BROOKHAVEN NATIONAL LABORATORY

APPLIED MATHEMATICS DIVISION

Internal Report

PROGRAMMING SYSTEM FOR MERLIN*

by

Betty O. Weneser

September, 1959.

* Work performed under the Auspices of the U.S.Atomic Energy Commission

This report contains a description of the Programming System presently available for use with the Merlin Computer at Brookhaven National Laboratory.

The Programming System was designed by Mary T. Kresge and Betty O. Weneser, with the direction and encouragement of Milton E. Rose.

It is being developed and prepared for use by Mary Anne Kelley, D. Alan Ravenhall and Betty O. Weneser, with auxiliary programs done by Mary Louise Buchanan, Francis Ding Lee and Peter Mumford.

Suggestions for future additions to the system will be welcomed.

TABLE OF CONTENTS

Page

Section D. Subroutine Library

Section A. Introduction

1. General

Merlin can understand only a unique binary language and can interpret and execute only a specific set of instructions written in this language.

In order to prepare a problem for solution on this or any other computer, one might take the following steps:

1. Analyze the physical, mathematical, or logical problem and reduce its solution to a numerical or logical procedure.
2. Reduce this "procedure" to a sequence of instructions ("program"), or to a flow-diagram, which is written with knowledge of the computer and the operations it can perform, and yet is meaningful to the writer.
3. Translate this "program" into the coded language of the machine.

In order to simplify the coding process, however, it is possible to use the computer itself to help perform step (3) and part of step (2), outlined above. To accomplish this, a programming system for Merlin has been developed which will allow the user to communicate with the machine in a symbolic language which is meaningful to him and yet is translatable into machine language. This system, (called the Utility System) permits the user not only to write his program in the "middle language" of step (2) above, but also to debug and correct it and obtain results in the same symbolic language, and to have easy access to previously "debugged" programs.

Descriptions of the symbolic language and the entire system of utility programs which converts and produces from this language a machine code, ready to be executed, form the remaining sections of this report.

2. Structure of the Utility System

The utility system is a group of programs written in machine language and capable of understanding the symbolic language which is described in the following section. It is the function of these programs to interpret certain of the symbolic instructions and perform the task specified, and to translate other symbolic instructions into machine language; thus, the utility system has both interpreting and translating facilities.

The system consists of: 1) an overall interpreting routine called "System's Control Program"; 2) symbolic language translators called "Paper Tape Assembly Program", "Correction Program" and "Reassembly Program"; 3) a library of subroutines; and 4) auxilliary programs such as the "Symbolic Language Memory Dump Program", "Magnetic Tape Print Program", etc. Each of these parts will be described in Section C. In order to explain the system more easily, however, the section describing the symbolic language follows directly (Section B.)

Section B. Symbolic Language

1. Description

a. General

As explained in Section A, communication with Merlin can be done through the system of utility programs which understands the symbolic language and can translate it into machine language.

The symbolic language is made up of mnemonic representations (composed of alphabetic, numeric, and mathematical symbols and punctuation marks) for:

All the basic instructions to the computer

These instructions are translated or converted
by the Utility System directly into machine code.
They are described in Section B.2a.

An extension of the vocabulary of the computer in the form of instructions to the Utility System itself

These instructions are understood and interpreted
by the Utility System. They are described in Section B.2b.

1b. Structure of Symbolic Computer Instructions

(1) Format

The symbolic language for the computer instructions is designed to resemble the coding notes a programmer might use if he were writing in machine code. Each of these symbolic instructions represents one machine (binary) instruction word.

The symbolic program is written on prepared coding sheets of the following form:

Symbolic Location Field	Symbolic Instruction Field	Breakpoint Field

From this written form, a paper tape is punched, one line across at a time, the fields separated by TAB punches, and the lines themselves separated by CARRIAGE RETURN punches.

(2) The Instruction Field is of variable length and contains all the information about the operation, operands, and receiver.

Development:

Rules for writing these instructions are kept simple and yet consistent with all the characteristics of the computer. For example, in the four basic arithmetic operations, one operand is always in the U register; after the operation the result always appears in U and in the specified receiver if other than U. The fundamental form for writing all the basic arithmetic instructions symbolically becomes:

U	+ - * /	U Ti (m, b)	→	U Ti (m, b)
first operand	operation	second operand		receiver

where the programmer selects the particular registers and operation he wants. (see list of definitions of symbols, Section B.lc)

In general, any additional information follows the main part of the instruction, separated from it by a comma. For example:

U operation second operand → receiver,FX for fixed point operation

continued on next page

In multiplication, the form can include

U operation second operand → receiver,N for normalized result

or

U operation second operand → receiver,R for rounded result

or

U operation second operand → receiver,N,R for normalized and rounded result

Addressing within the Instruction Field:

Memory addresses may be either absolute decimal or symbolic. In either case, an address used within the instruction field must always be enclosed within parentheses.

In order to modify an address by the contents of a B- box register, the number of the B- box must follow the address, separated from it by a comma. Both the address and the B- box number then appear within the parentheses.

In the notation used in the List of Instructions (Section B.2a), addresses in the instruction field therefore appear as: (m, b). (See Definitions of Symbols, Section B.1c)

Symbolic addresses, composed by the programmer, are combinations of one to five "printer" characters, at least one of which must be non-numeric, andnone of which can be any of the following seven characters:

+ − * / ,) space character

Addresses may be modified by following them with +n or -n or *n or /n, where n is a decimal integer. More than one modifier may be used with an address; the interpretation of the modifiers will be made , in order, from left to right. If this modification is used, it must be specified before any B- box modification. e.g., (ALPHA +2, 3)

(3) The Breakpoint Field, when it is present, contains n, a decimal integer whose range is 0 thru 99. The usual way to indicate a Breakpoint in an instruction is to enter a 0 in its Breakpoint Field. If this field is present, the Breakpoint bit will be inserted in the binary instruction word. The Breakpoint is considered to be "numbered" if $n > 0$; when it is > 0, n will be inserted in the third dioctad of the binary instruction unless the instruction field has already filled the third dioctad.

If the Breakpoint Field is not present, the Instruction Field is followed directly by a CARRIAGE RETURN.

(4) The Location Field of an instruction line, unlike the other two fields, contributes nothing to the machine word. It contains the symbolic address, assigned by the programmer, of the location in which the word will be stored in memory.

Only those instructions which are referred to in another part of the program need be given a symbolic location. In all other instruction lines the Location Field may be left empty. If left empty, a TAB must nevertheless be punched on the paper tape before the Instruction Field is begun.

1c. Definitions of Symbols

In the development of the symbolic language given above, certain rules of notation, either expressly stated or implied, were used.

The following list contains the explanations of these and other symbols used in the Symbolic. Language. (See Section B.2a)

Symbol	Meaning
U	Universal register
R	Remainder register
A	Tag transfer Address Register
PF	PathFinder register
S	Storage register
SN	SeNse register
Ti Tj	Ti or Tj register ($1 \leq i, j \leq 4$) (In the list of instructions, Ti is used to designate a T register when it is an operand; Tj when it is a receiver. However Ti can equal Tj or not in all instructions using both)
Bi	B-box i register ($1 \leq i \leq 6$) (Used to designate a B-box when it is an operand or a receiver)
b b'	B-box b or b' register ($1 \leq b, b' \leq 6$) (used to designate a B-box when it modifies an address)
m m'	a symbolic or absolute decimal memory address

Symbol	Meaning
(m, b) (m', b')	a symbolic or absolute memory address, m or m', modified by the contents of b or b', and used in the instruction field.
n n'	a decimal integer
k, k',...	each k is a decimal integer ($1 \leq k \leq 16$), and represents one of the 16 bits of the Sense Register.
U2	the second dioctad of U
(m,b)2	" " " (m, b)
U3	the third dioctad of U
(m,b)3	" " " " (m, b)
,FX	Fixed point arithmetic
,N	Normalized result
,R	Rounded result
,L	Logical operation
+ - /	the usual arithmetic symbols
*	multiplied by
| |	absolute value
->	replaces the contents of

Notes on the use of the above symbols:

1. Throughout the List of Symbolic Computer Instructions lower case alphabetic characters are to be replaced by the programmer with the specific numbers or characters he is using. These are the only variable characters in the instructions. They are:

n, n', m, m', b, b', i, j, k, k'

2. Where a lower case alphabetic character represents a decimal integer (not an address), it can be represented symbolically in the form: Ni. See the Pseudo Operation EQUals in (B.2 b(1))

3. Where a lower case alphabetic character represents a variable to be computed and stored by the program a single 0 should be written in its place. No other character may be thus "omitted".

4. Spaces, in most places, will be ignored by the Utility System and can therefore be used by the programmer to aid legibility. The exception where spaces will not be ignored, and therefore may not be inserted, is within the name of a pseudo or macro instruction or immediately preceding it. This is so because the pseudo or macro instruction is defined by the first three characters in the Instruction Field immediately following the TAB.

Throughout the list of instructions (See Section B.2a) the following abbreviations are used in the Description column:

Symbol	Meaning
tag	tag 2 is tested
es	exponent spill is possible
mo	magnitude overflow is possible

The following list contains the explanations of notations used in the Machine Language column (Section B.2a):

Symbol	Meaning
p	a 16-bit representation of a specified sense light pattern, each bit (from right to left) representing one sense light (from 1 to 16). These sense lights so specified correspond to those designated by k, k', k",... in the symbolic language.
ffff	a dioctad composed of all 16 bits equal to "1"
-	denotes a part of an instruction word which is not examined by the computer. Each bit of such a part will be zero in the binary word. (See the pseudo operation: < n < n' or(m) < n'' or(m'') in Section B.2b(1) for an exception)
m m'	a binary memory address
b b' i j	the same as described in the symbolic language list

2 a

Index

to

Instructions to the Computer

(1) Arithmetic Instructions

(2) Load and Store Instructions

2a. Instructions to the Computer

(1) Arithmetic Instructions

Addition - Fixed Point

Machine Language			Symbolic Language	Description
a 0 - 0	0	-	U+U → U,FX	U + L → U and L'
a 0 - 0	i	-	U+Ti → U,FX	fixed point arithmetic
a 0 - j	0	-	U+U → Tj,FX	
a 0 - j	i	-	U+Ti → Tj,FX	mo
a 4 b 0	m	-	U+(m,b) → U,FX	U + X → U and N
a 4 b j	m	-	U+(m,b) → Tj,FX	fixed point arithmetic
a 4 b 5	m	-	U+(m,b) → (m,b),FX	mo,tag

Subtraction - Fixed Point

Machine Language			Symbolic Language	Description
a 2 - 0	0	-	U-U → U,FX	U - L → U and L'
a 2 - 0	i	-	U-Ti → U,FX	fixed point arithmetic
a 2 - j	0	-	U-U → Tj,FX	
a 2 - j	i	-	U-Ti → Tj,FX	mo
a 6 b 0	m	-	U-(m,b) → U,FX	U - X → U and N
a 6 b j	m	-	U-(m,b) → Tj,FX	fixed point arithmetic
a 6 b 5	m	-	U-(m,b) → (m,b),FX	mo, tag

Explanation on next page.

