER
26.                        16.  LET RESULT
27.                             BE
28.             *          17.  RETURN FROM FACTORIAL
29. ANSWER EQUALS               DEFINING ANSWER EQUALS ONE
30.                        18.  LET 1
31.                             BE
32.             *          19.  RETURN FROM FACTORIAL
33.                             END OF THIS PROGRAM SEGMENT

END OF LISTING.     0 *DIAGNOSTIC* MESSAGE(S).

EXTERNAL DEFINITIONS    DATA ADDRESS ASSIGNED

FACTORIAL               0000

UNDEFINED SYMBOLS

STORAGE ASSIGNMENT FOR NOUNS AND LITERALS (LOCATION, TYPE/I=INDIRECT, A=ABSOLUTE LOC., B=I AND A/, NAME)

0000   I FACTORIAL   0002   0          0003   I ANSWER EQUAL 0004   N          0005   RESULT
0006   I JAIL        0007   1          0010   2              0011   I ANSWER
```

```
; LOAD

LOADER FOR ON-BOARD PROCESSOR SOFTWARE 12/7/57

PREAMBLE VALUES FOR COMBINATIONS
  DATA LENGTH                    9
  CODE LENGTH                   20
  PRESET LOCATIONS               3
  LITERALS                       1
  INDIRECT ADDRESSES             0
  EXTERNAL DEFINITIONS           0
  UNDEFINED SYMBOLS              1
  NOUNS                         10

PREAMBLE VALUES FOR FACTORIAL
  DATA LENGTH                   10
  CODE LENGTH                   19
  PRESET LOCATIONS               0
  LITERALS                       3
  INDIRECT ADDRESSES             3
  EXTERNAL DEFINITIONS           1
  UNDEFINED SYMBOLS              0
  NOUNS                          9

EXTERNAL DEFINITIONS
  FACTORIAL

CORE LIMITS
      DATA 000210-000234 / CODE 004000-004046
      STARTING ADDRESS 004000

CORE ALLOCATION

      COMBINATIONS
           DATA 000210-000221
           CODE 004000-004023
      FACTORIAL
           DATA 000222-000234
           CODE 004024-004046
END OF ALLOCATION
```

```
; LIST THE ABSOLUTE CORE IMAGE TAPE

ABSOLUTE CORE IMAGE LIST 12/7/57

                                                    **********
                                                      DATA
                                                    **********

     INTERRUPT LOCATIONS

000200  000000  000000  000000  000000  000000  000000  000000  000000

     COMBINATIONS 000210-000221

000210  600001  600001  400001  000021  000000  000000  000000  000000    000220  000000  000000

     FACTORIAL    000222-000234

000222  004024  004024  000000  004045  000000  000000  004032  000001    000232  000002  004043  000000

     ABSOLUTE 000235-003777

003775  000000  000000  000000

                                                    **********
                                                      CODE
                                                    **********

     COMBINATIONS 004000-004023

004000  32 0213              SET SCALE WITH          17
004001  16 0211              INPUT/OUTPUT FROM       NUMBER OF IT
004002  60 0214              YIELD                   N
004003  20 0214              LET (IF)                N
004004  06 0222              TRANSFORMED BY          FACTORIAL       004024
004005  60 0215              YIELD                   N FACTORIAL
004006  16 0210              INPUT/OUTPUT FROM       NUMBER OF IT
004007  60 0216              YIELD                   M
004010  20 0216              LET (IF)                M
004011  06 0222              TRANSFORMED BY          FACTORIAL       004024
004012  60 0217              YIELD                   M FACTORIAL
004013  20 0214              LET (IF)                N
004014  24 0216              MINUS                   M
004015  06 0222              TRANSFORMED BY          FACTORIAL       004024
004016  44 0217              TIMES                   M FACTORIAL
004017  60 0220              YIELD                   DENOMINATOR
004020  20 0215              LET (IF)                N FACTORIAL
004021  64 0220              OVER                    DENOMINATOR
004022  16 0212              INPUT/OUTPUT FROM       NUMBER OF CO
004023  00 0000              HALT
```

FACTORIAL 004024-004046

