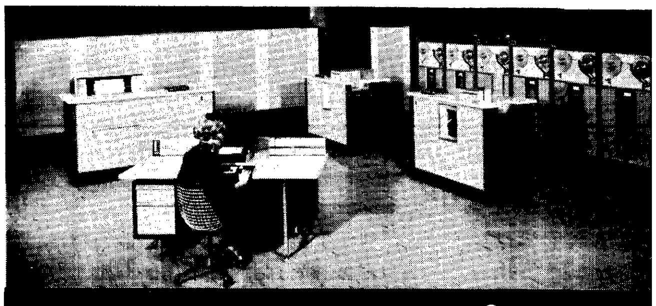


UNIVAC® III

TECHNICAL BULLETIN

UTMOST

Programmer's Guide

August 9,1962

The mnemonics for machine operation codes as printed in this manual are considered acceptable but non-standard by UNIVAC. This means that:

1. The UTMOST processor will accept programs written using either this set of mnemonics or the standard UNIVAC III set.

or,

2. A program will be furnished which will convert source programs with the non-standard mnemonics to source programs containing standard mnemonics.

The following table gives the relationship between standard and non-standard mnemonics:

SALT (standard)	UTMOST (non-standard)
L	LA
LCS	LAN
EXT	LF
ST	SA
STCS	SAN
---	SZ
A	DA
S	DS
AH	DAH
SH	DSH
M	DM
D	DD
BA	BA
BAH	BAH
BS	BS
BSH	BSH
SR	DSR
SL	DSL
SAR	ASR
SAL	ASL
SBC	BRR
CA	CM
C	C
CONE	CPA
CZRO	CPZ
TEQ	JE
THI	JG
TLO	JL
TPOS	JP
TUN	J
TR	SLJ
SSI	SS
RSI	RS
TSI	JS

SALT (standard)	UTMOST (non-standard)
ATD	LAD
DTA	SAA
ZUP	LAE
SUP	OR
ERS	AND
LX	LX
STX	SX
IX	IX
ICX	IXC
TCI	TC
RCI	RC
TPE	TPE
RPE	RPE
TIO	TIO
TIO	TW
TIO	TR
TCI	TOV
TCI	TOP
RIO	RIO
RIO	RW
RIO	RR
AIO	AI
PIO	PI
TIOP	JIP
IOF	LC
IOF	LWC
IOF	LRC
NOP	NOP
STMC	SC
TR*	SCJ
STMC	SL
STMC	SWC
STMC	SRC
STCR	ST
STCR	SRT
STCR	SWT
WAIT	HJ
LT	RCK
DIS	WD
RT	RT
WT	WT
ACT	AT

An assembler directive will be supplied for specifying the desired set of mnemonics. The method of accomplishing this will be specified later.

Manual Number:

Nº 01047

UNIVAC III UTMOST

MANUAL REGISTRATION SHEET

CHARGE TO:	NAME______________________________
	BRANCH____________________________
	ADDRESS___________________________
	__________________________________
	DATE______________________________

This registration entitles the holder of this manual to receive all updating materials.

Remove this sheet and supply the above information. Immediately mail to:

MANAGER, PROGRAM LIBRARY SERVICES
UNIVAC
315 PARK AVENUE SOUTH
NEW YORK 10, NEW YORK

UNIVAC III UTMOST

No 01047

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

TABLE OF CONTENTS (Cont'd)

I. INTRODUCTION

UTMOST (UNIVAC THREE MACHINE ORIENTED SYMBOLIC TRANSLATOR) is an easy to learn and easy to use assembly language designed to permit rapid efficient coding for UNIVAC III. UTMOST is a two-pass assembly system providing rapid translation from symbolic to object coding.

The UTMOST system contains a wide and sophisticated variety of operators which provide the ability to fabricate fields during assembly without restrictions on the programmer. The mnemonic operation codes describe machine functions and prevent the programmer from having to learn a wide variety of octal machine codes. The system has a series of ten assembly directing instructions which aid greatly in promoting easy communication with input-output and executive systems. In addition, the assembly directives provide the programmer with the ability to write short routines which are variable at assembly time. These routines and standard routines are easy to incorporate in the program, thereby reducing the effort of the programmer and increasing programming production.

UTMOST produces relocatable binary output in a card form suitable for processing by a binary card loader. It also supplies a listing of the original symbolic coding together with an octal representation of the word generated. Certain error flags are also supplied in the listing.

The UTMOST manual is in several sections. Section II is designed to aid the programmer unfamiliar with this type of system. Section III is designed to act as a brief programmers' reference guide to the UTMOST system.

II. A BASIC INTRODUCTION TO THE UTMOST ASSEMBLER LANGUAGE

A. GENERAL

1. Computers and Languages

In order to solve a problem, a computer must be given a series of instructions which determine how the computer is to operate. In addition, the computer must be given one or more sets of data upon which to operate. This combination of instructions and data is called a program. A program must define in complete detail exactly what the computer is to do, under every conceivable combination of circumstances, with the data which is read into or processed by the computer. The number of instructions required for the complete solution of a problem may be a few hundred or many thousands, depending upon the problem. The computer may refer to these instructions one after another. It can also be instructed to repeat, modify, or skip over certain instructions, depending upon intermediate results or circumstances. The ability to repeat operations, usually called looping, combined with other facilities of modifying and skipping over instructions, permits a significant reduction in the number of instructions required to perform a given job. For example, two sets of numbers exist and it is desired to add the corresponding numbers of each set together. Instructions may be written to add the first number of the first set to the first number of the second set, then to repeat this operation with the second, third, fourth, etc., numbers of each set. In this way, a few instructions may cause thousands of additions.

Since the computer does not respond to the English language, the program must be encoded in a form known as machine language. Considerable time and effort have been expended in developing programming systems that allow the programmer to write in a symbolic language more easily comprehensible to him than machine language. Associated with a programming system is a machine language program called a processor. The processor accepts a program written in the symbolic language (source program) and converts it into a machine language program (object program). The symbolic language utilized to program for UNIVAC III is known as UTMOST (Univac Three Machine Oriented Symbolic Translator).

2. The UTMOST Assembler

The UTMOST assembly program was designed to provide a programmer with an easy to learn and easy to use assembly system. UTMOST is a straightforward data processing program, accepting input data (symbolic coding) and processing it and producing as system output, object coding usable by UNIVAC III directly.

As the symbolic coding is processed, the UTMOST assembler tallies the number of lines produced in a location counter. The location counter can be referenced by the programmer in his symbolic coding and may be utilized throughout his program. UTMOST also provides the programmer with a series of 'operators' permitting him to fabricate any object code values which he may need. A small number of extremely powerful assembly directives are also made available which allow the programmer to direct the assembly in an extremely positive manner during the actual assembly. In addition, the programmer may use mnemonic operation codes which explain machine functions by their very nature rather than having to learn the machine code bit configurations.

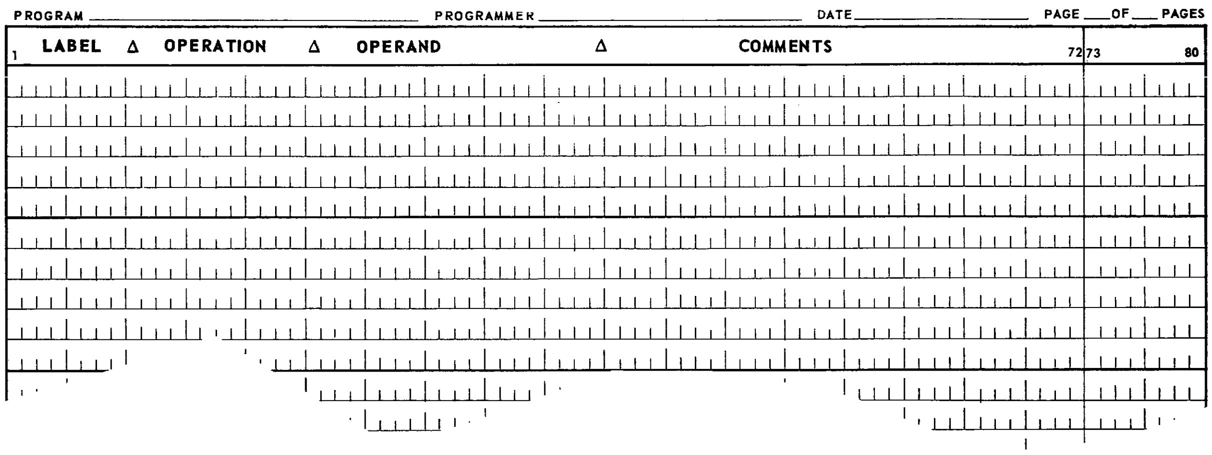
The UTMOST assembler provides output in the form of a loadable object program plus a listing of the symbolic program and the object program. The listing also provides the programmer with error flags at whatever points the assembly system detected the errors.

In the section following, each feature of the UTMOST assembly system is examined in detail with examples of each operation, as well as an illustrative problem demonstrating a legitimate approach to the solution of a simple data processing problem for UNIVAC III utilizing the UTMOST language.

3. Symbolic Coding Format

In writing a program in UTMOST symbolic language, the programmer is primarily concerned with three fields, a label field, operation field and operand field. In addition, it is possible to annotate the symbolic language at the time it is written through the use of comments which will provide clarity for the programmer and relate the coding to its associated flowchart.