Explanation

This is an addition or subtraction of two signed numbers, U and the specified operand. The result appears in bits 0 to 40 of U, and the specified receiver if other than U, and in S. The sign is in bit 0. The exponent bits and both tag bits of the original U are carried into the result in U and the specified receiver if other than U, S, and R. The sign of the result (bit 0) is also the sign of R. The rest of R is cleared. The specified operand, unless it is also the receiver, is unchanged.

Note:

The magnitude overflow condition will be set if the addition or subtraction of the two operands results in a magnitude which is 1 or greater (the binary point considered to be between bits 0 and 1). The result then will be incorrect.

Addition - Floating Point

Machine Language			Symbolic Language	Description
a 1 - 0	0	-	U+U → U	U + L → U and L'
a 1 - 0	i	-	U+Ti → U	floating point arithmetic
a 1 - j	0	-	U+U → Tj	
a 1 - j	i	-	U+Ti → Tj	es
a 5 b 0	m	-	U+(m,b) → U	U + X → U and N
a 5 b j	m	-	U+(m,b) → Tj	floating point arithmetic
a 5 b 5	m	-	U+(m,b) → (m,b)	es, tag

Subtraction - Floating Point

Machine Language			Symbolic Language	Description
a 3 - 0	0	-	U-U → U	U - L → U and L'
a 3 - 0	i	-	U-Ti → U	floating point arithmetic
a 3 - j	0	-	U-U → Tj	
a 3 - j	i	-	U-Ti → Tj	es
a 7 b 0	m	-	U-(m,b) → U	U - X → U and N
a 7 b j	m	-	U-(m,b) → Tj	floating point arithmetic
a 7 b 5	m	-	U-(m,b) → (m,b)	es, tag

Explanation on next page

Explanation

On floating point additions and subtractions the signed exponents of the operands are compared and the exponent which is algebraically larger becomes the exponent of the result. After the comparison of the exponents the fractional part of the number with the smaller exponent (bits 1 to 40) is shifted right by shifts of 8 until the difference between the two exponents is satisfied. The shifted bits appear in R bits 1 to 40. If the exponents were equal no shifting takes place and R, bits 1 to 40, are cleared. After shifting, the fractional parts of the operands are algebraically combined. If there is an overflow on the fractional add, the exponent and fraction are adjusted; the magnitude overflow indicator is not set. However, exponent spill can occur. The result of this operation, magnitude and exponent, appears in U, and the specified receiver if other than U, and in S. The R register contains the same sign as the result, and bits 1 to 40 of R contain the low order bits of the result. The exponent of R is the exponent of the result. The tag bits as they appeared in U before the operation are carried into the result in all registers. The specified operand, unless it is also the receiver, is unchanged.

Note: The positive exponent spill condition will occur if the magnitude of the exponent equals or exceeds 2^4 and the sign of the exponent is positive.

Multiplication - (Unnormalized)

Machine Language			Symbolic Language	Description
a 8 - 0	0	-	U*U → U	U·L → UR and L'(high order)
a 8 - 0	i	-	U*Ti → U	unnormalized product
a 8 - j	0	-	U*U → Tj	
a 8 - j	i	-	U*Ti → Tj	es
a c b 0	m	-	U*(m,b) → U	U·X → UR and N(high order)
a c b j	m	-	U*(m,b) → Tj	unnormalized product
a c b 5	m	-	U*(m,b) → (m,b)	es, tag

Explanation

The exponents of the operands are algebraically added and this sum becomes the exponent of the result. The magnitudes of the operands are multiplied and a signed 80 bit product is obtained. This appears in bits 0 to 40 of U and is extended to bits 1 to 40 of R. The sign bit and the exponent of R are the same as those of the resulting U. The high order bits of this product and the exponent (the new contents of U) also go to the receiver if it is other than U, and to S. The tag bits which were in the original U are the tag bits of the result and will appear in all of the result registers. The specified operand, if other than U, is unchanged after the operation.

Note: This is the multiplication operation which is used for fixed point arithmetic. (see note on exponent spill)

Multiplication - Normalized

Machine Language			Symbolic Language	Description
a 9 - 0	0	-	U*U → U,N	U·L → UR and L' (high order)
a 9 - 0	i	-	U*Ti → U,N	product is normalized
a 9 - j	0	-	U*U → Tj,N	
a 9 - j	i	-	U*Ti → Tj,N	es
a d b 0	m	-	U*(m,b) → U,N	U·X → UR and N (high order)
a d b j	m	-	U*(m,b) → Tj,N	product is normalized
a d b 5	m	-	U*(m,b) → (m,b),N	es, tag

Explanation

This operation is the same as the multiplication operation except that there is a normalization step added. After the multiplication operation the fractional or magnitude part of the result, in U and R, is shifted left, in shifts of 8, until the first 8 bits of the fraction (U 1-8) are not zero. This is done for a maximum of 5 such shifts. The exponent of the result is adjusted downward by "1" for each shift of 8 needed to meet the above condition. Bits shifted out of bit 1 of R are shifted into bit 40 of U. Zeros are shifted into the R register from the right to replace shifted bits. The normalized result appears in U with the low order bits in R. The exponent of R and the sign of R is the same as the exponent and sign of the result. The high order part of the normalized result also goes to the specified receiver if it is other than U, and to S. The tag bits of the result are set equal to the tag bits of the original U. The specified operand, unless it is also a receiver, remains unchanged.

(see note on exponent spill)

Multiplication - Rounded (Unnormalized)

Machine Language			Symbolic Language	Description
a a - 0	0	-	U*U → U,R	U·L → UR and L'(high order)
a a - 0	i	-	U*Ti → U,R	product is rounded (Unnormalized)
a a - j	0	-	U*U → Tj,R	
a a - j	i	-	U*Ti → Tj,R	es
a e b 0	m	-	U*(m,b) → U,R	U·X → UR and N(high order)
a e b j	m	-	U*(m,b) → Tj,R	product is rounded (unnormalized)
a e b 5	m	-	U*(m,b) → (m,b),R	es, tag

Explanation

This operation is the same as multiplication except that a rounding step is added. After the multiplication operation has been completed, bit 1 of R is tested and if it is '1', a '1' is added to the high order part of the result in U in position 40. If bit one of R is '0' there is no change in the high order part. The rounded result then appears in U, and the specified receiver if other than U, and S. R is unaffected by the rounding. The specified operand if it is not a receiver is unchanged. (see note on exponent spill)

Multiplication - Normalized and Rounded

Machine Language			Symbolic Language	Description
a b - 0	0	-	U*U → U,N,R	U·L → UR and L' (high order)
a b - 0	i	-	U*Ti → U,N,R	product is normalized then rounded
a b - j	0	-	U*U → Tj,N,R	
a b - j	i	-	U*Ti → Tj,N,R	es, mo
a f b 0	m	-	U*(m,b) → U,N,R	U·X → UR and N (high order) product is normalized then rounded
a f b j	m	-	U*(m,b) → Tj,N,R	
a f b 5	m	-	U*(m,b) → (m,b),N,R	es, mo

Explanation

After multiplication of the 2 operands, normalization takes place and then rounding, each in the manner explained in the three preceding multiplication instructions. The result is in U, and the specified receiver if other than U, and S. The R register contains the low order part of the product after normalization, and the same sign and exponent as U. Again the tag bits of the original U are the tag bits of the result. The magnitude overflow condition will be set if overflow occurs on rounding. (see note on exponent spill)

Cumulative Multiplication

Machine Language			Symbolic Language	Description
e d b b'	m	m'	**(m,b)*(m',b')+T4 → T4**	X·X'+T4 → U and T4 product is normalized; addition is floating point es, tag

Explanation

The multiplication with normalization operation is performed on the two specified operands. After this the product which is in U and extended to R (bits 1 to 40) is added to the content of T4 by a floating point addition operation. The result of the cumulative multiplication appears in U, S, and T4 after the operation. The R register contains any shifted bits from the exponent adjustment on the floating addition. The sign and exponent of R are the same as the sign and exponent of the result. The tag bits of the first operand are the tag bits of the result. (see note on exponent spill)

Note: The tag 2 bit is tested on both memory operands and the tag 'on' condition will be met if either or both operands are tagged.

Division - Fixed Point

Machine Language			Symbolic Language	Description
d 0 - -	i	-	U/Ti → U,FX	U R ÷ T → U (rounded quotient) fixed point arithmetic remainder → R unrounded quotient → S mo
d 4 b -	m	-	U/(m,b) → U,FX	U R ÷ X → U (rounded quotient) fixed point arithmetic remainder → R unrounded quotient → S mo, tag

Explanation

The signed magnitude of U (bits 0 - 40) and its extension into R (bits 1 to 40) is divided by the signed magnitude of the specified operand. The result of this operation appears in the sign and magnitude of S (bits 0-40). This quotient is then rounded and the rounded quotient appears in U (bits 0-40) with the remainder in R (bits 1 to 40). The exponent and tag bits of U remain as before the operation. The exponent, tag bits, and magnitude sign of R remain as before the operation. The sign of the R register is therefore not the sign of the quotient. The exponent and tag bits of S are the same as those of U. The specified operand remains unchanged after the operation. For a fixed point divide the magnitude in U (the dividend) must be smaller than the magnitude of the other operand. If this is not so, magnitude overflow condition will be set and the answer will be incorrect.