```
004024  26  0224  FACTORIAL      IS LESS THAN          0
004025  42  0222                 THEN GO TO            FACTORIAL       004024
004026  46  0224                 IS EQUAL TO           0
004027  42  0225                 THEN GO TO            ANSWER EQUAL    004045
004030  60  0226                 YIELD                 N
004031  60  0227                 YIELD                 RESULT
004032  20  0226  JAIL           LET (IF)              N
004033  24  0231                 MINUS                 1
004034  60  0226                 YIELD                 N
004035  26  0232                 IS LESS THAN          2
004036  42  0233                 THEN GO TO            ANSWER          004043
004037  20  0227                 LET (IF)              RESULT
004040  44  0226                 TIMES                 N
004041  60  0227                 YIELD                 RESULT
004042  62  0230                 GO TO  (RETURN FROM)  JAIL            004032
004043  20  0227  ANSWER         LET (IF)              RESULT
004044  62  0222                 GO TO  (RETURN FROM)  FACTORIAL       004024
004045  20  0231  ANSWER EQUAL   LET (IF)              1
004046  62  0222                 GO TO  (RETURN FROM)  FACTORIAL       004024
```

ABSOLUTE 004047-007777

```
; SIMULATE

; TRACE OCTALLY

; DATA FOR INPUT DEVICE 1 USING SCALE REGISTER SETTING OF 17

     7  3  $

; START

ON-BOARD PROCESSOR SIMULATOR 12/7/57
```

IC	INSTR	OPERAND	MOR	AC	EA	ACDO	S	MEM	P	SS	X	EAR	SL	TIME	I
004000	SET SCAL	17	000021	000000	000000	0000	21	320213	00	000000	0	000213	000000	5.00	
*** 600001 IS SENT TO THE I/O UNIT.			INPUT DATA (000007)							7.0000000000 FROM DEVICE			1		5.00
004001	INPUT/OU	NUMBER OF IT	600001	000007	000000	0000	21	160211	00	000000	0	000211	000000	16.25	
004002	YIELD	N	000007	000007	000000	0000	21	600214	00	000000	0	000214	000000	23.75	
004003	LET (IF)	N	000007	000007	000000	0000	21	200214	00	000000	0	000214	000000	30.00	
004004	TRANSFOR	FACTORIAL	004005	000007	000000	0000	21	060222	00	000000	0	000222	000000	40.00	
FACTORIAL															
004024	IS LESS	0	000000	000007	000000	0000	21	260224	00	000000	0	000224	000000	46.25	
004025	THEN GO	FACTORIAL	004005	000007	000000	0000	21	420222	00	000000	0	000222	000000	51.25	
004026	IS EQUAL	0	000000	000007	000000	0000	21	460224	00	000000	0	000224	000000	57.50	
004027	THEN GO	ANSWER EQUAL	004045	000007	000000	0000	21	420225	00	000000	0	000225	000000	62.50	
004030	YIELD	N	000007	000007	000000	0000	21	600226	00	000000	0	000226	000000	70.00	
004031	YIELD	RESULT	000007	000007	000000	0000	21	600227	00	000000	0	000227	000000	77.50	
004032	LET (IF)	N	000007	000007	000000	0000	21	200226	00	000000	0	000226	000000	83.75	
004033	MINUS	1	000001	000006	000000	0100	21	240231	00	000000	0	000231	000000	90.00	
004034	YIELD	N	000006	000006	000000	0100	21	600226	00	000000	0	000226	000000	97.50	
004035	IS LESS	2	000002	000006	000000	0100	21	260232	00	000000	0	000232	000000	103.75	
004036	THEN GO	ANSWER	004043	000006	000000	0100	21	420233	00	000000	0	000233	000000	108.75	