In writing in UTMOST language, the programmer is not bound by a fixed length field concept as is the case with older assembly languages. All of the fields in UTMOST are in free form, and are designed to provide the greatest convenience possible for the programmer.


PROGRAM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___ PAGES

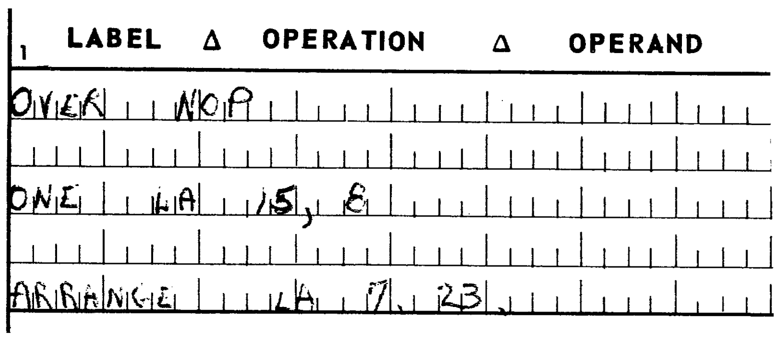
1 LABEL Δ OPERATION Δ OPERAND Δ COMMENTS 72 73 80

a. Label Field

A label is a method of identifying either a symbolic line of coding, or a word of data. In writing a label in UTMOST, the programmer may use any meaningful combination of one to eight characters. Of these eight characters, the first must be an alphabetic (A... Z), and the others, if present, may be either alphabetics or numerics (0-9). Sample labels are listed below:

PRNT	ARRANGE
ONE	ADOL
A	OVER2

In writing a label in the label field of a symbolic line, the first character of the label must be left justified within the line and the field terminated by a blank. There must be no blanks within the label field itself. When the label is analyzed by the UTMOST assembly program, it is equated to the current value of the location counter except in the cases of a label associated with the EQU, FORM, DO, FLD, PROC and NAME assembly directives. Each of these special cases is discussed separately in the portions of the manual dealing with the specific directive.



```
1 LABEL   Δ   OPERATION   Δ   OPERAND
OVER      NOP

ONE       LA      15, 8

ARRANGE       LA   7, 23,
```

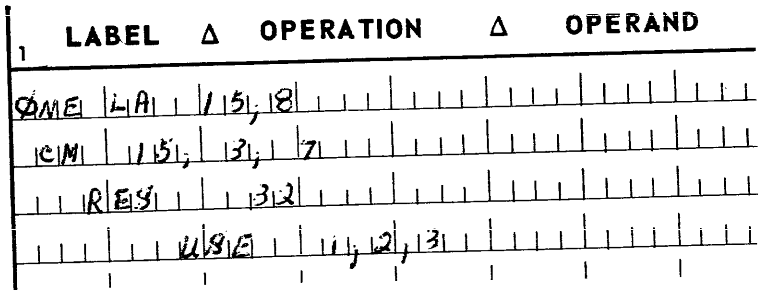
In the symbolic lines illustrated above, each of the labels in the label field, OVER, ONE and ARRANGE follow the requirements of the label field. Each starts with an alphabetic in column 1, is from one to eight characters in length, and is terminated by a space.

b. Operation Field

The operation field of a symbolic line informs the assembler of the purpose of the line. An operation field may be up to eight characters in length, and may contain a mnemonic machine operation code, an assembler directive, a label associated with a FORM NAME or PROC directive or a data generating code. Each of the above categories will be discussed in detail in its appropriate section.

An entry in the operation field is terminated by a blank unless it is a plus or minus sign, in which case the operand field may begin in the succeeding column. If the line does not have a label, the operation field may begin in the second column of the coding form.

If an operation field contains an assembler directive other than RES (which increments the location counter), the location counter will not be affected. In all other cases, the location counter will be incremented by one after the line has been generated.



```
1 LABEL  Δ  OPERATION  Δ  OPERAND
ØNE LA  15,8
 CM  15,3,7
   RES  32
      USE  1,2,3
```

In the illustration of operation fields above, Line 1 contains an operation field LA following the label ONE.

Line 2 contains an operation field, CM, starting in column 2, showing that no label is present.

Line 3 contains an assembler directive as an operation field, RES.

Line 4 also contains an assembler directive in the operation field, USE.

Note that each operation field follows the rules stated above.

c. Operand Field

The operand field of a symbolic line follows the label and operation fields. It consists of one or more expressions defining the information required by the operation field of the line.

Expressions within the operand field are separated by commas, and the comma indicates that another expression follows. Termination procedures are discussed under Line Control, below. The maximum number of expressions on a line is determined by the content of the operation field of the line. However, any line may contain less than the maximum number of expressions indicated by the operation field; so long as it has at least one. The unwritten expressions will be assumed by the assembler to be zero.

```
1  LABEL  Δ  OPERATION  Δ  OPERAND
 LA 0
 LX    3,   71,  2
 LA
```

In the examples, the 0 following LA represents a single expression in the operand field. The second line of symbolic coding represents a three expression operand field, each expression separated from the previous one by a comma.

d. Line Control

The information content of a line to the assembler consists of a label, operation, and operand fields. The information content is normally terminated when the maximum number of expressions required by the operation have been encountered (or maximum number of lists in the case of a procedure reference, or by column 72, whichever occurs first. There are two special marks which override the normal rule:

1) Continuation: If a ";" is encountered outside of an alphabetic item, the current line is continued with the first non-blank on the following line and there is no more information to the assembler on the line in which the ";" occurred.

2) Termination: If a "." followed by a blank is encountered outside of an alphabetic item, the line is terminated at this point. If additional expressions are required by the operation field, they are assumed by the assembler to be zero.

A continuation or termination mark may occur anywhere on a line. Following the information control of a line, any characters may be entered.

```
1 LABEL  Δ  OPERATION  Δ  OPERAND
LABEL FOR;
M ;
1,24
```

The semicolons indicate that the line is continued on the next line. The assembler would treat the three lines as though they were the following line.

```
LABEL FORM 1,24

LABEL FORM 1,24. THIS LINE IS TERMINATED BY THE PERIOD SPACE
. THIS LINE IS ALSO TERMINATED BY THE PERIOD SPACE
. AND WILL BE PRINTED ON THE SYMBOLIC LISTING ONLY
```

The three lines above use a period followed by a space to terminate the lines. Any information following the period space is considered to be a comment and will be printed on the symbolic output listing. The assembler will take no action on the information following the period.

4. Expressions

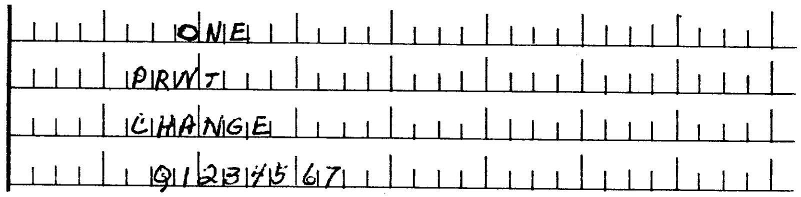
An expression is an elementary item or a series of elementary items connected by operators. It normally appears in the operand field of a symbolic line.

a. Elementary Items

UTMOST permits the utilization of a series of elementary items which may be used in expressions.

1) Label: Any label may be used as an elementary item. The structure of a label corresponds to the description of the label field discussed earlier. A label may be from one to eight characters in length, the first of which must be an alphabetic. When a label has been encountered in the label field of a symbolic line (with exceptions

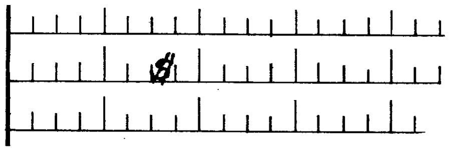
noted under Label Field), it is assigned the current value of the location counter. Thereafter, when it is encountered within an expression, the integer value initially assigned to it will be substituted for the label within the expression.



Various sample labels are illustrated above as they would appear as elementary items in the operand field.

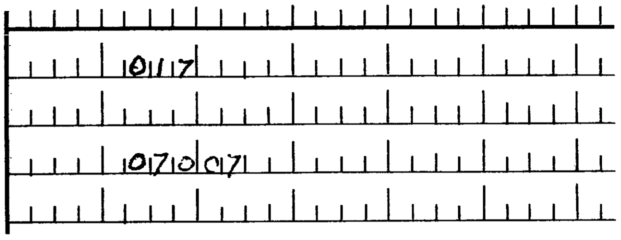
2) Location:

The current value of the location counter may be used as an elementary item within the operand field of a symbolic line. The format of a reference to the location counter is the dollar sign ($). When this sign appears in an expression, the value of the location counter is substituted for it. It is useful in reflexive addressing.



In the example above, if the current value of the location counter was 5280, the integer value 5280 would be substituted for the dollar sign ($) in its expression, and right justified within the object field.

3) Octal: Octal values (base eight) may be represented in expressions as elementary items by preceding the desired value with a zero. The assembler will convert these values to their corresponding binary (base two) equivalents. The converted binary integer will be right justified in its object coded field.

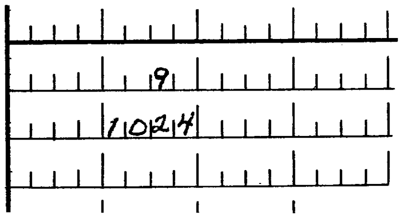


In the examples above:

017 is equivalent to	000 000 000 000 000 000 001 111
07007 is equivalent to	000 000 000 000 111 000 000 111

in their converted object code.