Division - Floating Point

Machine Language			Symbolic Language	Description
d 1 - -	i	-	U/Ti → U	U R ÷ T → U (rounded quotient) floating point arithmetic remainder → R unrounded quotient → S es, mo
d 5 b -	m	-	U/(m,b) → U	U R ÷ X → U (rounded quotient) floating point arithmetic remainder → R unrounded quotient → S es, mo, tag

Explanation

The exponent of the specified divisor is subtracted from the exponent of U. Then the magnitude of U (bits 1 to 40) is adjusted by shifts of 8 until it is smaller than the magnitude of the divisor.[*] The exponent is adjusted accordingly. After the adjustment, the magnitude of U is divided by the magnitude of the other operand and this quotient (magnitude & exponent) appears in S. The quotient is then rounded and the rounded quotient appears in U with the remainder in R (bits 1 to 40). The exponent of R, the sign of R, and the tag bits of R remain as they were before the operation. The tag bits of U also remain as before the operation and appear in S. The specified operand is unchanged after the operation. Magnitude overflow can occur on rounding.

[*] If this adjustment cannot be made in at most 4 shifts of 8, then the insignificant divide condition is set. If the adjustment cannot be made in at most 5 shifts of 8, then the magnitude overflow indicator is set and no magnitude division takes place.

Square Root

Machine Language			Symbolic Language	Description
c 0 - -	-	-	SQ	$\sqrt{\|U\|} \rightarrow U$ result is rounded

Explanation

After this operation, U contains the rounded square root of the absolute value of the original U. S is cleared after the operation. The tag bits of the result are the same as the tag bits of the original U.

The square root operation may be used as both floating and fixed point operation since the operation takes the square root of the magnitude of U and divides the exponent by two. If the exponent is not a multiple of two an adjustment is made by a magnitude right shift of 4, where bits shifted out of position 40 of U are shifted into bit 1 of R. The rounding takes place after this adjustment.

Normalize

Machine Language			Symbolic Language	Description
c 1 - -	-	-	N	normalize UR → UR es

Explanation

The number in U and extended to R (bits 1 to 40) is normalized by left shifts of 8 until the most significant 8 bits of the magnitude are not 0. This is done for at most 5 shifts. The exponent is adjusted downward by 1 for each shift of 8 bits . S is cleared after the operation. The normalized result appears in U and is extended to R, bits 1 to 40. The rest of R remains the same as before the operation. The tag bits of U are unchanged by the operation. (see note on exponent spill)

Round

Machine Language			Symbolic Language	Description
c 2 - -	-	-	R	round UR → U mo

Explanation

The number in U and extended to R (in bit 1) is rounded and the result appears in U. If bit 1 of R is equal to '1', a '1' is added in bit 40 of U. If bit 1 of R is equal to '0' no change occurs. R is unchanged by the operation; S is cleared. Magnitude overflow may occur in this operation.

Normalize and Round

Machine Language			Symbolic Language	Description
c 3 - -	-	-	N,R	normalize UR; then round UR → U es, mo

Explanation

The number in U and extended to R is normalized as in the normalize operation. Then the normalized number is rounded as in the round operation. The normalized rounded result appears in U. The R register, in bits 1 to 40, contains the extension of U as formed by the normalizing operation. The rest of R is unchanged. S is cleared. The tag bits of the original contents of U are the tag bits of the result.

Absolute Value

Machine Language			Symbolic Language	Description
e 1 - 0	0	-	$\lvert U \rvert \rightarrow U$	$\lvert L \rvert \rightarrow U$ and L'
e 1 - 0	i	-	$\lvert Ti \rvert \rightarrow U$	(set sign plus)
e 1 - j	0	-	$\lvert U \rvert \rightarrow Tj$	
e 1 - j	i	-	$\lvert Ti \rvert \rightarrow Tj$	
e 5 b 0	m	-	$\lvert (m,b) \rvert \rightarrow U$	$\lvert X \rvert \rightarrow U$ and N (set sign plus)
e 5 b j	m	-	$\lvert (m,b) \rvert \rightarrow Tj$	
e 5 b 5	m	-	$\lvert (m,b) \rvert \rightarrow (m,b)$	tag

Explanation

The contents of the specified operand (U, a T register, or a memory location), with the sign of the magnitude set positive, is loaded into the U register and the specified receiver if other than U, and in S. The operand, unless it is also a receiver, is unchanged by the operation.

Negative Value

Machine Language			Symbolic Language	Description
e 2 - 0	0	-	**-U → U**	-L → U and L'
e 2 - 0	i	-	**-Ti → U**	(change sign)
e 2 - j	0	-	**-U → Tj**	
e 2 - j	i	-	**-Ti → Tj**	
e 6 b 0	m	-	**-(m,b) → U**	-X → U and N
e 6 b j	m	-	**-(m,b) → Tj**	(change sign)
e 6 b 5	m	-	**-(m,b) → (m,b)**	tag

Explanation

The contents of the operand (U, a T register, or a memory location), with the sign of the magnitude changed, is loaded into U and the specified receiver if other than U, and in S. The operand, unless it is also a receiver, is unchanged by the operation.

Negative Absolute Value

Machine Language			Symbolic Language	Description
e 3 - 0	0	-	$-\lvert U \rvert \rightarrow U$	$-\lvert L \rvert \rightarrow$ U and L'
e 3 - 0	i	-	$-\lvert Ti \rvert \rightarrow U$	(set sign minus)
e 3 - j	0	-	$-\lvert U \rvert \rightarrow Tj$	
e 3 - j	i	-	$-\lvert Ti \rvert \rightarrow Tj$	
e 7 b 0	m	-	$-\lvert (m,b) \rvert \rightarrow U$	$-\lvert X \rvert \rightarrow$ U and N
e 7 b j	m	-	$-\lvert (m,b) \rvert \rightarrow Tj$	(set sign minus)
e 7 b 5	m	-	$-\lvert (m,b) \rvert \rightarrow (m,b)$	tag

Explanation

The contents of the operand (U, a T register, or memory location), with the sign of the magnitude set minus, is loaded into the U register and the specified receiver if other than U, and into S. The operand, unless it is also a receiver, is unchanged by the operation.

(2) Load and Store Instructions

Load into U and/or T

Machine Language			Symbolic Language	Description
e 0 - 0	i	-	Ti → U	L → U and L'
e 0 - j	0	-	U → Tj	
e 0 - j	i	-	Ti → Tj	
e 4 b 0	m	-	(m,b) → U	X → U and N tag
e 4 b j	m	-	(m,b) → Tj	

Explanation

The contents of the operand (U, a T register, or memory location) is loaded into the U register and the specified receiver if other than U, and into S. The operand is unchanged by this operation.

Machine Language			Symbolic Language	Description
d 3 - 0	-	-	R → U	R → L
d 3 - j	-	-	R → Tj	
d 2 - 0	-	-	S → U	S → L
d 2 - j	-	-	S → Tj	

Explanation

The contents of the specified operand (S or R) is loaded into the U register and the specified receiver if other than U, and into S. The operand is unchanged by the operation.

Store into Memory

Machine Language			Symbolic Language	Description
b e b -	m	-	U → (m,b)	U → X
e c b b'	m	m'	(m,b) → (m',b')	X → X' tag
d 7 b -	m	-	R → (m,b)	R → X
d 6 b -	m	-	S → (m,b)	S → X

Explanation

The contents of the operand (U, R, S or memory location) is loaded into the specified memory location, and into S. The operand is unchanged by this operation.

Machine Language			Symbolic Language	Description
b 7 b 0	m	-	CLEAR(m,b)	set to 0 all bits of X

Explanation

All bits of the specified memory location, and of S, are set to zero.

Store 16 bits into Memory

From U

Machine Language			Symbolic Language	Description
d d b -	m	-	U2 → (m,b)2	U2 → X2 (X1 and X3 are unaltered)
d e b -	m	-	U3 → (m,b)3	U3 → X3 (X1 and X2 are unaltered)

Explanation

The contents of the second or third dioctad of U is deposited into the second or third dioctad of X. The rest of X is unchanged by either operation. Thus when X2 is the receiver X1 and X3 are unaffected and when X3 is the receiver X1 and X2 are unaffected. U is unchanged by these operations. S has the same contents as X after operation.

From Pathfinder

Machine Language			Symbolic Language	Description
d c b -	m	-	PF → (m,b)	PF → X2

Explanation

The contents of the pathfinder is loaded into the second dioctad of X. The first and third dioctads of X, and the pathfinder, are unchanged by the operation. S has the same contents as X after the operation.

From Tag Transfer Address (A)

Machine Language			Symbolic Language	Description
b f b -	m	-	A → (m,b)	A → X2 (X1 and X3 are cleared)

Explanation

The contents of the Tag Transfer Address register is deposited into the second dioctad of X. The rest of X is cleared. A is unchanged by the operation. S has the same contents as X after the operation.

Load R

Machine Language			Symbolic Language	Description
e 8 - -	0	-	U → R	L → R
e 8 - -	i	-	Ti → R	
d f b -	m	-	(m,b) → R	X → R

Explanation

The content of the specified operand (U, a T register, or memory location) is loaded into R. The operand is unchanged by this operation. No other register is affected.

Load Tag Transfer Address (A)

Machine Language			Symbolic Language	Description
b d b -	m	-	$(m,b) \rightarrow A$	$X2 \rightarrow A$

Explanation

The contents of the second dioctad of a memory location is loaded into the Tag Transfer Address register (A). X is unchanged by the operation. No other register is affected.

(3) B-Box Instructions

Load B

Machine Language			Symbolic Language	Description
b 0 - i	n	-	**n → Bi**	n → B
b 3 - i	-	-	**U → Bi**	U2 → B
b 4 b i	m	-	**(m,b) → Bi**	X2 → B

Explanation

The second dioctad of the operand (either the instruction itself, U, or a memory location) is loaded into the specified B box. After the operation, the second dioctad of S has the same contents as the specified B box and the rest of S is cleared. The operands are unchanged.

Note: If the B box specified is 0 no B box is affected; however the second dioctad of S contains the same number as the second dioctad of the operand.