004037	LET (IF)	RESULT	000007	000007	000000	0100	21	200227	00	000000	0	000227	000000	115.00	
004040	TIMES	N	000006	000052	000000	0100	21	440226	00	000000	0	000226	000000	178.75	
004041	YIELD	RESULT	000052	000052	000000	0100	21	600227	00	000000	0	000227	000000	186.25	
004042	GO TO	JAIL	004032	000052	000000	0100	21	620230	00	000000	0	000230	000000	191.25	
JAIL															
004032	LET (IF)	N	000006	000006	000000	0100	21	200226	00	000000	0	000226	000000	197.50	
004033	MINUS	1	000001	000005	000000	0100	21	240231	00	000000	0	000231	000000	203.75	
004034	YIELD	N	000005	000005	000000	0100	21	600226	00	000000	0	000226	000000	211.25	
004035	IS LESS	2	000002	000005	000000	0100	21	260232	00	000000	0	000232	000000	217.50	
004036	THEN GO	ANSWER	004043	000005	000000	0100	21	420233	00	000000	0	000233	000000	222.50	
004037	LET (IF)	RESULT	000052	000052	000000	0100	21	200227	00	000000	0	000227	000000	228.75	
004040	TIMES	N	000005	000322	000000	0100	21	440226	00	000000	0	000226	000000	292.50	
004041	YIELD	RESULT	000322	000322	000000	0100	21	600227	00	000000	0	000227	000000	300.00	
004042	GO TO	JAIL	004032	000322	000000	0100	21	620230	00	000000	0	000230	000000	305.00	

IC	INSTR	OPERAND	MOR	AC	EA	ACDO	S	MEM	P	SS	X	EAR	SL	TIME	I
JAIL															
004032	LET (IF)	N	000005	000005	000000	0100	21	200226	00	000000	0	000226	000000	311.25	
004033	MINUS	1	000001	000004	000000	0100	21	240231	00	000000	0	000231	000000	317.50	
004034	YIELD	N	000004	000004	000000	0100	21	600226	00	000000	0	000226	000000	325.00	
004035	IS LESS	2	000002	000004	000000	0100	21	260232	00	000000	0	000232	000000	331.25	
004036	THEN GO	ANSWER	004043	000004	000000	0100	21	420233	00	000000	0	000233	000000	336.25	
004037	LET (IF)	RESULT	000322	000322	000000	0100	21	200227	00	000000	0	000227	000000	342.50	
004040	TIMES	N	000004	001510	000000	0100	21	440226	00	000000	0	000226	000000	406.25	
004041	YIELD	RESULT	001510	001510	000000	0100	21	600227	00	000000	0	000227	000000	413.75	
004042	GO TO	JAIL	004032	001510	000000	0100	21	620230	00	000000	0	000230	000000	418.75	
JAIL															
004032	LET (IF)	N	000004	000004	000000	0100	21	200226	00	000000	0	000226	000000	425.00	
004033	MINUS	1	000001	000003	000000	0100	21	240231	00	000000	0	000231	000000	431.25	
004034	YIELD	N	000003	000003	000000	0100	21	600226	00	000000	0	000226	000000	438.75	
004035	IS LESS	2	000002	000003	000000	0100	21	260232	00	000000	0	000232	000000	445.00	
004036	THEN GO	ANSWER	004043	000003	000000	0100	21	420233	00	000000	0	000233	000000	450.00	
004037	LET (IF)	RESULT	001510	001510	000000	0100	21	200227	00	000000	0	000227	000000	456.25	
004040	TIMES	N	000003	004730	000000	0100	21	440226	00	000000	0	000226	000000	520.00	
004041	YIELD	RESULT	004730	004730	000000	0100	21	600227	00	000000	0	000227	000000	527.50	
004042	GO TO	JAIL	004032	004730	000000	0100	21	620230	00	000000	0	000230	000000	532.50	
JAIL															
004032	LET (IF)	N	000003	000003	000000	0100	21	200226	00	000000	0	000226	000000	538.75	
004033	MINUS	1	000001	000002	000000	0100	21	240231	00	000000	0	000231	000000	545.00	