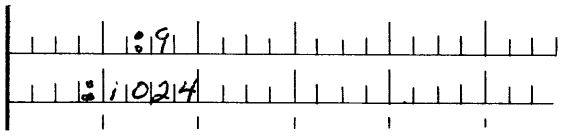
4) Decimal: Decimal values may be used as elementary items within an expression. Where they appear, decimal values (base 10) will be converted into their binary equivalents and right justified within their object fields. A decimal item is represented as a non-zero digit followed by decimal (0-9) digits.



In the examples above:

9 is equivalent to	000000000000000000001001
1024 is equivalent to	000000000000010000000000

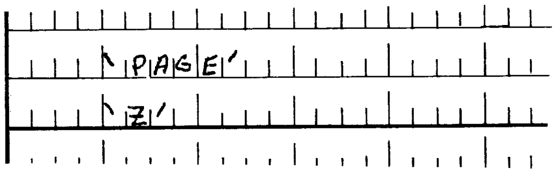
5) BCD: UNIVAC III binary coded decimal excess three values in four bit notation may be utilized in elementary items by preceding the value with a colon (:). When a decimal value appears in this format, it will be translated by the assembler into its corresponding 4 bit base 16 value and right justified within its field.



In the examples above:

:9 is equivalent to	0000 0000 0000 0000 0000 1100
:1024 is equivalent to	0000 0000 0100 0011 0101 0111

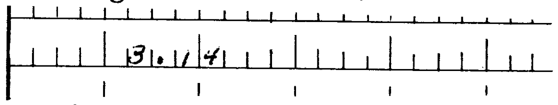
6) Alphabetics: Excess three six bit alphabetic characters may be represented in an elementary item by enclosing the desired characters within apostrophes ('). Since the assembler recognizes an apostrophe as the end of the alphabetic value, it is not permitted to use an apostrophe within the alphabetic grouping. The six bit object code resulting from an alphabetic item will be right justified within its field and preceded by binary zeros (space codes).



In the example above:

'PAGE' is equivalent to	101010 010100 011010 011000
'Z' is equivalent to	000000 000000 000000 111100

7) Floating Point Numbers: Floating point numbers may be represented within an elementary expression by including a decimal point (period) within the desired decimal value. the converted value will be in standard UNIVAC excess 50 floating point format with a ten digit mantissa and a two digit characteristic.



In the example above:

3.14 is equivalent to 513140000000 in 4 bit BDC digits.

8) Field: A field may be referenced as an elementary expression by writing a field label followed by an expression enclosed in parentheses representing the address of the partial word. The field item is discussed in greater detail in the section on Assembler Directives, FLD directive.

```
   EXT(VALUE)
```

In the example above:

EXT represents the bit control pattern for field selection, (VALUE) represents the address from which the field will be selected.

9) Parameter: A parameter may exist as an elementary item by following the procedure label with one or two expressions enclosed in parentheses. The parameter item is discussed in detail under Assembly Directives, PROC directive.

10) Line: An entire line may exist as an elementary item by enclosing the line within parentheses. The assembler will generate the value of the word that the line would generate if it existed as a separately coded line.

```
    ('DON')
```

In the above example:

('DON') would generate the constant DON in six bit excess three alphabetics preceded by binary zeros in the same manner that 'DON' would on a symbolic line by itself.

b. Operators

An expression may consist either of an elementary item, or a series of elementary items connected by operators as shown in the table below:

+	Arithmetic Sum
-	Arithmetic Difference
*	Arithmetic Product
/	Arithmetic Quotient
++	Logical Sum (OR)
--	Logical Difference (EXCLUSIVE OR)
**	Logical Product (AND)
//	Covered Quotient $(a//b = \frac{a+b-1}{b})$
=	Equals
>	Greater Than
<	Less Than
*+	$a *{+}b = a*10^{b}$
*-	$a *{-}b = a*10^{-b}$

1) + Arithmetic Sum : The arithmetic sum operator may be used to combine two or more items. The assembler will sum the integer values of the items and the resultant integer value will be utilized in the resulting expression.

```
    +   7+3
    +$+15
```

In the above examples:

7 + 3 would produce the integer 10 in binary.
$ + 15 would produce the current value of the location counter incremented by 15 in binary.

2) - Arithmetic Difference : The arithmetic difference operator may be used to subtract one item from another. The assembler will subtract the integer value of the second item from that of the first, and the resultant integer difference will be substituted in the expression.

```
   +  $-3
   +VALUE-10
+     7-4
```

In the above examples:

$ - 3 will produce the current contents of the location counter less three.
VALUE - 10 will produce the integer equivalent of the label "VALUE" minus ten.
7 - 4 will produce the integer three.

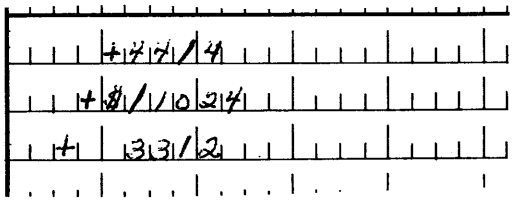
3) <u>* Arithmetic Product</u>: The arithmetic product operator may be used to multiply one item by another producing the arithmetic product. The assembler will multiply the integer value of the first item by the integer value of the second item and the resultant integer value will be substituted in the expression.

```
   +7*3
+   $*2
```

In the above examples:

7*3 will produce the integer value 21.
$*2 will produce an integer value equivalent to the current contents of the location counter times 2.

4) <u>/ Arithmetic Quotient</u>: The arithmetic quotient operator may be used to divide one item by another producing the arithmetic quotient. The assembler will divide the integer value of the first item by the integer value of the second item, and the resultant quotient will be utilized in the expression. The remainder is discarded by the assembler.

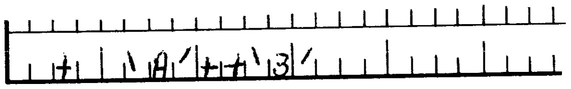


```
+44/4
+$/1024
+ 33/2
```

In the above examples:

44/4 will produce the integer value 11.
$/1024 will produce an integer value equivalent to the number of possible index registers required for area addressing in the program up to this point in the program.
33/2 will produce an integer value of 16 (remainder has been discarded).

5) ++ Logical Sum (OR): The logical sum operator (OR) may be used to logically sum the binary equivalents of two items. The assembler will logically add the two values and the resulting logical sum will be utilized in the expression.



```
+ 'A'++'3'
```

In the above example:

'A' in six bit code is	010100
'3' in six bit code is	000110
Logical sum generated	010110

6) -- Logical Difference (EXCLUSIVE OR): The logical difference operator may be used to obtain the logical difference between the integer values of two items. The assembler will perform an EXCLUSIVE OR on the two items (where a bit is present in corresponding position in both items, the result is binary 0, where no bit is present in corresponding positions, the result is binary 0, where a bit is present in either one of corresponding positions, the result is 1). The resultant integer is then utilized as the value of the expression.

```
   +   'V'-'T'
```

In the above example:

'V' in six bit code is	111000
'T' in six bit code is	110110
Logical difference is	001110

7) <u>** Logical Product (AND)</u>: The logical product operator may be used to AND (Logically multiply) the integer value of one item by another. The assembler will logically multiply the two values and the resulting logical product will be utilized in the expression.

```
    +   'V'**'T'
```

In the above example:

'V' in six bit code is	111000
'T' in six bit code is	110110
Logical product is	110000

8) <u>// Covered Quotient $(a//b = \frac{a+b-1}{b})$</u>: The covered quotient operator may be used to divide the integer value of an item by the integer value of a second item or expression. The effect is the same as adding one to the integer value of the quotient in straight division (A/b) if there were a remainder. The resultant integer will be utilized in the expression.

```
   ($-START)//1024
```

In the above example:

($-START)//1024 (where START is the first location required by the program and greater than 1024) will produce a covered quotient equivalent to the number of index registers required for area addressing up to the point where the expression appeared.

9) = Equal: The equals operator may be used to compare the integer values of two items or expressions. If the two integer values are equal, the assembler will generate a binary 1 as the resultant field. If the two integer values are not equal, the assembler will generate a binary 0 as the resultant field.

```
$=7083
```

In the above example:

If $ = 7083, a value of binary 1 will be generated.
If $ ≠ 7083, a value of binary 0 will be generated.

10) > Greater Than: The greater than operator may be used to compare the integer values of two items or expressions. If the integer value of the first item or expression is greater than the integer value of the second, the assembler will generate a binary 1 as the resultant field. If the first value is less than or equal to the second, the assembler will generate a binary 0 as the resultant field.

```
AMOUNT>2
```

In the above example:

If the value of AMOUNT is greater than 2, a binary 1 will be generated, otherwise a binary 0 will be generated.

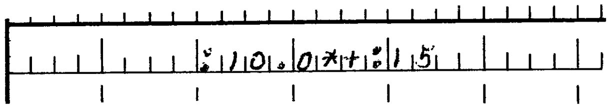
11) < Less Than: The less than operator may be used to compare the integer values of two items or expressions. If the integer value of the first item or expression is less than the integer value of the second, the assembler will generate a binary 1 as the resultant field. If the first value is greater than or equal to the integer value of the second, a binary 0 will be generated.

```
COUNT<5
```

In the above example:

If the value of COUNT is less than 5, a binary 1 will be generated, otherwise a binary 0 will be generated.