Alter B

Machine Language			Symbolic Language	Description
b 1 - i	n	-	**Bi+n → Bi**	B + n → B
b 5 b i	m	-	**Bi+(m,b) → Bi**	B + X2 → B

Explanation

The contents of the specified B box is added to the second dioctad of the operand (either the instruction itself or a memory location) and stored into the B box specified. After the operation, the second dioctad of S has the same contents as the B box; the rest of S is cleared. The operand is unchanged. (see note about B = 0 following the previous instruction)

Store B

Machine Language			Symbolic Language	Description
b 7 b i	m	-	**Bi → (m,b)**	B → X2 (X1 and X3 are cleared)

Explanation

The contents of the B box specified is loaded into the second dioctad of X and the rest of X is cleared. After the operation, S has the same contents as X. The B box is unchanged. If B is specified as 0, X is cleared and S again has the same contents as X after the operation.

Alter B and Test

Machine Language			Symbolic Language	Description
b 2 - i	n	n'	**SKIF n'≤Bi+n**	B + n → B, then skip the next instruction if $n' \leq B$.
b 6 b i	m	-	**SKIF (m,b)3≤Bi+(m,b)2**	B + X2 → B, then skip the next instruction if $X3 \leq B$

Explanation

The contents of the second dioctad of the operand (either the instruction itself or a memory location) is added to the contents of the specified B box. The new contents of the B box is then compared to the contents of the third dioctad of the operand. If the contents of the third dioctad is less than or equal to the contents of the B box, the next instruction is skipped. If this condition is not met, control proceeds in sequence. The second dioctad of S, after the operation, has the same contents as the specified B box after the operation and the rest of S is cleared. The operand is unchanged by the operation. (see note on B = 0) .

Test B

Machine Language			Symbolic Language	Description
b 2 - i	0	n'	**SKIF n'≤Bi**	Skip the next instruction if $n' \leq B$

Explanation

The contents of the specified B box is compared to the contents of the third dioctad of the operand. If the contents of the third dioctad of the operand is less than or equal to the contents of the B box, the next instruction is skipped. If not, control proceeds in sequence. The second dioctad of S, after the operation, has the same contents as the B box and the rest of S is cleared. The operand is unchanged by the operation.

(4) Logical Instructions

Load U

Machine Language			Symbolic Language	Description
b c b –	m	–	(m,b) → U,L	X → U (The tag 2 bit is not tested.)

Explanation

The contents of the specified memory location is loaded into the U register, and S. The contents of the memory location remains unchanged. In this instruction, unlike the instruction"(m, b) → U"on page 33, the tag 2 bit of the operand is not tested.

Complement

f b – –	–	–	CØMP U	Complement of U → U
f c b –	m	–	CØMP(m,b)	Complement of X → U

Explanation

The one's complement of the contents of the operand (U or a memory location) replaces the contents of U and of S. When a memory location is the operand, its contents is unchanged by the operation.

INTERSECT - LOGICAL "AND"

Machine Language			Symbolic Language	Description
f 7 b -	m	-	AND(m,b)	intersection or logical "AND" of U and X → U.

Explanation

In the result of this operation, a '1' will appear in any bit position in which the corresponding bits of the contents of both U and the specified memory location are '1'. Any other bit position of the result will contain a '0'. This result appears in U. X is unchanged by the operation. After the operation, R contains the original U; S has the same contents as X.

UNION - LOGICAL "OR"

Machine Language			Symbolic Language	Description
f 4 b -	m	-	ΘR(m,b)	union or logical "OR" of U and X → U

Explanation

In the result of this operation, a "1" will appear in any bit position where either one or both of the corresponding bit positions in the contents of U and the specified memory location contain a "1". All other bit positions of the result contain a "0". This result appears in U. X is not altered by the operation. After the operation, R and S have the same contents as X.

SYMMETRIC DIFFERENCE - EXCLUSIVE OR

Machine Language			Symbolic Language	Description
f 6 b -	m	-	XØR(m,b)	symmetric difference or "exclusive OR" of X and U → U.

Explanation

In the result of this operation, a '1' will appear in any bit position in which the corresponding bits of U and X are *not* alike. A '0' will appear in any bit position in which the corresponding bits of U and X *are* alike. This result appears in U. X is not altered by the operation. R, after the operation, has the original contents of U. S, after the operation, contains the one's complement of the contents of X.

EXTRACT

Machine Language			Symbolic Language	Description
f 5 b -	m	-	EXT(m,b)	EXTract from X the bit pattern in R → U.

Explanation

The bits specified by the binary pattern in R are extracted from X and deposited in U as follows: a '1' in any position in R causes the corresponding bit of U to be set equal to the corresponding bit in X. The rest of U is unchanged. R and X are not altered by the operation. After the operation, S has the same contents as X.

(5) Shift Instructions

Magnitude —— Short Shifts

Machine Language			Symbolic Language	Description
f 0 - -	n	-	UnL	shift the magnitude of U n bits to the Left mo
f 8 - -	n	-	UnR	shift the magnitude of U n bits to the Right

Explanation

These instructions cause the contents of U, bits 1 to 40 only, to be shifted left or right n (mod 128) bits, respectively. The left shift causes '0's' to be shifted into the right side of U to replace shifted bits, and the bits shifted out of bit 1 of U to be lost. The right shift causes '0's' to be shifted into bit 1 of U to replace shifted bits, and bits shifted out of bit 40 to be lost. Note: The overflow indicator is set if a '1' is shifted left out of bit 1 of U.

Magnitude ——Long Shifts

Machine Language			Symbolic Language	Description
f 1 - -	n	-	**URnL**	shift the magnitudes of UR n bits to the Left mo
f 9 - -	n	-	**URnR**	shift the magnitudes of UR n bits to the Right

Explanation

These instructions cause the contents of U, bits 1 to 40 only, and the contents of R, bits 1 to 40 only, to be shifted, as if one register, left or right n (mod 128) bits, respectively. In the long shifts, bit 40 of U is "tied to" bit 1 of R and the shifting takes place left from R to U and right from U to R. As in the short shifts, "0"s replace the shifted bits and bits shifted out of the left of U or right of R are lost. Overflow indicator is set if a "1" is shifted left out of bit 1 of U.

Magnitude —— Shift and Count

Machine Language			Symbolic Language	Description
f 3 - i	-	-	S,C → Bi	Shift the magnitude of U left until bit 1 of $U \neq 0$, Counting B by 1 after each shift.

Explanation

This instruction causes the contents of U, bits 1 to 40, to be shifted left, until bit 1 of U is not equal to 0. Before shifting begins, the specified B box is cleared and during the shifting a "1" is added to the B box for each single shift necessary to satisfy the requirement that bit 1 of $U \neq 0$. A maximum of 40 shifts will be performed. After the operation the B box contains the number of shifts required, and the second dioctad of S has the same count. The rest of S is cleared.

Logical Shifts

Machine Language			Symbolic Language	Description
f 2 - -	n	-	**UnL,L**	shift <u>all</u> of U n bits to the Left
f a - -	n	-	**UnR,L**	shift <u>all</u> of U n bits to the Right

Explanation

These instructions cause the contents of the entire U register (bits -7 through 40)to be shifted left and right n (mod 128) bits, respectively. "0"s again replace the shifted bits on the left or right. Bits shifted out of bit -7 on a logical left shift are lost and bits shifted out of bit 40 of U on a logical right shift are lost. The overflow indicator is never set on these instructions.

(6) Transfer of Control Instructions

Unconditional Transfer

Machine Language			Symbolic Language	Description
c 4 b -	m	-	**TR(m,b)**	TRansfer control to X, unconditionally

Explanation

This instruction transfers control to the specified memory location by resetting the control counter to that location. The pathfinder will contain 1 plus the location of this instruction. No other registers are affected.

Conditional Transfer

Machine Language			Symbolic Language	Description
c 5 b -	m	-	TR(m,b)IF+	TRansfer control to X if U is positive
c 7 b -	m	-	TR(m,b)IF 0	TRansfer control to X if the magnitude of U is 0
c f b -	m	-	TR(m,b)IF 0,L	TRansfer control to X if U is a Logical 0 (all 48 bits = 0)

Explanation

These operations test the contents of U for the following conditions:

c 5 - tests for a positive sign (bit 0 = 0)

c 7 - tests for a zero magnitude (bits 1 thru 40 = 0)

c f - tests for <u>all</u> bits equal to 0 (bits -7 thru 40 = 0)

In each case if the condition tested for is found, a transfer of control to the specified memory location is effected and the pathfinder is set to 1 plus the address of this instruction.

If the condition is not met, there is no transfer and the pathfinder is not reset. In either case no other register is affected.

Conditional Transfer (Cont.)

Machine Language			Symbolic Language	Description
c 6 b -	m	-	TR(m,b)IF ΘV	TRansfer control to X if the overflow indicator is on, and turn off the indicator.
c c b -	m	-	TR(m,b)IF PES	TRansfer control to X if the positive exponent spill indicator is on, and turn off the indicator.

Explanation

These operations test the overflow and positive exponent spill indicators respectively. If the specified indicator is on, the indicator is turned off and control is transferred to the specified memory location. If the indicator is off, control proceeds to the next instruction in sequence. If a transfer does take place, the pathfinder is set to 1 plus the address of this instruction. No other register is affected.

(For transfers on tags, see Tag Instructions)

(7) Unconditional Skip Instruction

Machine Language			Symbolic Language	Description
b b - -	n	-	**SKIPn**	SKIP the next n instructions

Explanation

The contents of the second dioctad, n, of this instruction is added to the contents of the control counter and the sum is stored into the control counter, thus causing the n words following the SKIP instruction to be ignored during execution.

(conditional skips are listed with B-Box Instructions and Sense Instructions)

(8) Repeat Instruction

Machine Language			Symbolic Language	Description
e b - -	n	-	DΘn	DO the next instruction n times, increasing the effective left and right address of that instruction by the contents of B6 and B5 respectively, after each perform.

Explanation

This instruction causes the machine to execute the instruction which follows it n (mod 256) times. After each performance of the "following instruction", the second dioctad of the instruction is increased by the contents of B6.. If the instruction is one which uses the third dioctad, it is increased after each perform by the contents of B5.