004034	YIELD	N	000002	000002	000000	0100	21	600226	00	000000	0	000226	000000	552.50	
004035	IS LESS	2	000002	000002	000000	0100	21	260232	00	000000	0	000232	000000	558.75	
004036	THEN GO	ANSWER	004043	000002	000000	0100	21	420233	00	000000	0	000233	000000	563.75	
004037	LET (IF)	RESULT	004730	004730	000000	0100	21	200227	00	000000	0	000227	000000	570.00	
004040	TIMES	N	000002	011660	000000	0100	21	440226	00	000000	0	000226	000000	633.75	
004041	YIELD	RESULT	011660	011660	000000	0100	21	600227	00	000000	0	000227	000000	641.25	
004042	GO TO	JAIL	004032	011660	000000	0100	21	620230	00	000000	0	000230	000000	646.25	
JAIL															
004032	LET (IF)	N	000002	000002	000000	0100	21	200226	00	000000	0	000226	000000	652.50	
004033	MINUS	1	000001	000001	000000	0100	21	240231	00	000000	0	000231	000000	658.75	
004034	YIELD	N	000001	000001	000000	0100	21	600226	00	000000	0	000226	000000	666.25	
004035	IS LESS	2	000002	000001	000000	0110	21	260232	00	000000	0	000232	000000	672.50	
004036	THEN GO	ANSWER	004043	000001	000000	0100	21	420233	00	000000	0	000233	000000	677.50	
ANSWER															
004043	LET (IF)	RESULT	011660	011660	000000	0100	21	200227	00	000000	0	000227	000000	683.75	
004044	GO TO	FACTORIAL	004005	011660	000000	0100	21	620222	00	000000	0	000222	000000	688.75	
FACTORIAL															
004005	YIELD	N FACTORIAL	011660	011660	000000	0100	21	600215	00	000000	0	000215	000000	696.25	
*** 600001 IS SENT TO THE I/O UNIT.				INPUT DATA (000003)						3.0000000000 FROM DEVICE			1		696.25
004006	INPUT/OU	NUMBER OF IT	600001	000003	000000	0100	21	160210	00	000000	0	000210	000000	707.50	
004007	YIELD	M	000003	000003	000000	0100	21	600216	00	000000	0	000216	000000	715.00	
004010	LET (IF)	M	000003	000003	000000	0100	21	200216	00	000000	0	000216	000000	721.25	
004011	TRANSFOR	FACTORIAL	004012	000003	000000	0100	21	060222	00	000000	0	000222	000000	731.25	

IC	INSTR	OPERAND	MOR	AC	EA	ACDO	S	MEM	P	SS	X	EAR	SL	TIME	I
FACTORIAL															
004024	IS LESS	0	000000	000003	000000	0100	21	260224	00	000000	0	000224	000000	737.50	
004025	THEN GO	FACTORIAL	004012	000003	000000	0100	21	420222	00	000000	0	000222	000000	742.50	
004026	IS EQUAL	0	000000	000003	000000	0100	21	460224	00	000000	0	000224	000000	748.75	
004027	THEN GO	ANSWER EQUAL	004045	000003	000000	0100	21	420225	00	000000	0	000225	000000	753.75	
004030	YIELD	N	000003	000003	000000	0100	21	600226	00	000000	0	000226	000000	761.25	
004031	YIELD	RESULT	000003	000003	000000	0100	21	600227	00	000000	0	000227	000000	768.75	
004032	LET (IF)	N	000003	000003	000000	0100	21	200226	00	000000	0	000226	000000	775.00	
004033	MINUS	1	000001	000002	000000	0100	21	240231	00	000000	0	000231	000000	781.25	
004034	YIELD	N	000002	000002	000000	0100	21	600226	00	000000	0	000226	000000	788.75	
004035	IS LESS	2	000002	000002	000000	0100	21	260232	00	000000	0	000232	000000	795.00	
004036	THEN GO	ANSWER	004043	000002	000000	0100	21	420233	00	000000	0	000233	000000	800.00	
004037	LET (IF)	RESULT	000003	000003	000000	0100	21	200227	00	000000	0	000227	000000	806.25	
004040	TIMES	N	000002	000006	000000	0100	21	440226	00	000000	0	000226	000000	870.00	