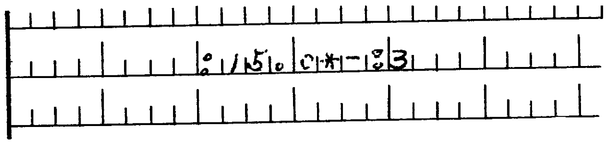
12) <u>*+ Positive Exponent</u>: The positive exponent operator may be used to create a two word floating point constant in excess 50 notation where a * + b is equivalent to a $*10^b$. Both words must be excess three binary coded decimal numerics.


:10.0*+:15

In the above example:

:10.0*+:15 will produce 671000000000

13) <u>*- Negative Exponent</u>: The negative exponent operator is similar to the positive exponent operator except that it will produce a floating point word in excess 50 notation with a characteristic from 0 to 50.


:15.0*-:3

In the above example:

:15.0*-:3 will produce 491500000000 as the integer equivalent in standard UNIVAC excess 50 floating point format.

In all of the foregoing cases where items are connected by operators, if the value produced by an expression is a negative integer, it will be represented by a 2's complement unless the operation field of the line contains an EQU directive or, in some cases, the operation field is + or -.

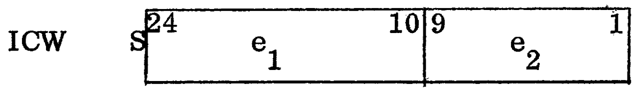
5. Data Word Generation

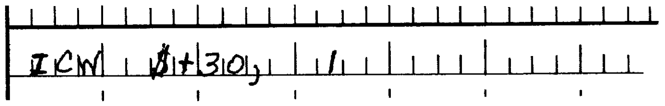
The UTMOST assembly system provides three means of generating data words other than expressions. These data words consist of Increment and Compare Words, two word constants, and words with a plus (+) or minus (-) operation field. The last category provides the ability to generate one word constants, indirect address words and field select words with or without index register indices.

a. Increment and Compare WORD, ICW

The increment and compare word is used to prepare a word suitable for incrementing and comparing an index register (with the IX and IXC instructions).

The Increment and Compare word is written with ICW in the operation field of the line, followed in the operand field by two expressions, e_1 and e_2. The first expression, e_1, represents the comparison amount and the second expression, e_2, represents the increment. The format of the generated word is illustrated below:




ICW $+30,1

In the above example:

ICW informs the assembler that this is an increment and compare word. $ + 30, the first expression,represents the comparison amount;1, the second expression,represents the increment.

b. Two Word Constant Generation, TWC

A two word constant may be generated by placing TWC in the operation field of a line, and the constant in the operand field. This symbolic line must have a label. The assembler will generate the value of the expression in the operand field, right justify filling with binary zeros the resultant value in the two word field, and assign an address to the label. The left half of the two word constant may be addressed by using the label, the right half by using the label plus one.

```
ZERO    TWC    (0)
HDR     TWC    ('PAGE NO.')
```

In the above examples:

ZERO TWC (0) will produce a two word constant of binary zeros.

HDR TWC ('PAGE NO.') will generate a header line for editing purposes.

The first example may be referenced by ZERO+1 and a two register indicator in the "a" field of an instruction, the second by HDR+1, and a two register indicator in the "a" field.

c. <u>+ or - Operation Field</u>: A + or - operation field plus from one to four expressions in the operand field may be used to generate specific constants consisting of a one word constant of datum, an indirect address word, a field select word without index register notation (or implied index notation), and a field select word with specific index register notation.

1) <u>One word data constants:</u> One word constants may be generated by placing a + or a - in the operation field followed by one expression in the operand field. It is not necessary to leave a blank between the + or - sign in the operation field and the operand field.

```
A  +'DATA'
B  -$
C    +VALUE+10
D  -=5280
```

In the above examples:

A will produce a one word alphabetic constant in six bit code containing the word "DATA".

B will produce a one word constant containing the current value of the location counter in binary, right justified with preceding binary 0s and a negative sign.

C will produce a positive binary constant containing the address plus ten of label "VALUE".

D will contain a negative constant in excess three binary coded decimal notation preceded by binary zeros of the value "5280".

2) Indirect Address Words: Indirect address words may be generated through the use of a + or - operation field plus two expressions in the operand field. The first expression will be generated as a fifteen bit UNIVAC III address, and the second expression will be generated as a four bit index register code. The sign of the word will be the sign in the operation field.

```

  +DATA+10   9
```

In the above example:

An indirect address word will be generated containing the fifteen bit address of the expression 'DATA+10' in the least significant fifteen bits of the word, Index Register #9 in the four most significant bits of the word, and the sign of the word will be positive, indicating that no chaining of indirect addresses is desired.

3) Field Select Words: Field select words may be generated through the use of a + or - operation field plus three expressions in the operand field. The first expression will be generated into a five bit left bit control (plus binary three) integer indicating the left boundary of the field to be selected. The second expression will generate the right boundary of the field, also as a five bit binary integer plus binary three.

The third expression will generate a ten bit binary address for the word(s) from which the field is to be selected. The sign of the generated word must be positive.

```
   +  12, 5, VALUE
```

In the above example:

The first expression will generate 01111 (binary 15) as the left bit control, the second will generate 01000 (binary 8) as the right bit control, and the ten bit address equivalent to 'VALUE' from the third expression.

4) Field Select Words: As in 3, above, a field select word may be generated using four expressions in the operand field following a + or - operation field. The first expression will generate the left bit parameter, the second expression the right bit parameter, the third expression the ten bit 'm' address, and the fourth will be used to generate the index register designator.

```
   +12, 5, VALUE, 8
```

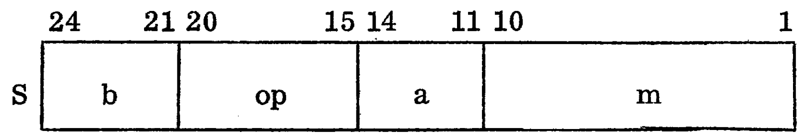
In the above example:

The first expression will generate binary 15 as the left bit control, the second will generate binary 8 as the right bit control, the third will generate a ten bit address equivalent to 'VALUE', as modified by the index register, 8, specified in the fourth expression.

6. <u>Mnemonic Instructions</u>

The UTMOST assembly system utilizes a series of mnemonic instructions corresponding to the octal machine code instructions in object coding which are recognizable by the computer. The mnemonic operation codes describe the function of the instructions, thereby removing the problem of learning the octal operation codes, or their binary equivalents. In some cases, a combination of octal operation code and bits in the AR portion form instructions. Mnemonics have been created to save a programmer from writing or knowing the parameter AR bit configuration for most of these.

UNIVAC III's instruction word consists of a 24 bit word with the sign in bit 25 used to indicate either indirect addressing or field selection. The format of the word on a bit basis is illustrated below:



where "b" indicates the index register designator,

"op" the operation code,

"a" the arithmetic register(s) designator, and

"m" the ten bit area address of the operand.

Since UTMOST provides semi-automatic insertion of area index register assignments, it is unnecessary to write a "b" designator in many cases.

(USE Directive) The order of writing a symbolic instruction line has been altered from the hardware format to provide greater convenience in programming. The format is:

```
op    a,m,b
```

Type 0 Instructions: Type 0 instructions have three fields representing the "a", "m", and "b" fields of the instruction word, respectively. The sign of the instruction will be + unless the "m" portion of the instruction is preceded by an asterisk indicating indirect addressing or field selection.

```
A   LA      1,  ('.0    '), 1
    OR      1,  TEMP+6,  1
       SA     1,  TEMP+6,  1
```

In the above illustration:

LA, OR, and SA are mnemonic instruction codes of type 0 category, requiring in each case the "a", "m", and "b" fields. (The "b" field may be omitted, if the USE assembler directive has been inserted in the program prior to the assembly encountering these instructions.

Type 1 Instructions: Type 1 instructions have two fields representing the "m" and "b" portions of the instruction word, respectively. The sign of the instruction word will be + unless the "m" portion of the instruction is preceded by an asterisk indicating indirect addressing or field selection.

```
 J   END,  1

 J   *ARRANGE-1,  1
```

In the above illustration:

J is the mnemonic code for the Jump instructions, the first instruction utilizing direct addressing, the second indirect addressing.

7. Line Item

A line item is an instruction line, form reference line, or data word line without label field and without leading or trailing blanks, enclosed in parentheses.

```
 LA  1,   (J   $+5)
 LA  2,   (MASK 70707070)
 LA  4,   ('DON')
```

In the above examples:

LA 1, (J $+5) The last expression is an instruction line written as line item.

LA 2, (MASK 70707070) The parenthetical expression (MASK 70707070) is a form reference line written as a line item.

LA 4 ('DON') The parenthetical expression ('DON') is a data word line written as a line item.

In each case, the assembler will generate an address which will be the address of the translated parenthetical expression. The translated parenthetical expression is called a literal. If the literal is identical to any other literal, the location assigned is the location of the previous literal, thus eliminating duplication.

When a line item appears in the address field of an IX or IXC instruction and has two expressions, it is evaluated as a data word with ICW in the operation field.

```
IXC  8,  (LIMIT, 10)
```

In the above example:

The assembler will generate an index register increment and compare word equivalent to the same expressions in an ICW line.