Where the second or third dioctad is a memory reference the normal B-box address modification takes place before this special repeat mode modification.

(9) Stop Instruction

Machine Language			Symbolic Language	Description
9 0 - -	-	-	STΘP	STOP the computer.

Explanation

This instruction causes the control to stop the machine; however, the next instruction has been fetched from memory so that pushing the "start" button will cause the control to proceed again.

(10) Sense Instructions

Alter Sense

Machine Language			Symbolic Language	Description
b 8 - -	p	-	CLEAR SN k,k' ,k",...	Set bits k, k', k",... of the SeNse register = 0
b 8 - -	ffff	-	CLEAR ALL SN	set all bits of the SeNse register = 0
b 9 - -	p	-	SET SN k,k',k",...	set bits k, k', k",... of the SeNse register = 1
b 9 - -	ffff	-	SET ALL SN	set all bits of the SeNse register = 1

Explanation

The second dioctad, p, of the instruction contains 16 bits, one corresponding to each sense light. Light 1 is addressed by the rightmost bit of the second dioctad, light 2 by the second from right, ..., light 16 by the leftmost bit. For each bit in the second dioctad which is a "1", the corresponding sense light is set as directed by the instruction, i.e., b 8 causesthe light or lights to be turned off or set to 0; b 9 causes the light or lights to be turned on or set to 1.

Test Sense

Machine Language			Symbolic Language	Description
b a - -	p	-	**SKIF SN k,k',k",...**	SKip one instruction IF bits k, k', k",... of the SeNse register = 1.
b a - -	ffff	-	**SKIF ALL SN**	SKip one instruction IF ALL bits of the SeNse register = 1.

Explanation

If all sense lights addressed by a "1" in the second dioctad are on then the next instruction is skipped. If they are not on, control goes to the next instruction in sequence. After the operation, the second dioctad of S contains the binary pattern of the sense light register. The rest of S is cleared. The sense light register is unchanged. Note: If p = 0, the next instruction is always SKipped (see the following instruction).

Save Sense

b a - -	0	-	**SN → S,SKIP**	SeNse register → S2 and SKIP 1 instruction (S1 and S3 are cleared)

Explanation

The contents of the sense light register becomes the contents of the second dioctad of S and the rest of S is cleared. The next instruction is always skipped and the sense light register is unchanged.

(11) Tag Instructions

Alter

Machine Language			Symbolic Language	Description
c 8 - -	-	-	**CLEAR TAG1**	0 → Tag 1 bit of U
c 9 - -	-	-	**CLEAR TAG2**	0 → Tag 2 bit of U
c a - -	-	-	**SET TAG1**	1 → Tag 1 bit of U
c b - -	-	-	**SET TAG2**	1 → Tag 2 bit of U

Explanation

The first 2 operations deposit a "0" in the Tag 1 bit and Tag 2 bit of U respectively. The next 2 operations deposit a "1" in the Tag 1 and Tag 2 bits of U respectively. The rest of U is unchanged and no other registers are affected.

Test

Machine Language			Symbolic Language	Description
c d b -	m	-	TR(m,b)IF TAG1	TRansfer control to X IF TAG 1 bit of U = 1
c e b -	m	-	TR(m,b)IF TAG2	TRansfer control to X IF TAG 2 bit of U = 1

Explanation

These operations test the tag 1 and tag 2 bits, respectively. If the respective tag bit of U is set (equal to "1"), a transfer of control to the specified memory location takes place. If not, control proceeds to the next instruction in sequence. If control is transferred, that is, if the condition is met, the pathfinder will contain the address of this instruction plus one. The pathfinder is not altered if the transfer does not take place.

(12) Input-Output Instructions

Read Paper Tape

Machine Language			Symbolic Language	Description
9 4 b -	m	-	PT → (m,b)	Paper Tape read. Read 1 hexad character into the rightmost 6 bits of X and skip (unless parity failure occurs)

Explanation

Six bits from the paper tape are read, through the feeder stage and S, into the rightmost six bits of X. The rest of S and X are cleared. If the six bit character is the End of Block symbol, the paper tape will stop (the computer will not). If the character is a delete, it will not be read into S or X (the paper tape will not stop). If parity failure occurs, control goes to the next instruction in sequence; otherwise the next instruction is skipped.

Punch Paper Tape

Machine Language			Symbolic Language	Description
9 5 b -	m	-	(m b) → PT	Paper Tape punch. Punch 1 hexad character from the leftmost 6 bits of X

Explanation

The contents of X goes to S, and the leftmost 6 bits of S are shifted through the feeder stage and punched on to paper tape.

Stop Paper Tape

9 3 - -	-	-	STOP PTR	stop Paper Tape Reader

Explanation

This stops the motion of the paper tape, and does nothing else.

Flexowriter

Machine Language			Symbolic Language	Description
9 8 - -	-	-	S → FL,4	FLexowrite, tetrad mode. Type the contents of S on the console typewriter.
9 9 - -	-	-	S → FL,6	FLexowrite, hexad mode. Type the contents of S on the console typewriter.

Explanation

These two instructions take the contents of S and by shifts of 4 or 6 bits into the feeder stage type out tetrad or hexad characters, respectively, on the flexowriter. Tetrad mode types out twelve characters, hexad mode types out eight characters.

Fast Printer

Machine Language			Symbolic Language	Description
9 c b -	m	-	(m,b)32 → PR	PRint columns 1-48 using the 32 character print matrix starting at X.
9 e b -	m	-	(m,b)64 → PR	PRint columns 1-48 using the 64 character print matrix starting at X.

Explanation

These instructions assume that a print matrix* of 32 or 64 words in length has been set up in memory starting at the specified memory address. The contents of this matrix determines one line of 48 columns of printed output. S is destroyed during the execution of these instructions.

Machine Language			Symbolic Language	Description
9 d b b'	m	m'	(m,b)(m',b')32 → PR	PRint columns 1-96 using two 32 character print matrices starting at X and X'
9 f b b'	m	m'	(m,b)(m',b')64 → PR	PRint columns 1-96 using two 64 character print matrices starting at X and X'

Explanation

These instructions assume that two print matrices*, each of 32 or 64 words in length, have been set up in memory, the first matrix starting in the first memory location specified and the second matrix starting in the second memory location specified. The contents of the first matrix determines the first 48 columns in the line of printed output, and the second determines columns 49-96. S and R are destroyed during execution of these instructions.

* Explanation of "print matrix" is given in <u>Report on Merlin</u> (Section 4.2).

Fast Printer (Cont.)

Machine Language			Symbolic Language	Description
9 b - -	-	-	**BLANK LINE**	Advance one line on the fast printer

Explanation

As the result of this instruction, the fast printer advances the paper one row, thus creating a blank line. S and R, and all other registers, remain unchanged.

Magnetic Tapes

Note: The magnetic tape instructions are omitted provisionally from this report because magnetic tapes are not yet available for use.

2b. Instructions To The Utility System

The instructions to the utility system are included in this section along with the instructions to the computer so that the entire list of Symbolic Instructions appears together and as such can be used as a reference list. The functions of these instructions are explained in detail in Section C (Utility System).

(1) Pseudo Instructions to the Assembler

These pseudo instructions are instructions to the assembler to take certain actions which may or may not result in one or more machine language words. Some of them may have a symbolic address in the location field; some may not. None of them take a breakpoint field. The following is a list of pseudo instructions to the assembler

NOTE: The particular designation given to the pseudo instruction cannot be broken up by spaces; thus DEC is acceptable, D EC is not. See the rules on spaces (Note 4 following Definitions of Symbols (B.1c)).

Symbolic Language	Explanation
ABS x	ABSolute at x, where x is a decimal memory location. This is an instruction to the assembler to produce the machine language program with an origin at x and with all address references located absolutely. If this instruction is used, it must precede any other instructions in the symbolic program. This instruction cannot have a symbolic address in the location field.

NOTE: Any symbolic program which does not use this instruction will be assembled as a relocatable machine language program.

Symbolic Language	Explanation
PAP	PAPer tape output. This is an instruction to the assembler to produce the loading routine and machine language program on paper tape as well as magnetic tape. This instruction does not result in a machine language instruction and therefore cannot take a symbolic address in its location field.

Symbolic Language	Explanation
REM k	REMark k where k is any statement written in characters which can be translated to printer-coded characters. This is an instruction to the assembler to print the remark as part of the symbolic listing of the program. The instruction results in no machine words. It cannot take a symbolic address in the location field.

Symbolic Language	Explanation
DEC ±d,±d',±d",...	DECimal ± d where d is a decimal number of several specified forms. This is an instruction to the assembler to convert the decimal numbers to binary numbers. Each decimal number must be separated by a comma from the next number. The converted numbers will occupy consecutive memory locations. The several forms of the decimal number and their resulting binary forms are as follows:

d	machine word
integer (no decimal pt) eg 5, + 20, -3	fixed point binary with the binary point assumed after bit 40.
fixed or floating decimal followed by B and a number specifying the position of the binary point in the machine word eg +5 B 4, -6.2B3 .6+2 B20	fixed point binary with the binary point after the bit specified by the number following B.
fixed or floating decimal (with either a decimal point or an exponent) eg -5.6,6-3, .6+4.	normalized floating point binary.

(continued on next page)

Symbolic Language

Explanation (cont.)

The plus sign may be omitted from the magnitude of the number but must always be used to express a positive exponent.

Any one of the numbers following DEC can be followed by a T1, T2, or T3 before the comma. These have the following meaning to the assembler:

T1 means set tag 1 bit
T2 means set tag 2 bit
T3 means set both tag bits

This instruction can take a symbolic address in the location field; the symbolic location will be assigned to the machine word corresponding to the first number.

Symbolic Instruction	Explanation
UFD ±d,±d',±d",...	Unnormalized Floating Decimal where d is a decimal number. This is an instruction to the assembler to convert the number or numbers to unnormalized floating point binary numbers and store them into consecutive memory locations. Each decimal number must be separated from the next by a comma. The converted numbers will occupy consecutive memory locations. The input form is a fixed or floating point decimal number e.g., .0006, -6, +25.2-10, 6.200. The number will be converted so that the last bit of the last converted decimal digit is at bit 40 in the machine word. No normalization takes place. Here again the sign of the number if it is plus may be omitted. The sign of the exponent may not. The tag bits of any machine word corresponding to any of the numbers can be set in the same manner as specified under the instruction DEC. The location field preceding this instruction field may contain a symbolic address and this address will be assigned to the first of the converted numbers.