004041	YIELD	RESULT	000006	000006	000000	0100	21	600227	00	000000	0	000227	000000	877.50	
004042	GO TO	JAIL	004032	000006	000000	0100	21	620230	00	000000	0	000230	000000	882.50	
JAIL															
004032	LET (IF)	N	000002	000002	000000	0100	21	200226	00	000000	0	000226	000000	888.75	
004033	MINUS	1	000001	000001	000000	0100	21	240231	00	000000	0	000231	000000	895.00	
004034	YIELD	N	000001	000001	000000	0100	21	600226	00	000000	0	000226	000000	902.50	
004035	IS LESS	2	000002	000001	000000	0110	21	260232	00	000000	0	000232	000000	908.75	
004036	THEN GO	ANSWER	004043	000001	000000	0100	21	420233	00	000000	0	000233	000000	913.75	
ANSWER															
004043	LET (IF)	RESULT	000006	000006	000000	0100	21	200227	00	000000	0	000227	000000	920.00	
004044	GO TO	FACTORIAL	004012	000006	000000	0100	21	620222	00	000000	0	000222	000000	925.00	
FACTORIAL															
004012	YIELD	M FACTORIAL	000006	000006	000000	0100	21	600217	00	000000	0	000217	000000	932.50	
004013	LET (IF)	N	000007	000007	000000	0100	21	200214	00	000000	0	000214	000000	938.75	
004014	MINUS	M	000003	000004	000000	0100	21	240216	00	000000	0	000216	000000	945.00	
004015	TRANSFOR	FACTORIAL	004016	000004	000000	0100	21	060222	00	000000	0	000222	000000	955.00	
FACTORIAL															
004024	IS LESS	0	000000	000004	000000	0100	21	260224	00	000000	0	000224	000000	961.25	
004025	THEN GO	FACTORIAL	004016	000004	000000	0100	21	420222	00	000000	0	000222	000000	966.25	
004026	IS EQUAL	0	000000	000004	000000	0100	21	460224	00	000000	0	000224	000000	972.50	
004027	THEN GO	ANSWER EQUAL	004045	000004	000000	0100	21	420225	00	000000	0	000225	000000	977.50	
004030	YIELD	N	000004	000004	000000	0100	21	600226	00	000000	0	000226	000000	985.00	
004031	YIELD	RESULT	000004	000004	000000	0100	21	600227	00	000000	0	000227	000000	992.50	
004032	LET (IF)	N	000004	000004	000000	0100	21	200226	00	000000	0	000226	000000	998.75	
004033	MINUS	1	000001	000003	000000	0100	21	240231	00	000000	0	000231	000000	1005.00	
004034	YIELD	N	000003	000003	000000	0100	21	600226	00	000000	0	000226	000000	1012.50	
004035	IS LESS	2	000002	000003	000000	0100	21	260232	00	000000	0	000232	000000	1018.75	
004036	THEN GO	ANSWER	004043	000003	000000	0100	21	420233	00	000000	0	000233	000000	1023.75	
004037	LET (IF)	RESULT	000004	000004	000000	0100	21	200227	00	000000	0	000227	000000	1030.00	
004040	TIMES	N	000003	000014	000000	0100	21	440226	00	000000	0	000226	000000	1093.75	
004041	YIELD	RESULT	000014	000014	000000	0100	21	600227	00	000000	0	000227	000000	1101.25	
004042	GO TO	JAIL	004032	000014	000000	0100	21	620230	00	000000	0	000230	000000	1106.25	

IC	INSTR	OPERAND	MOR	AC	EA	ACDO	S	MEM	P	SS	X	EAR	SL	TIME	I
JAIL															
004032	LET (IF)	N	000003	000003	000000	0100	21	200226	00	000000	0	000226	000000	1112.50	
004033	MINUS	1	000001	000002	000000	0100	21	240231	00	000000	0	000231	000000	1118.75	
004034	YIELD	N	000002	000002	000000	0100	21	600226	00	000000	0	000226	000000	1126.25	
004035	IS LESS	2	000002	000002	000000	0100	21	260232	00	000000	0	000232	000000	1132.50	