A literal will be double precision if the line was a TWC line or if it was a data line with one expression and the mode of the expression was floating.

```
LA   3,  (TWC 'PAGE NO.')

LA   12, (3.14)
```

In the above examples:

The first example will generate a two word constant (double precision) of the alphabetic constant "PAGE NO."

The second example will generate a two word excess 50 floating point constant where 3.14 is equivalent to 513140000000.

8. Assembler Directives

The UTMOST assembler provides the programmer with a series of powerful operation codes in the form of Assembler Directives. These assembler directives do not produce coding in and of themselves, but effectively provide a programmed means of controlling the process of assembly.

There are ten assembler directives as shown in the table below:

	Directive	Purpose
1.	EQU	Equate operand value to label field.
2.	RES	Reserve memory locations.
3.	USE	Assign index registers for area addressing.
4.	FORM	Designate arbitrary word format.
5.	FLD	Specify Field Selection pattern.
6.	END	Designate end of program or procedure.
7.	DO	Generate designated line(s) of coding.
8.	PROC	Generate associated coding if referenced.
9.	NAME	Qualify procedural coding.
10.	SET	Set index register to assumed value.

None of the assembler directives except RES will cause the location counter to be incremented. However, if coding is generated as a result of an assembler directive, the location counter will be incremented in the usual manner. A detailed discussion of each directive follows in this section.

a. EQU

The EQU assembler directive causes the label in the label field of its line to be equated in all succeeding references in the coding to the value of the expression in the operand field of the symbolic line. Thereafter, the label may be used in an expression, and the assembler will substitute for the label the integer value of the original expression in the operand field of the EQU line.

```
AR1  EQU  8
AR2  EQU  4
AR3  EQU  2
AR4  EQU  1
```

In the above example:

The four arithmetic register names have been equated to the binary values utilized in object code to address the respective registers. After these four EQU directives have been encountered by the assembler, the AR portion of an instruction may contain the label names of the registers, and the assembler will recognize them as the associated binary values. Accordingly, coding referencing these registers could read as follows:

```
   LA  AR1+AR3
   SA  AR2+AR4
   DA  AR1
```

b. RES

The RES assembler directive causes the value of the expression in the operand field to be added to the location counter. It may be used to reserve a specific or variable number of locations for input/output storage, or any other programmable purpose. (If the expression in the operand field is negative, the value of the expression will effectively be deducted from the location counter.) If it is desired to address any location within a reserved area, the label associated with the reserve directive may be used.

```
PRINTAREA   RES  32
```

In the above example:

The RES directive will cause 32 words of storage to be set aside (32 will be added to the location counter). These 32 words are equivalent to the 32 words or 128 characters required for one line on the High Speed Printer.

```
 LA  AR1+AR2,PRINTAREA+15
 SA  AR1+AR2, PRINTAREA+31
```

The two symbolic lines reference words 15 and 16, and 31 and 32 in the reserved area respectively.

c. <u>USE</u>

The USE assembler directive is utilized to load index registers with base values relative to the value contained in the location counter at the time the USE directive is encountered by the assembler. After a USE directive is encountered, it is not necessary to indicate index register designators in the operand field of a symbolic instruction line, since the assembler will insert the values automatically, unless a specific index register is desired by the programmer.

The USE directive, when encountered by the assembler will assign the current value of the location counter to the first index register specified in the operand field of the USE line, the current value plus 1024 to the second, and so on through the number of index registers specified in the operand field of the line.

It is possible to use more than one USE directive in a program, however, the value assigned an index register by a USE directive is loaded into that register at object time. Therefore, any particular index register may not be referred to more than one in a USE directive, or series of USE directives.

```
    USE 5, 6, 7
```

In the above example:

Assuming that the location counter reads 4000 at the time the directive is encountered, IR 5 will contain the value 4000, IR 6 will contain 5024, and IR 7 will contain 6048. IR's 5, 6, and 7 will automatically be inserted into object code where required by the program, and no indexing has been specified by the symbolic coding.

d. <u>FORM</u>

The FORM assembler directive may be used to define arbitrary word formats, label these formats, and thereafter reference the format by using the associated format label as an operation code in the operation field. When the assembler encounters a FORM directive, it notes the pattern specified in the operand field. Thereafter, the expressions in the operand field of the associated label, appearing as an operation code, will be interpreted and generated in the "form" specified by the initial directive.

In writing a FORM directive, the label field must contain a label, the operation field must contain the directive FORM, and the operand field must contain a series of expressions whose sum is equal to 25, the total number of bits in a UNIVAC III word (a single expression = 25 is illegal)

```
INST FORM     1,4,6,4,10

   INST       0,5 02, 3, 01400
```

In the above example:

The FORM directive has been used to define an object code format equivalent to a UNIVAC III instruction word. When INST is encountered by the assembler in the operation field of a symbolic line, the expressions in the operand field will be generated into a sign bit, 4 bit "b" field, 6 bit "op" field, 4 bit "a" field, and a 10 bit "m" field.

```
MASK FORM     1,3,3,3,3,3,3,3,3

  MASK        0,0,0,7,0,0,0,0,7
```

In the above example:

The FORM directive has been used to provide a simple means of writing a masking constant in octal mode equivalent to a UNIVAC III word. Whenever the label MASK appears in the operation field, the assembler will generate the appropriate masking constant. As illustrated in the second line above, the use of MASK in the operation field followed by the expressions 0,0,0,7,0,0,0,0,7 will generate a masking constant in the following pattern: + 000 000 111 000 000 000 000 111.

```
PRNT FORM    1,6,2,1,15

     PRNT    0,5,2,1,1004
     PRNT    0,2,0,1,0
     PRNT    0,6,2,1,13254
```

In the above example:

The FORM directive has been used to define a printer control word. The first example below the form directive will generate a line of object code which will cause the paper to be spaced 5 lines, and printing to take place from location 1004 through location 1035. The second example will cause the paper to be spaced 2 lines. The third example will cause the generation of a line which will cause the paper to be spaced 6 lines, and printing to take place from location 13254 through location 13285. In all cases interrupt is specified.

3. SET

The SET assembler directive may be used to arbitrarily indicate to the assembler that a specific value should be assigned to an index register for assembly purposes. The value assigned will be utilized by the assembler for automatic index register assignment until another SET directive specifying the same index register is encountered by the assembler. The assembler does not load the index register, that is the responsibility of the programmer. The format of a SET directive consists of SET in the operation field followed by two expressions. The first expression indicates the index register to be set, the second expression indicates the value to which the register is to be set.

```

     LX   15, $
     SET  15, $-1
```

In the above example:

Index Register 15 will be assumed by the assembler to contain the integer value equivalent to the current content of the location counter. The index register load instruction immediately preceding physically will accomplish the actual loading of IR 15 with the value of $.

f. FLD

The FLD assembler directive may be used to define the leftmost and rightmost bit limits of a field. A FLD directive line must have a label in the label field, FLD in the operation field, and the operand field must contain two expressions defining the left and right bit boundaries of the field. After a FLD directive has defined a field, the label may be used followed by the label in parentheses of the word(s) containing the field.

```
LMT     FLD     12,1

        LA      AR1,LMT(VALUE)
```

In the above example:

The label LMT has been defined as a field label through the use of the FLD directive. Its leftmost bit is bit 12, its rightmost bit is bit 1.

In the symbolic coding following, AR1 is being loaded from word VALUE as defined by the field LMT; i.e., bits 1-12 of word VALUE are being loaded into AR1.

g. END

The END assembler directive indicates to the UTMOST assembler that the last line of symbolic code in a program or procedure (PROC assembler directive) has been read by the assembler. This directive is required both at the end of a program and of a procedure. In the case of a procedure, the operand field is ignored by the assembler. In the case of a program, the starting address of the program should be placed in the operand field in the form of an expression.

```
J   NEXT
    END  STRT.   END OF PROGRAM
```

In the above example:

END indicates that the last line of coding in the program has preceded the END directive. The label STRT will be the starting address of the assembled program.

```
    END  END OF PROCEDURE: OPERAND
         FIELD IS IGNORED
```

In the above example:

END indicates that the last line of coding of a procedure has been read. The content of the operand field of a procedural END directive is ignored.

h. DO

The DO assembler directive may be used to optionally generate a line of coding a variable number of times. A DO symbolic line consists of an optional label, DO in the operation field, an expression in the operand field stating the number of times the DO is to be performed, and any symbolic line.

The format of a DO assembler directive is:

label DO e_1, line.

The label associated with a DO directive varies from the usual type of label in that, when referenced, its integer value will be equal to the number of times that the DO directive has been performed.

The expression of a DO directive, e_1, is a value which indicates to the assembler the number of times the associated line is to be generated. The 'line' may be any legitimate symbolic line of coding, or any directive except EQU, FORM, PROC, NAME, and END.

```
   DO   STRT+3072<$, USE 4,5,6
```

In the above example:

If the current value of the location counter is greater than the initial value of the location counter plus 3072 (3x1024), a 1 will be generated by the = operator. In that case, the assembler will be controlled by the USE directive line in the DO symbolic line, and three additional index registers will be set up by the assembler. If the condition is not met, a 0 will be generated, and the USE line will not become effective.

i. <u>PROC</u>

A PROC assembler directive informs the assembler that all succeeding symbolic lines until an END directive is read, are not to be assembled, but retained by the assembler until referenced by some other portion of the symbolic program. When the PROC (procedure) is referenced, the symbolic coding associated with the PROC will then be assembled and inserted into the object program.