Symbolic Language	Explanation
TET t,t',t",...	Machine language code t, t', t'',...., where t is a group of at most twelve TETrads (hexadecimal characters) representing a word in machine language. Each group of characters is separated from the next by a comma and each group results in one machine word. The groups of characters are set up in consecutive memory locations. Since the machine words are set up from left to right, the first character of any group will occupy the first four bits of the word; following characters are set up in succeeding bits by fours. If the right of a word is meant to be zero, the programmer need not write the zero characters in the group; thus f is the same as f00000000000. This instruction may take a symbolic address in the location field and it will be assigned to the first group of characters in the list.

Symbolic Language	Explanation
<n<n' or (m)<n" or (m')	This is an instruction to the assembler to set up a machine word such that the binary representation of n is the first dioctad; the locations equivalent to m and m' or the binary representations of n' and n'' are the second and third dioctads. Since the word is set up from left to right, there is no need to include the third dioctad if it is meant to be 0, or the second and third dioctads if they are both meant to be 0. The first dioctad, however, must always be present even if it is 0, i.e., < < n will set 0 into the first and third dioctads and n into the second. This instruction may take a symbolic address in the location field.

This pseudo instruction may be written after any symbolic computer instruction. The assembler will then combine into one computer word, by means of a logical union, the bits of the two instructions.

Symbolic Language	Explanation
FCC fc	Flexowriter Coded Characters fc where fc is a group of eight flexowriter characters. These eight characters will be set up as one machine word. Eight characters must be specified as failure to will cause the omission of the next instruction. This instruction can take a symbolic address in the location field.

Symbolic Language	Explanation
PCC c	Printer Coded Characters c where c is a group of characters which appear on the print wheel. This is an instruction to the assembler to convert all the characters of c to printer-coded characters and to set these characters into machine words - eight to one word. This instruction will result in α machine words for a group of β characters, where $\alpha = \beta/8$ when β is a multiple of 8, and $\alpha =$ (integral part of $\beta/8$)+ 1 otherwise. In this latter case blank characters will fill the remaining right bits of the last word. This instruction may have a symbolic address in the location field and this address will be assigned to the first machine word formed.

Symbolic Language	Explanation
BLƏ n	BLOck n where n is a decimal integer. This is an instruction to the assembler to reserve a block of storage n words long. This instruction can take a symbolic address in the location field; it will be assigned to the location of the first word of the block.

Symbolic Language	Explanation
LIB	LIBrary 2 insert. This is an instruction to the assembler to pick-up the library program specified by the symbolic address in the location field of this instruction from the library 2 tape and insert it into the program at this point.

Symbolic Language	Explanation
STA(m)	STArt at (m) where m can be a decimal address or a symbolic address which is defined by some other instruction in the program. This instruction tells the assembler to set up an instruction in such a way as to cause the loading program to transfer to address m when the machine language program has been loaded. This pseudo instruction does not result in a word which is part of the machine language program and can therefore not take an address in the location field.

Symbolic Language	Explanation
ΘVE(m)	OVErlap (m) where m is a symbolic address which has been previously defined by some other instruction in the program. This is an instruction to the assembler to set the location counter to m, allowing the programmer to set up the program which follows it at the same place in memory as some preceding part. It is useful for "breaking-up" programs which exceed the length of memory. Since this instruction merely causes the assembler to reset the location counter, it does not result in a machine word and therefore cannot take an address in the location field.

Symbolic Language	Explanation
DEF(m)	DEFined by (m) where m is a decimal address or a symbolic address which has been previously defined. This is an instruction to the assembler either to assign to a symbol the memory location already assigned to another, or to assign an absolute decimal location to a symbolic address. This instruction must have a symbolic address in the location field, since the purpose of the instruction is to define this symbol. No machine language word results from this instruction.

Symbolic Language	Explanation
EQUn	EQUals n where n is a decimal integer. This is an instruction to the assembler to assign the binary equivalent of n to a symbolic integer. This symbolic integer appears in the location field in the specific form: Ni where i is an integer from 1 to 9. This instruction is used to define an integer which is constant in the program but is to be used symbolically throughout the program. The instruction does not result in a machine word.

(2) Macro Instructions to the Assembler

Macro Instructions are instructions to the assembler which when translated result in more than one machine language instruction. At present all of these instructions are concerned with the use of the main subroutine library. Additions to the list of macro instructions will be made as the programming system is used and as the subroutine library increases.

Since at present these instructions do refer to subroutines, the use of a macro operation causes the assembler to insert the calling sequence to the specified library subroutine into the program at the point the macro instruction is used and to add the subroutine itself to the program.

These instructions may have a symbolic address in the location field. They do not have a breakpoint field.

The following is a list of macro instructions so far included in the system. Complete instructions for using any of them are given in the write-up for the particular subroutine referred to.

NOTE: The particular designation given to the macro instruction cannot be broken up by spaces; thus SIN is acceptable, S IN is not. See the rule on spaces (Note 4 following Definitions of Symbols (B.1c)).

Symbolic Language	Explanation
SIN	SINe of U → U
COS	COSine of U → U
ART	ARcTangent of U → U
NLG	Natural LoG of U → U
EXP	EXPonential of U → U

All of the above refer to commonly used elementary functions. They all assume floating point arithmetic; however, fixed point arguments may be used and fixed point operations may be obtained by adding, ",FX" to any of the instructions, e.g.: SIN,FX. Before any of the instructions are used the argument must be placed in U.

Symbolic Language	Explanation
LST(m)form,c	print Line SeT from (m) where m is a symbolic or decimal address. This instruction causes the assembler to set up the appropriate calling sequence to the print Line SeT subroutine. This subroutine takes the word in "m", converts it according to the specified "form" and stores the converted characters into a block of memory registers called the print line block, such that the rightmost character is set in the position corresponding to column "c" of the print line. This print line block is the one used in the Line PRint subroutine. After a complete line has been set up as the programmer wishes, he then uses the LPR instruction, which produces the calling sequence to the Line PRint subroutine, to do the printing. A list of "form" expressions follows:

Symbolic Language — Explanation

LST(m)I → I,c

This "form" specifies that the binary Integer stored at location m is to be converted to a decimal integer and the printer-coded characters for this decimal number are to be set into the print line.

LST(m)FXb → FXd,c

This "form" specifies that the FiXed point binary number stored at location m is to be converted to a FiXed point decimal number with d places to the right of the decimal point and the printer-coded characters for this decimal number are to be set into the print line. In conversion the binary number is to be assumed to have a binary point after bit b.

LST(m)FL → FXd,c

This "form" specifies that the FLoating point binary number at location m is to be converted to a FiXed point decimal number with d places to the right of the decimal point and the printer coded characters for this number are to be stored into the print line.

LST(m)FXb → FLd,c

This "form" specifies that the FiXed point binary number at location m is to be converted to a FLoating point decimal number with d places to the right of the decimal point and the print-characters stored into the print line. The binary point is to be assumed after bit b of the binary number.

Symbolic Language	Explanation
LST(m)FL → FLd,c	This "form" specifies that the FLoating point binary number at location m is to be converted to a FLoating point decimal number with d places to the right of the decimal point and the printer-coded characters for this number are to be set into the print line.
LST(m)TXn,c	This "form" specifies that the n printer-coded characters of TeXt stored in memory starting at location m are to be set into the print line.

NOTE: Floating point decimal numbers are defined here to have the form x.xx ··· ± xx and are distinguished from fixed point numbers only by their format.

Symbolic Language | Explanation

LPR

Line PRint. This instruction causes the assembler to set up a calling sequence to the Line PRint subroutine such that a print line will be printed on the fast printer. LPR follows one or more LST instructions which have caused a print line to be constructed.

BPR(m)n,n' ,form

Block PRint starting at (m) where m is a symbolic or decimal address. This instruction causes the assembler to produce a calling sequence to the Block PRint subroutine specifying that a block of n numbers located in memory starting at location m is to be printed, having a format of n' words per line. Each number is to be converted as expressed in "form".

There are 5 expressions of "form" which are identical to the first five described in the LST instruction. There is no form equivalent to TXn.

Symbolic Language	Explanation
PHD(m)n	Print Hexadecimal Dump of memory starting at m where m is a symbolic or decimal address. This instruction causes the assembler to produce a calling sequence to the Print Hexadecimal Dump subroutine specifying that n words starting at location m are to be converted from binary to hexadecimal characters and printed on the fast printer.
THD(m)n	Type Hexadecimal Dump is the same as PHD (m) n, except that the dump is typed on the flexowriter.
TDD(m)n	Type Decimal Dump of memory starting at m where m is a symbolic or decimal address. This instruction causes the assembler to produce a calling sequence to the Type Decimal Dump subroutine specifying that n words starting at location m are to be converted from floating point binary numbers to floating point decimal numbers and then typed on the flexowriter.

Symbolic Language — Explanation

INP(m) type n

Input from paper tape to memory starting at m where m is a symbolic or decimal address. This instruction causes the assembler to produce a calling sequence to the INPut subroutine which specifies that, in general, n words of data from paper tape, converted according to "type" of input, are to be read into memory starting at location m.

INP(m) type n

The various "types" of input, listed below, cause the assembler to alter the calling sequence and in this way give the needed information to the subroutine.

"type" of input — Meaning

DEC

Data on paper tape consists of decimal numbers to be converted to binary in the same way the pseudo instruction DEC performs this conversion. Each number on the paper tape is separated from the next by a comma, and the last number is followed by the End-of-Blocksymbol. The routine will read either n words or until End-of-Block, whichever comes first. n need not be specified at all in which case the routine will read until End-of-Block.

PCC

Data on paper tape consists of Printer-Coded Characters and they are treated in the same way the pseudo instruction PCC treat them. There is no separation between characters (spaces are considered to be characters), but the last is followed by the End-of-Block symbol. The routine will read n characters or until an End-of-Block, whichever comes first. If n is not specified, characters will be read until End-of-Block. If necessary, space characters will be added after the last PCC character to make n (or the number of PCC characters before E.O.B.) a multiple of 8.