004036	THEN GO	ANSWER	004043	000002	000000	0100	21	420233	00	000000	0	000233	000000	1137.50	
004037	LET (IF)	RESULT	000014	000014	000000	0100	21	200227	00	000000	0	000227	000000	1143.75	
004040	TIMES	N	000002	000030	000000	0100	21	440226	00	000000	0	000226	000000	1207.50	
004041	YIELD	RESULT	000030	000030	000000	0100	21	600227	00	000000	0	000227	000000	1215.00	
004042	GO TO	JAIL	004032	000030	000000	0100	21	620230	00	000000	0	000230	000000	1220.00	
JAIL															
004032	LET (IF)	N	000002	000002	000000	0100	21	200226	00	000000	0	000226	000000	1226.25	
004033	MINUS	1	000001	000001	000000	0100	21	240231	00	000000	0	000231	000000	1232.50	
004034	YIELD	N	000001	000001	000000	0100	21	600226	00	000000	0	000226	000000	1240.00	
004035	IS LESS	2	000002	000001	000000	0110	21	260232	00	000000	0	000232	000000	1246.25	
004036	THEN GO	ANSWER	004043	000001	000000	0100	21	420233	00	000000	0	000233	000000	1251.25	
ANSWER															
004043	LET (IF)	RESULT	000030	000030	000000	0100	21	200227	00	000000	0	000227	000000	1257.50	
004044	GO TO	FACTORIAL	004016	000030	000000	0100	21	620222	00	000000	0	000222	000000	1262.50	
FACTORIAL															
004016	TIMES	M FACTORIAL	000006	000220	000000	0100	21	440217	00	000000	0	000217	000000	1326.25	
004017	YIELD	DENOMINATOR	000220	000220	000000	0100	21	600220	00	000000	0	000220	000000	1333.75	
004020	LET (IF)	N FACTORIAL	011660	011660	000000	0100	21	200215	00	000000	0	000215	000000	1340.00	
004021	OVER	DENOMINATOR	000220	000043	000000	0100	21	640220	00	000000	0	000220	000000	1446.25	
*** 400001 IS SENT TO THE I/O UNIT.			OUTPUT DATA (000043)							35.0000000000 TO IO DEVICE 1					1446.25
004022	INPUT/OU	NUMBER OF CO	400001	000043	000000	0100	21	160212	00	000000	0	000212	000000	1457.50	
004023	HALT		000000	000043	000000	0100	21	000000	00	000000	0	000000	000000	1457.50	

RUNNING TIME = .001 SECS
INSTRUCTIONS EXECUTED = 131

VERB	COUNT	INSTR TIME	TOTAL TIME
STEP SUBSCRIPT BY	0	6.25	.00
PLUS	0	6.25	.00
TRANSFORMED BY	3	10.00	30.00
SAVE EXTENSION IN	0	7.50	.00
EXECUTE	0	5.00	.00
SHIFTED BY	0	6.25	.00
INPUT/OUTPUT FROM	3	11.25	33.75
LET (IF)	26	6.25	162.50
IF SS ISNT GREATER THAN	0	6.25	.00
MINUS	12	6.25	75.00
IS LESS THAN	14	6.25	87.50
ANDED WITH	0	6.25	.00
SET SCALE WITH	1	5.00	5.00
CYCLED BY	0	6.25	.00
DOUBLE SHIFTED BY	0	6.25	.00
LET LOCATION	0	5.00	.00
THEN GO TO	17	5.00	85.00
TIMES	9	42.50	382.50
IS EQUAL TO	3	6.25	18.75
ORED WITH	0	6.25	.00
SET EXTENSION WITH	0	6.25	.00
USE SUBSCRIPT	0	5.00	.00
DOUBLE CYCLED BY	0	6.25	.00
YIELD	30	7.50	225.00
GO TO (RETURN FROM)	11	5.00	55.00
OVER	1	85.00	85.00
IS GREATER THAN	0	6.25	.00
EXCLUSIVELY ORED WITH	0	6.25	.00
RESUME FROM	0	21.25	.00
SAVE SUBSCRIPT IN	0	7.50	.00
*** ILLEGAL INSTRUCTION	0	.00	.00
HALT	1	.00	.00
IF OVERFLOW	0	3.75	.00
PASS	0	3.75	.00
IS POSITIVE	0	3.75	.00
NEGATED	0	5.00	.00
IF PARITY ODD	0	26.25	.00
PLUS CARRY	0	5.00	.00
RESET OVERFLOW	0	3.75	.00
COMPLEMENTED	0	5.00	.00
AND	0	3.75	.00
SET PAGE	0	3.75	.00
CLOSE EXTENSION WITH DEC	0	3.75	.00
NORMALIZED	0	3.75	.00
OR	0	3.75	.00
EXIT	0	35.00	.00
IS FALSE	0	3.75	.00