A PROC directive line must have a label and the expression in the operand field indicates the maximum number of lists of expressions associated with the procedure, if any.* If no expression is given, the number of lists is indeterminate. (No expression is indicated by a period followed by a blank. In this case, every reference to the PROC must have a period followed by a blank following the last line.)

* A discussion of PROC lists follows under the NAME directive.

```
TRAN      PROC 0
          LA   15,0,8
          SA   15,0,9
          IX   8,4
          IX   9,4
          END
```

In the above example:

The PROC line has the label TRAN (for TRANsfer), PROC in the operation field and a 0 in the operand field indicating that there are no lists associated with the PROC. The four lines of coding following make up a very simple straight line four word transfer routine followed by an END directive.

The previous procedure may be referenced by the following symbolic coding:

```
    LX   8,   *(RESERVE)
    LX   9,   *(CURRENT)
    DO   5,   TRAN
```

The DO directive line will cause the procedure to be generated five times, since the expression in the DO line is 5, effectively generating the following symbolic coding transferring twenty words.

```
1   LABEL  Δ  OPERATION  Δ  OPERAND
    LA   15,0,8
    SA   15,0,9
    IX   8,(4)
    IX   9,(4)
    LA   15,0,8
    SA   15,0,9
    IX   8,(4)
    IX   9,(4)
    LA   15,0,8
    SA   15,0,9
    IX   8,(4)
    IX   9,(4)
    LA   15,0,8
    SA   15,0,9
    IX   8,(4)
    IX   9,(4)
    LA   15,0,8
    SA   15,0,9
    IX   8,(4)
    IX   9,(4)
```

j. <u>NAME</u>

A NAME directive, or several NAME assembler directives, may be used to qualify a PROC procedure. The NAME line(s) must follow the PROC line within the procedure. Each NAME line must have a label, and may have an expression in the operand field.

A procedure may be referenced by placing any of the procedure names or the label associated with the PROC line in the operation field of the referencing line.

```
FDOL    PROC  0
ADOL    NAME  0
NDOL    NAME  1
```

In the above example:

The procedure is a routine to generate a floating dollar sign edit routine. The two names applying to the routine are ADOL and NDOL respectively, ADOL if the value to be edited is in six bit excess three format, and NDOL if the value is in 4 bit numeric format.

```
ARRANGE  LA  7, 23, 7
  ADOL
  SA  16,  SAVE+3
```

The coding above references the floating dollar subroutine. Since the alphanumeric variant of the routine is applicable to the data to be edited, the subroutine is called by writing the NAME of the alphanumeric version in the operation field, ADOL and since there are no lists required by the routine, nothing need be written in the operand field of the symbolic line. When the assembler encounters this symbolic line, the floating dollar procedure will be generated and inserted in the program at this point.

k. Procedure Lists

Procedures may be written referencing lists of variables which are submitted by the calling program. During the assembly of the procedure, when variables are required, the assembler will call upon the lists submitted with the calling line.

1) PROC symbolic line: As stated under the PROC directive, the PROC symbolic line consists of a label, PROC in the operation field, and an expression in the operand field indicating the number of lists expected by the procedure during generation. If the procedure expects a variable number of lists, the expression should be a period followed by a blank.

```
FDOW  PROC 0
```

In the above example:

The PROC line states that the procedure does not require any lists.

```
MOVE  PROC.
```

In the above example:

The PROC line states that the procedure requires a variable number of lists.

2) List References within a procedure: When information is required by a procedure from the calling program, it is obtained by referencing the label of the procedure by an expression in the operand field stating the procedure label.

   a) To reference an expression within a list, the expression is written as: label (s, e) where label is the label of the procedure, s is the number of the list, and e is the number of the expression within list s.

LABEL	OPERATION	OPERAND
	LX	MOVE(1,4), (MOVE(1,1))

In the example above which is taken from the MOVE PROC, line 22:

MOVE(1,4) refers to list #1, 4th expression in the calling symbolic line in the main program. In this case, it would be the number of an index register.

MOVE(1,1) refers to list #1, 1st expression. This expression within the list provides the address of the first word to be moved.

b) To reference the number of lists supplied by the calling symbolic line in the main program, the expression is written as: label where label is the label of the procedure. The assembler will substitute the number of lists currently submitted by the referencing line as the integer value of the expression.

LABEL	Δ	OPERATION	Δ	OPERAND
DO		MOVE>3,		A

In the above example:

The condition MOVE>3, refers to the number of lists submitted by the referencing line in the main program. If the number of lists is greater than three an integer 1 will be generated.

If the expression had been written:

MOVE(1)>3,

it would refer to the number of expressions within the first list of the referencing line.

c) To reference the expression in the operand field of a NAME line within a procedure, the expression is written as: label (0, 0) where label is the label of the procedure, and (0, 0) is the operand field of the NAME line which is currently referencing the procedure.

```
FDOL  PROC   0
ADOL  NAME   0
NDOL  NAME   1

 DO  FDOL(0,0)=0, SA   7, TEMP+2
 DO  FDOL(0,0)=1, SAA  3, TEMP+2
```

In the above example:

The operand field of the ADOL NAME line is 0, the operand field of the NDOL Name line is 1. The two DO lines reference the procedure label, FDOL, with the expression FDOL(0, 0) and the equal operator will generate an integer 1 in whichever line the condition is met, causing the associated line to be generated once. In this way, the assembler has determined which NAME was used to reference the procedure in the main program.

3) References to a procedure from outside the procedure:
The label of the appropriate procedure or qualifying NAME line is written in the operation field of the referencing line. It is followed by the lists of parameters required by the procedure, if any.

LISTS

When referencing a procedure, the operand field of the calling line contains the lists required by the procedure. A list consists of a series of expressions separated by commas. Lists are separated by blanks. If the PROC line contains a period followed by a blank in the operand field indicating an indeterminate number of lists required by the procedure, the last list of the calling line must be terminated by a period followed by a blank.

```
 ADO4
```

In the above example:

The floating dollar procedure requires no lists, therefore the operand field of the calling line will be ignored by the assembler.

```
  ST   0,8  0,9  50.
```

In the example above:

The MOVE procedure requires a variable number of lists. The example line calls for straight line move coding to be generated through the use of the ST name in the operation field. Three lists are submitted. The lists are terminated by a period followed by a blank since the MOVE procedure calls for an indeterminate number of lists.

```
CHANGE  IT  HERE,0 THERE,0  50  10.
```

In the example above:

The example line calls for iterative coding to be generated. Four lists are submitted. The expressions within the lists are separated by commas, the lists by a blank. The last list is terminated by a period followed by a blank.

9. Sample Problem--Two Way Merge with Editing

The attached sample problem is deliberately simple and designed to illustrate a number of the features of the UTMOST assembler for UNIVAC III. It consists of a basic business oriented two way merge between a master file and a change file. Where record identifiers are identical, the change record is substituted for the master record, and the dollar value of the change record edited by a floating dollar sign editing routine in preparation for printing. In addition, the floating dollar sign routine is generated by a procedural reference, and all data transfers are accomplished by a MOVE procedure which will provide iterative or straight line transfers at the option of the user.

Input/output record advance routines are shown as subroutines, but not included within the coding. (All input/output area addresses are supplied by the record advance routines in Index Registers at the time of return to the main program.)

PROGRAM 2 WAY MERGE W/EDIT, PRINT PROGRAMMER BOB VARNEY DATE ______ PAGE 1 OF 2 PAGES

```
1 LABEL  Δ  OPERATION  Δ  OPERAND  Δ  COMMENTS                                              72|73    80
  USE 11,2,3.        SET UP INDEX REGISTERS FØR AREA ADDRESSING
  RES 32             PRINTER WØRKING STØRAGE
STRT NØP.
PRNT EQU $-32        PRØVIDE BASE ADDRESS FØR PRINTER EDIT AREA.
  LX 8,IN-M.         LØAD IR'S 7, 8, 9 WITH BASE ADDRESSES ØF INPUT AND ØUTPUT AREAS
  LX 9,ØUT-M.        FØR MASTER AND CHANGE FILES AND PRINTER ØUTPUT FILE AREA.
  LX 7,CHNG
  LX 10,PRINTER.
ØNE  LA  15,3,8      SET UP COMPARISØN FØR MATCH ØR MERGE PRØCEDURE
  CM 15,3,7
  JH  CHANGE.        IF MASTER REC. HAS HIGHER SEQUENCE NUMBER WRITE CHANGE RECØRD
  JE  EDIT.          IF SEQUENCE NØ.S AGREE, MAKE CHANGE AND PERFØRM PRINT EDIT.
  ST 0,8 0,9 50.     INSERT STRAIGHT LINE TRANSFER MASTER INPUT TØ ØUTPUT PRØCEDURE
  SLJ  M-ØUT.        EXECUTE ØUTPUT RECØRD ADVANCE (WRITE).
  SLJ  M-IN.         EXECUTE MASTER RECØRD INPUT ADVANCE (READ).
  J ØNE              RETURN TØ PRØCESS NEXT RECØRD
CHANGE IT 0,7 0,9 50.  INSERT ITERATIVE TRANSFER PRØCEDURE AT THIS PØINT.
  SLJ  M-ØUT.        EXECUTE MASTER RECØRD WRITE SUBRØUTINE.
  SLJ  CHNG-IN.      EXECUTE RECØRD READ SUBRØUTINE.
  J ØNE              PRØCESS NEXT RECØRD
EDIT SLJ ARRANGE.    EXECUTE EDITING RØUTINE.
  SLJ PRINTØUT.      EXECUTE PRINTER TAPE WRITE SUBRØUTINE.
  IT 0,6 0,9 50.     GENERATE MØVE CØDING.
  SLJ M-ØUT.         EXECUTE WRITE SUBRØUTINE.
  SLJ M-IN.          EXECUTE READ SUBRØUTINE
```

UP-2507 CODING FORM ING. BK.