"Type" of input	Meaning
TET	Data on paper tape consists of groups of hexadecimal characters, at most 12 to a word, each word separated from the next by a comma, and the last followed by the End-of-Block symbol. If there are fewer than 12 tetrads in any group, the routine will fill in the remaining right bits with zeros, as the pseudo instruction TET. The routine will read tetrads for n words or until an End-of-Block, whichever comes first. If n is not specified, words will be read until an End-of-Block.
FCC	Data on paper tape consists of at most 8 characters to be stored into one memory location in the way the pseudo instruction FCC does this. If fewer than 8 characters appear, the routine will fill in space characters into the remaining right bits of the word. No n should be specified, but an End-of-Block must follow the last character.
UFD	This "type" is the same as DEC except for the form of decimal-to-binary conversion. The conversion is explained in the pseudo instruction UFD.

(3) Instructions to the Corrector

The instructions to the corrector do not result in instructions in the machine language program. They are merely instructions which tell the corrector what kind of action to take in altering a symbolic program on the programmer's magnetic tape. The location fields of these instructions must always be blank and there is no breakpoint field.

Symbolic Language	Explanation
DELETEnLINEp	Delete n symbolic instructions starting at line p, where n is a decimal integer and p is the number of a line of symbolic code as specified on the printed listing of the symbolic program being corrected. This instruction, unlike the other instructions to the corrector, cannot be followed by symbolic instructions to the assembler. (machine, pseudo, or macro)

Symbolic Language	Explanation
INSERTnLINEp	Insert n symbolic instructions after line p in the symbolic program being corrected, where n is a decimal integer specifying the number of symbolic instructions to be inserted and p is the number of the line of symbolic code after which they are to be inserted. This instruction must be followed by n symbolic instructions to the assembler. (machine, pseudo or macro)

Symbolic Language	Explanation
CHANGEnLINEp	Change n symbolic instructions starting at line p in the symbolic code, where n is the number of symbolic instructions to be altered and p is the line number of the first line of symbolic code to be changed. This instruction must be followed by n symbolic instructions to the assembler. (machine, pseudo, or macro)

(4) Instructions to the Systems Control Program

The following instructions are interpreted by the System's Control Program. They specify actions which are to be taken by this overall control. The instructions have only one field. The location and breakpoint fields *are absent*, not merely blank. Each instruction is followed by an End-of-Block character.

Symbolic Language

Explanation

P T ASSEMBLE

ASSEMBLE the symbolic program which follows on Paper Tape.

Whatever follows this instruction either on the same paper tape or on another paper tape must be a program written in symbolic language. The instruction causes the systems control program to search the utility tape for the paper tape assembler program, read it into the memory and transfer control to it.

Symbolic Language	Explanation
CORRECT	CORRECT the symbolic program on the Programmer's Magnetic Tape using the symbolic corrections which follow this instruction on paper tape. What follows this instruction must be a paper tape of symbolic corrector instructions. This instruction causes the corrector to be searched for on the utility tape. When it is found and read into memory, control is transferred to it.

Symbolic Language	Explanation
EXECUTE	Load into memory and EXECUTE the machine language program on the Programmer's Magnetic Tape. This instruction causes the Systems Control program to search for the loader on the Programmer's Magnetic Tape, to read it into memory and transfer control to it.

Several other System's Control instructions have been proposed. They are intended for use with auxiliary programs. Examples of the proposed instructions follow:

Symbolic Language	Explanation
HEX DUMP	Print the contents of memory in hexadecimal on the fast printer.
DECIMAL DUMP	Print the contents of memory as floating decimal numbers on the fast printer.
SYMBΘLIC DUMP	Print the contents of memory in symbolic language on the fast printer.
RENUMBER	Renumber the lines of the symbolic program on the programmer's magnetic tape.

Section C. Utility System

The structure of the system, introduced earlier in Section A.2, is outlined in greater detail in this section.

1. Symbolic Language Translators

a. Paper Tape Assembly Program

This program will accept a symbolic language program from paper tape and convert it to a machine language program.

(1) Input

The input to the Paper Tape Assembler is a paper tape containing symbolic instructions, written in the format specified in section .B1. These instructions can be any of the symbolic machine language instructions, pseudo instructions or macro instructions all listed in section B2. The symbolic program on paper tape must always start with an assigned program numbe followed by a carriage return. The last line must be followed by an end of block mark (EOB).

(2) Output

This program will produce the following output:

(a) a Programmer's Magnetic Tape containing the symbolic program, symbol table, and self-loading machine language program in relocatable binary.*

(see pseudo instruction ABS for exception)

* A relocatable binary program is a binary program which may, at the operator's discretion, be loaded into different sections of memory. This is achieved by modifying each location used by the location which will be specified on the console when the program is being loaded.

(b) a printed listing of the symbolic program with each line numbered, the machine language program in hexadecimal, and a list of errors. (see diagram on next page)

(c) a paper tape, if requested, (see pseudo instruction PAP) of the self loading machine language program.

(3) Internal Operation

(a) Each symbolic instruction from paper tape is converted from flexo-coded characters to printer coded characters* and each instruction is given a line number. This converted and numbered program is written onto magnetic tape.

(b) Memory locations are assigned to the symbolic instructions and a table of symbolic locations vs. memory addresses is set up (Symbol Table). This symbol table is written onto magnetic tape.

(c) A "Loader"; that is, a program to load the machine language program from the magnetic tape into memory before operation, is written onto the Programmer's Magnetic Tape.

(d) Requested subroutines (see macro and pseudo instructions) are "picked up" from the subroutine library tape, their address references are adjusted, and they are written onto the Programmer's Magnetic Tape.

* Flexo-coded characters are six bit characters punched by depressing proper keys on the flexowriter. Printer coded characters are 6 bit characters which are used to set up a matrix for printing on the fast printer.

Printed Listing of Assembled Program

```
                HEXADECIMAL      HEX.  E   SYMB.                                                               BP
 LINE #         INSTRUCTION      LOC.  R   LOC.                 SYMBOLIC INSTRUCTION                          #
                                       R
                                       O
                                       R
                                       S
XXXXX.XX  XXXX XXXX XXXX  XXXX  XX  XXXXX  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  XX
```

(e) The symbolic instruction and breakpoint fields of the symbolic instructions are translated to produce machine language instructions. These also are written onto the Programmer's Magnetic Tape.

(f) A printed listing of the original symbolic program with each line numbered, the machine language program in hexadecimal, and a list of definite errors and possible errors is produced.

b. Corrector Program

Corrections to a symbolic program on the Programmer's Magnetic Tape* can be made by use of the Corrector. This program accepts a paper tape with corrections and directions for making the corrections and produces a corrected symbolic program plus a self-loading machine language program.

(1) Input

The input to the Correction Program is:

(a) A paper tape containing Instructions to the Corrector and the corrections themselves in symbolic assembly language (symbolic machine instructions, pseudo instructions and macro instructions). The first line on this paper tape must be the assigned program number followed by a carriage return. An End of Block symbol must end the paper tape.

(b) Programmer's Magnetic Tape containing the symbolic program to be corrected.

* A symbolic program on magnetic tape is one which has been produced by the paper tape assembler or by previous use of the correction program.

(2) Output

The output from the Corrector is

(a) a Programmer's Magnetic Tape containing the corrected symbolic program, symbol table, and the self-loading machine language program.

(b) a printed listing of the contents of the correction paper tape, the new symbolic program, the machine language program in hexadecimal and a list of errors.

(c) a paper tape, if requested.

(3) Internal Operation of the Correction Program

(a) The symbolic instructions on the correction paper tape are converted from flexo-coded characters to printer-coded characters and written on magnetic tape. The Instructions to the Corrector and the corrections which follow them are sorted according to line number.

(b) Each Instruction to the Corrector is interpreted by the Corrector and the direction specified by it is followed with respect to the old symbolic program on magnetic tape. Thus the lines of the old symbolic program are either deleted or changed or new lines are added to produce a corrected symbolic program on the Programmer's Magnetic Tape.

(c) Once the new symbolic program has been produced by the Corrector, the translation process has reached the same point as that reached by the Paper Tape Assembly Program when the instructions from paper tape have been converted and written on magnetic tape. Hence the remaining steps taken by the Corrector are identical to steps (b) through (f) performed by the Paper Tape Assembler and these steps are often referred to as the Assembler.

c. Reassembly Program

It is often necessary to make short corrections to a symbolic program which has already been assembled, that is, a symbolic program for which a self-loading machine language program already exists. If these corrections can be made directly to the existing machine language program, reassembly time will be saved. It is convenient to make the corrections in symbolic language and to have a corrected symbolic program as part of the output. Facilities for correcting programs in this way are inherent in the Utility System and such a reassembly process is planned for future use. However, at the outset such a program will be unavailable for use by programmers. All corrections will be made through the Correction Program.

2. Subroutine Library

A library of subroutines to compute mathematical functions and perform input-output and debugging tasks for the programmer will be part of the utility system. Most of these routines can be called for within the symbolic language program by use of macro instructions (see section B.2b(2)). The less commonly used subroutines will be called for by use of the pseudo instruction LIB and a specified calling sequence. The subroutines themselves are "debugged" routines which meet certain specifications and are present in the Utility System as machine language programs. The language translators will incorporate any of these subroutines into programs which "call" for them. (See the detailed description of the subroutine library in section D)

3. Auxiliary Routine Library

Another part of the Utility System is a group of programs to perform certain functions which are not integral parts of any program but which are often necessary while "running" or "debugging" programs. These programs will, for example, allow the programmer to print the contents of a magnetic tape on the fast printer, to print the contents of memory on the fast printer, etc.

4. System's Control Program

Access to any of the parts of the utility system is gained through the System's Control Program. By use of Instructions to the System's Control (see B.2b(4)), the programmer can specify to this overall interpreting program which language translator or auxiliary program he wants to use. Since the entire utility system is contained on a magnetic tape called the "System Tape", the control program will interpret the Instructions to the System's Control given to it on paper tape and take from the System Tape into memory the proper program. It then transfers control to this program. For example if the SCP instruction read by the System's Control Program is PT ASSEMBLE, the paper tape assembly program is brought into memory and it operates as specified in section C.1a.