PROGRAM 2 WAY MERGE W/EDIT, PRINT PROGRAMMER BØB VARNEY DATE ______ PAGE 2 OF 2 PAGES

```
1 LABEL  Δ OPERATION    Δ  OPERAND           Δ          COMMENTS                 72|73      80
 SLJ CHNG-IN.               EXECUTE CHANGE READ SUBRØUTINE
 J ØNE                      PRØCESS NEXT RECØRD
 NØP
ARRANGE LA 7,23,7           LØAD VALUE TØ BE EDITED INTØ AR'S 2,3,4
 ADØL                       GENERATE FLØATING DØLLAR EDIT CØDING
 SA 15,PRNT+29              STØRE EDITED DØLLAR VALUE
 LA 15,3,7                  EDIT RECØRD IDENTIFIER FØR PRINTING
 SA 15,PRNT+5
 IT 0,7 0,8 50              GENERATE ITERATIVE MØVE CØDING FØR RECØRD INSERTIØN
 J *ARRANGE-1.              EXIT
   END STRT.                END ØF PRØGRAM.
```

UP-2507 CODING FORM ING. BK.

10. Sample Floating Dollar Sign Editing Procedure

The following coding represents a procedure designed to edit an 11 character field, inserting a decimal point, commas where required, and floating a dollar sign to the character position immediately preceding the first significant digit in the field.

The procedure will accept either 6 bit alphanumeric values or 4 bit numeric values, and the coding generated is dependent upon the name by which the procedure is referenced in the main body of coding.

Programming reference:

The routine expects the value to be edited to be present in AR's, 2, 3, and 4 if in alphanumeric format, in AR's 3 and 4 if numeric format. To call the Alphanumeric version, ADOL should be written in the operation field of the line where it is desired to generate the routine. If the numeric version is desired, NDOL should be written in the operation field of the referencing line.

PROGRAM FLØATING DØLLAR PRØCEDURE PROGRAMMER BØB VARNEY DATE PAGE 1 OF 5 PAGES

```
LABEL  Δ OPERATION  Δ OPERAND                 Δ COMMENTS
FDØL   PRØC  0
ADØL   NAME  0
NDØL   NAME  1
ØWE    J  $+7
TEMP   RES  7
  DØ   FDØL(0,0)=0,SA 7,TEMP+2
  DØ   FDØL(0,0)=1,SAA 3,TEMP+4,
  LA   1,(0)
  C  1,TEMP+2
  JG  $+3
  SA  1,TEMP+6
  J  $+3
  LA  1,('-')
  J  $-3
  LA  8,(01700000)
  LAE 7,TEMP+2
  SA  7,TEMP+2
  LA  4,(9)
  CPZ 8,TEMP
  JG  C
  BS  4,(1)
  ASR 8,1
  CPZ 8,TEMP
  JG  C
  BS  4,(1)
```

UP-2507 CODING FORM ING. BK.

REVISION:

SECTION: II

DATE: July 1, 1962

PROGRAM FLØATING DØLLAR PRØCEDURE PROGRAMMER BØB VARNEY DATE ______ PAGE 2 OF 5 PAGES

```
1 LABEL  Δ  OPERATION  Δ  OPERAND  Δ  COMMENTS                72 73   80
       ASR   8,1
       CPZ   8,TEMP
       JG    C
       BS    4,(1)
       LA    8,(0770000000)
       CPZ   8,TEMP+1
       JG    C
       BS    4,(1)
       ASR   8,1
       CPZ   8,TEMP+1
       JG    C
       BS    4,(1)
       ASR   8,1
       CPZ   8,TEMP+1
       JG    C
       BS    4,(1)
       ASR   8,1
       CPZ   8,TEMP+1
       JG    C
       BS    4,(1)
       LA    8,(0770000000)
       CPZ   8,TEMP+3
       JG    C
       BS    4,(1)
       ASR   8,1
```

UP-2507 CODING FORM ING. BK.

PROGRAM FLØATING DØLLAR PROCEDURE PROGRAMMER BOB VARNEY DATE ______ PAGE 3 OF 5 PAGES

LABEL Δ OPERATION Δ OPERAND Δ	COMMENTS	73–80
`CPZ 8,TEMP+2`		
`JG G`		
`ASR 8,1`		
`CPZ 8,TEMP+2`		
`JG B`		
`ASR 8,1`		
`CPZ 8,TEMP+2`		
`JG A`		
`ZER TWC (0)`		
`LA 12,ZER`		
`LA 3,ZER`		
`J END`	EXIT	
`A LA 1,('.0   ')`		
`ØR 1,TEMP+6`		
`SA 1,TEMP+6`		
`LA 1,TEMP+2`		
`ASL 1,1`		
`ØR 1,TEMP+6`		
`LA 2,('$')`		
`LA 12,ZER`		
`J END`	EXIT	
`B LA 1,TEMP+2`		
`ASL 1,1`		
`ØR 1,('.     ')`		
`J $-7`		

UP-2507 CODING FORM ING. BK.

PROGRAM FLØATING DØLLAR PRØCEDURE PROGRAMMER BØB VARNEY DATE PAGE 4 OF 5 PAGES

```
LABEL  OPERATION  OPERAND                          COMMENTS
D      TWO        ('   ,     ,   ')
C      LA         3,D
       SA         3,TEMP+5
       LA         1,TEMP+2
       ASL        1,1
E      FLD        24,19
       LF         1,E('.   ')
       ØR         1,TEMP+6
       SA         1,TEMP+6                         CENTS IN TEMP+6
       LA         3,TEMP+2
       ASR        3,1
       ASR        1,1
       ØR         3,TEMP+5
       SA         3,TEMP+5                         INSERT ALL CØMMAS
       LA         1,TEMP
       SA         1,TEMP+3
       LA         1,(10)
       C          1,TEMP+3
       JE         E
       BS         4,(7)
       SA         4,TEMP
       LA         8,('$ ')
       ASL        8,*TEMP
       LA         7,TEMP+6
       ØR         8,TEMP+3
```

PROGRAM FLOATING DOLLAR PROCEDURE PROGRAMMER BOB VARNEY DATE PAGE 5 OF 5 PAGES

```
LABEL  OPERATION  OPERAND            COMMENTS
       J          $+20               EXIT
E      LAE        11,TEMP+6
       C          4,(3)
       JG         $+11
       JE         $+8
       C          4,(2)
       JE         $+4
       OR         2,('$  ')
       LA         14,TEMP+5
       J          $+14
       OR         2,('$   ')
       J          $-3
       OR         2,('$    ')
       J          $-5
       BS         4,(4)
       SA         4,TEMP
       LA         4,('$  ')
       AS4        4,*TEMP
       OR         4,TEMP+4
       LA         8,(0)
END               END
```

UP-2507 CODING FORM ING. BK.

11. Sample MOVE PROCEDURE

This MOVE PROC is a generalized routine to move n words from one area in memory to another. It is activated and appropriate coding generated by a procedure reference line: one of the following

IT	Label	Label	# of words	IR	(4 lists)
ST	Label	Label	# of words		(3 lists)
IT	0, IR	0, IR	# of words		(3 lists)
ST	0, IR	C, IR	# of words		(3 lists)

The above reference lines indicate that the sending and receiving addresses may be given as a label or in an index register. If iterative coding is called for but the number of words (list 3) is not greater than twenty, then straight line coding will be provided. This allows the number of words to be computed elsewhere in the program and the routine to determine the better coding.

The MOVE procedure is composed of a number of procedures to determine which coding should be generated and how much coding is needed in the case of straight line coding.

Lines 1 - 6

The opening lines are the entrances to the MOVE PROC. The period in the MOVE statement indicates that the number of lists provided to the PROC is variable. The DO statement in line 4 tests to see whether STraight line or ITerative coding is called for. If ITerative coding is desired PROC A will be generated, if STraight line coding is desired, PROC B will be generated.

Lines 7 - 10 PROC A

PROC A is reached by IT in the reference line. These lines further determine whether the addresses (sending and receiving) were given as a label or in an index register.

Lines 11 - 14 PROC B

PROC B makes the same test as PROC A, but the switches are different as they must create coding to handle straight line coding.

Line 15 - 18 PROC C

PROC C performs the test for the number of words to be moved. It is generated in PROC A and therefore is a continuation of the coding necessary to generate iterative coding with an address supplied in an index register. If the number of words were 20 or less, then straight line coding would be generated.

Lines 19 - 22 PROC D

This procedure makes the same test as PROC C but sets the switches so that the coding generated will handle the words to be moved with labels provided instead of in an index register.