Section D. Subroutine Library

The Utility System will have a subroutine library in two parts, as was introduced in section C.2. The first or main library will include commonly used subroutines such as elementary functions routines, print routines, etc. The second, or library 2, will be made up of specialized subroutines used less frequently. Both parts of the library can be enlarged or changed as need arises.

1. Structure of the Library

In order to aid the programmer in using library subroutines, several forms of information about them will be available: short descriptions of the subroutines from the card file of Subroutine Abstracts; a detailed discussion of any subroutine in the Catalog of Subroutine Write-ups. The subroutines will be kept in symbolic language and in machine language in a paper tape file. All subroutines in the main library will be available in relocatable machine language on magnetic tape. These will be available to the programmer by means of macro instructions. Most of the library 2 subroutines will also be available as a separate file on magnetic tape. The pseudo instruction LIB will make these available to the programmer.

2. Requirements

For each subroutine which is included in the library, certain information about it must be made available and certain rules of programming must be conformed to.

a. Programming Method.

(1) The subroutines should be written in symbolic language

(2) The calling sequences for the subroutines, while they can be of variable length, should have the following form:

Location	Instructions
α	TR (SUBR)
$\alpha+1$	SKIP n-1
$\alpha+2$	locations containing variable information required by SUBR and error return or returns if required
'	
'	
'	
$\alpha+n$	
$\alpha+n+1$	normal return

where α is the first location of the calling sequence and SUBR is the symbolic name of the subroutine.

(3) Erasable storage should be assigned locations immediately following the last instruction of the main body of the subroutines.

(4) Before a subroutine returns to the main program, it must restore the following registers if they have been destroyed during the operation of the subroutine: B registers, A register and Sense register.

(5) Subroutines may require that an argument be placed in the U or R registers before entrance to the subroutine. These registers may be destroyed on exit from the subroutines.

(6) Results may be stored in the U and R registers on exit from a routine.

(7) The T registers may be used during operation of subroutines and they need not be restored; however the subroutine write-ups must contain a list of T registers which are destroyed.

(8) Any programming techniques - use of indicators, etc. - should be internal to the subroutine. The main program should never be required to preset any part of a subroutine.

(9) There should be no programmed stops within subroutines.

(10) Manual operation of Merlin, that is, use of Breakpoint switches, etc., should not be required by subroutines.

(11) Subroutines should not use the Tag 2 bit.

b. Subroutine Abstract

The card file of Abstracts will be a quick reference source for all library subroutines. The information which must be available for the abstract file is:

(1) Name of Subroutine and statement of problem.

(2) Which part of the library contains this routine - main library or library 2.

(3) Short statement of method used, range of arguments and other pertinent information.

(4) Name of programmer and date checkout procedure was completed. (Names of programmers making corrections and date of each correction).

c. Subroutine Write-Up.

The Catalog of Write-Ups will be the source of detailed information about each subroutine. Each write-up must include the following information:

(1) Name of subroutine and statement of problem.

(2) An adequate description, in outline form, of the method used.

(3) A discussion of accuracy and/or any other limitations of the subroutine.

(4) The number of storage registers used by the subroutine (length of program), not including erasable storage registers.

(5) The number of erasable storage registers used.

(6) A list of disturbed and unrestored registers.

(7) The time of operation, if possible.

(8) A description of the calling sequence including: the form of arguments, the form of result, where the argument or arguments are to be placed on entry, where the result or results are placed on exit, a description of indicators and other information used in the calling sequence, a description of the error return or returns.

(9) A list of other library subroutines used by this subroutine.

(10) Names and references of any other routines which are included in this subroutine and which the programmer submitting the subroutine obtained from other sources.

(11) A printed listing of the subroutine as obtained from the assembler.

(12) The name of the programmer and date of final checkout (and names of programmers making corrections and dates of corrections).

d. Symbolic and machine language programs on paper tape.

The paper tape file of symbolic programs will contain the final versions of the symbolic programs; that is, the subroutine after it has been completely debugged. The machine language program in this file will be a direct copy of the subroutine as it appears on the magnetic tape library file.

e. Machine language program on magnetic tape.

The files of the main library and library 2 (in relocatable binary codes) will be available on magnetic tape. Subroutines which have been completely debugged and for which all the preceding information is available will be added to the Utility System as part of one of the above library files. Since any alteration of these library tapes affects the entire Utility System, any such additions must be made only by the group working on the system.

```
 0.00 c400 008a 0000   0000   BEG    TR (LST)
 1.00 bb00 0002 0000   0001          SKIP 2
 2.00 0400 0735 0000   0002          <1024<(HEAD)<0
 3.00 0000 0050 004b   0003          <<80<75       ——— NG
 4.00 c400 03a0 0000   0004          ~~TR (LPR)~~
 5.00 c400 04b0 0000   0005   START  TR (DIN)
 6.00 bb00 0001 0000   0006          SKIP 1
 7.00 0000 0045 0002   0007          <<(HVAL)<2
 8.00 e400 0045 0000   0008          (HVAL)→U
 9.00 a700 0049 0000   0009          U-(RADI)→U
10.00 be00 0047 0000   000a          U→(A)
11.00 e400 0045 0000   000b          (HVAL)→U
12.00 a500 0049 0000   000c          U+(RADI)→U
13.00 be00 0048 0000   000d          U→(B)
14.00 c400 06d8 0000   000e          TR (INT)
15.00 bb00 0002 0000   000f          SKIP 2
16.00 0000 0047 0048   0010          <<(A)<(B)
17.00 0000 0030 0014   0011          <<(F)<20
18.00 ad00 004a 0000   0012          U*(2OVA) → U,N
19.00 be00 004b 0000   0013          U→(DOVK)
20.00 ad00 004c 0000   0014          U*(KCON)→U,N
21.00 be00 004d 0000   0015          U→(DOSE)
22.00 e400 0045 0000   0016          (HVAL) → U
23.00 a904 0000 0000   0017          U*U → T4,N
24.00 ed00 0046 0046   0018          (KVAL)*(KVAL)+T4 → T4
25.00 e400 073e 0000   0019          (ONE) → U
26.00 d100 0004 0000   001a          U/T4 → U
27.00 be00 073f 0000   001b          U → (1OVP)
28.00 ad00 004b 0000   001c          U*(DOVK) → U,N
29.00 be00 0740 0000   001d          U → (RATO)
30.00 c400 008a 0000   001e          TR (LST)
31.00 bb00 0002 0000   001f          SKIP 2
32.00 0100 073f 0000   0020          <256<(1OVP)<0
33.00 0000 0005 0016   0021          <<5<22
34.00 c400 008a 0000   0022          TR (LST)
35.00 bb00 0002 0000   0023          SKIP 2
36.00 0100 0740 0000   0024          <256<(RATO)<0
37.00 0000 0005 002d   0025          <<5<45
38.00 c400 008a 0000   0026          TR (LST)
39.00 bb00 0002 0000   0027          SKIP 2
40.00 0100 004b 0000   0028          <256<(DOVK)<0
41.00 0000 0005 0041   0029          <<5<65
42.00 c400 008a 0000   002a          TR (LST)
43.00 bb00 0002 0000   002b          SKIP 2
44.00 0100 004d 0000   002c          <256<(DOSE)<0
45.00 0000 0005 0055   002d          <<5<85
46.00 c400 03a0 0000   002e          TR (LPR)
47.00 c400 0005 0000   002f          TR (START)
48.00 dc00 0044 0000   0030   F      PF→(EXIT)
49.00 e001 0000 0000   0031          U→T1
50.00 a700 0045 0000   0032          U-(HVAL)→U
51.00 a902 0000 0000   0033          U*U→T2,N
52.00 e400 0049 0000   0034          (RADI)→U
53.00 a900 0000 0000   0035          U*U→U,N
54.00 a300 0002 0000   0036          U-T2→U
55.00 c000 0000 0000   0037          SQ
56.00 e002 0000 0000   0038          U→T2
57.00 e400 0046 0000   0039          (KVAL)→U
58.00 a903 0000 0000   003a          U*U→T3,N
59.00 e000 0001 0000   003b          T1→U
```

```
60.00 a900 0000 0000   003c          U*U→U,N
61.00 a103 0003 0000   003d          U+T3→T3
62.00 c000 0000 0000   003e          SQ
63.00 e003 0000 0000   003f          U→T3
64.00 e000 0002 0000   0040          T2→U
65.00 d100 0003 0000   0041          U/T3→U
66.00 c400 004e 0000   0042          TR (ART)
67.00 d100 0003 0000   0043          U/T3→U
68.00 c400 0000 0000   0044   EXIT   TR (0)
69.00 0000 0000 0000   0045   HVAL   <
70.00 0000 0000 0000   0046   KVAL   <
71.00 0000 0000 0000   0047   A      <
72.00 0000 0000 0000   0048   B      <
73.00 0079 eb85 1eb8   0049   RADI   DEC 0.47625
74.00 1002 ce90 ff97   004a   2OVA   DEC 2.8069
75.00 0000 0000 0000   004b   DOVK   <
76.00 2001 91ee 147b   004c   KCON   DEC 401.93
77.00 0000 0000 0000   004d   DOSE   <
78.00                  004e   ART    LIB
79.00                  008a   LST    LIB
80.00                  04b0   DIN    LIB
81.00                  06d8   INT    LIB
82.00                                STA (BEG)
83.00 6186 014a 3a32   0735   HEAD   PCC    1/DIST, SQ,                  RATIO
      cd36 32c1 3618   0736
      6186 1861 8618   0737
      6186 1861 8c60   0738
      ce8b 9861 8618   0739
84.00 6186 1861 8618   073a          PCC                 INTEGRAL          DOSE
      6186 28b7 3926   073b
      c60a d861 8618   073c
      6186 23bb 2918   073d
85.00 1001 0000 0000   073e   ONE    DEC 1.0
86.00 0000 0000 0000   073f   1OVP   <
87.00 0000 0000 0000   0740   RATO   <
END   0200 0000 0000
```