Lines 23 - 36 PROC E

PROC E would be generated if there were more than 20 words to be moved and the addresses to be manipulated were in index registers.
Line 24
First a test is made to determine and move any words not multiples of four. PROC L would be called for and it has one list. The expression given would create the correct bit pattern to be placed in the AR portion of the word. Lines 25 - 30 are used to manipulate the beginning address and create the proper increment and compare control word for use in iteration.

Lines 31 - 35 compromise the entire coding needed to move four words iterating on index register given as containing the beginning area address. Line 36 is the conclusion of a PROC, an END line.

Lines 37 - 44 PROC F

PROC F, generated by PROC D, moves the words iteratively; the addresses having been supplied as labels. Note in this PROC that the non-multiples of four words are moved at the conclusion of the 4-word-multiples.

Lines 45 - 48, PROC G

This PROC isused by both straight line and iterative coding procedures to move non-multiples of four when the addresses were given in labels rather than index registers.

Lines 49 - 52 PROC H

These lines would be generated if straight line coding would be desired and the area address were given as labels. The first DO determines if there are any non-multiples of four words and generates a PROC to move them.

Line 51 creates the number of four word loads and stores necessary to move all multiples of four. The DO statement has a "label" which will be used by the M PROC called for in this DO line.

Lines 53 - 56 PROC J

This PROC accomplishes the same thing as PROC H, but the switches here would call for a PROC necessary to create straight line coding where the index register contain the area addresses.

Lines 57 - 60 PROC K

This PROC contains the two four word load and store lines for straight line coding. Note the use of the indexing feature of the DO "label" to increment the m address. Each time the coding is generated the COUNT will be one greater and when multiplied by four will give the proper address increment.

Lines 61 - 64 PROC L

This PROC is called for in PROC E where the non-multiples of 4 have to be moved before the iterative process can commence.

Lines 65 - 68 PROC M

This PROC generates the coding necessary to move words in straight line coding, but it differs from PROC K, in that the addresses of the sending and receiving areas are in index registers. Notice the use of the indexing feature of a DO "label".

PROGRAM MOVE PROC PROGRAMMER DON PRIGGE DATE July 1, 1962 PAGE 1 OF 3 PAGES

```
1 LABEL  Δ OPERATION  Δ OPERAND                          Δ COMMENTS   72 73      80
MOVE       PROC.                                                         1
IT         NAME 0                                                        2
ST         NAME 1                                                        3
           DO      MOVE(0,0)=0, A                                        4
           DO      MOVE(0,0)=1, B                                        5
           END                                                           6
A          PROC.                                                         7
           DO      MOVE(1,1)=0, C                                        8
           DO      MOVE(1,1)>0, D                                        9
           END                                                           10
B          PROC.                                                         11
           DO      MOVE(1,1)=0, H                                        12
           DO      MOVE(1,1)>0, J                                        13
           END                                                           14
C          PROC.                                                         15
           DO      MOVE(3,1)>20, E                                       16
           DO      MOVE(3,1)<21, H                                       17
           END                                                           18
D          PROC.                                                         19
           DO      MOVE(3,1)>20, F                                       20
           DO      MOVE(3,1)<21, J                                       21
           END                                                           22
E          PROC.                                                         23
           DO      MOVE(3,1)**3>0, L (3**MOVE(3,1)*3-3)/2+4              24
           SX      MOVE(1,2), TEMP                                       25
```

UP-2507 CODING FORM ING. BK.

PROGRAM MOVE PROC PROGRAMMER DON PRIGGE DATE JULY 1, 1962 PAGE 2 OF 3 PAGES

```
LABEL  OPERATION  OPERAND                                              COMMENTS  73-80
       LA     8, TEMP                                                            26
       BA     8, 4*(MOVE(3,1)/4)                                                 27
       BRR    8, 15                                                              28
       OR     8, (4)                                                             29
       SA     8, TEMP                                                            30
       LA     15, 3, MOVE(1,2)                                                   31
       SA     15, 3, MOVE(2,2)                                                   32
       IXC    MOVE(1,2), TEMP                                                    33
       IX     MOVE(2,2), (4)                                                     34
       JL     $-4                                                                35
       END                                                                       36
F      PROC.                                                                     37
       LX     MOVE(4,1), (MOVE(1,1))                                             38
       LA     15, 3, MOVE(4,1)                                                   39
       SA     15, 3, MOVE(2,1)-MOVE(1,1)+3                                       40
       IXC    MOVE(4,1), ((4*(MOVE(3,1)/4))+MOVE(1,1), 4)                        41
       JL     $-3                                                                42
       DO     MOVE(3,1)**3>0, G   (3**MOVE(3,1)*3-3)/2+4                         43
       END                                                                       44
G      PROC 1                                                                    45
       LA     G(1,1), MOVE(1,1)+MOVE(3,1)-1                                      46
       SA     G(1,1), MOVE(2,1)+MOVE(3,1)-1                                      47
       END                                                                       48
H      PROC.                                                                     49
       DO     MOVE(3,1)**3>0, L   (3**MOVE(3,1)*3-3)/2+4                         50
```

UP-2507 CODING FORM ING. BK.

PROGRAM MOVE PROC   PROGRAMMER DON PRIGGE   DATE JULY 1, 1962   PAGE 3 OF 3 PAGES

```
1 LABEL  Δ OPERATION  Δ OPERAND                                     Δ   COMMENTS    72|73     80
ADD        DO         MOVE(3,1)/4,                                                  51
           END                                                                      52
J          PROC.                                                                    53
           DO         MOVE(3,1)**>0,    G    (3**MOVE(3,1)*3-3/2+4                  54
COUNT      DO         MOVE(3,1)/4,      K                                           55
           END                                                                      56
K          PROC.                                                                    57
           LA         15, MOVE(1,1)+(4*COUNT-1)                                     58
           SA         15, MOVE(2,1)+(4*COUNT-1)                                     59
           END                                                                      60
L          PROC 1                                                                   61
           LA         L(1,1), MOVE(3,1)-1, MOVE(1,2)                                62
           SA         L(1,1), MOVE(3,1)-1, MOVE(2,2)                                63
           END                                                                      64
M          PROC.                                                                    65
           LA         15, 4*ADD-1, MOVE(1,2)                                        66
           SA         15, 4*ADD-1, MOVE(2,2)                                        67
           END                                                                      68
```

UP-2607 CODING FORM ING. BK.

III. PROGRAMMERS' REFERENCE SECTION

A. LINE CONTROL

The information content of a line to the assembler consists of the label, operation and operand fields. The information content is normally terminated when the maximum number of expressions required by the operation have been encountered (or maximum number of lists in the case of a procedure reference).

There are two special marks which override the normal rule:

1. Continuation

   If a ";" is encountered (outside of an alphabetic item) the current line is continued with the first non-blank on the following line, and there is no more information to the assembler on this line.

2. Termination

   If a "." followed by a blank is encountered (outside of an alphabetic item) the line is terminated at this point. If any more expressions are required, they are taken to be zero.

A continuation or termination mark may occur anywhere on the line. Following the information content of a line any characters may be entered.

B. LABEL FIELD

If a line is to have a label, it is written in the label field. A label is composed of one to eight alphanumeric characters, the first of which is an alphabetic character. The label field starts in column one and is terminated by a blank. Except for the EQU, FORM, DO, FLD, PROC and NAME directives, the label is equated to the current value of the location counter.

C. OPERATION FIELD

The operation field is up to eight characters in length, and may contain an assembler directive, a mnemonic machine operation code, a label associated with the FORM, PROC or NAME directive, or a data generating code. The operation field starts in the first non-blank following the label field and is ~~terminated by a blank unless it consists of a + (plus) or - (minus) sign,~~ in which case the + or - signs is the operation field and the next column need not be blank. If the operation field contains an assembler directive other than RES (which increments the location counter), the location counter will not be affected. If the operation field contains TWC, the location counter is incremented by two. In all other cases, the location counter is incremented by one after the line is generated.

D. OPERAND FIELD

The operand field starts in the first column following the operation field and is composed of lists of expressions. Lists are separated by blanks. The number of lists is one except in the case of a procedure reference line. Each expression in a list except the last is terminated by a comma.

E. EXPRESSIONS

An expression is an elementary item or a series of elementary items connected by the operators shown in the table below. An item may have preceding blanks.

+	Arithmetic Sum	
-	Arithmetic Difference	
*	Arithmetic Product	
/	Arithmetic Quotient	
++	Logical Sum (OR)	
--	Logical Difference (exclusive or)	
**	Logical Product (AND)	
//	Covered Quotient $(a//b=\frac{a+b-1}{b})$	

=	Equal	a=b is 1 if a=b
		a=b is 0 if $a \neq b$

>	Greater Than	a>b is 1 if a>b a>b is 0 if $a \leq b$
<	Less Than	a<b is 1 if a<b a<b is 0 if $a \geq b$
*+	$a*+b=a*10^{b}$	
*-	$a*-b=a*10^{-b}$	

An expression may also have a leading + or - sign. Any negative value produced by an expression will be represented by a 2's complement unless the operation field of the line contains an EQU assembler directive, or TWC, or, in some cases, if the operation field is + or -.

If an expression represents an address, it may be preceded by an *. This will cause the sign of the generated word containing the expression to be - (indirect address or field select).

The various types of items and their values are given in the following table.

TYPE	FORM	VALUE	EXAMPLE
Label	any label	value assigned to label	L
Location	$	value of location counter	$
Octal	the digit 0 followed by octal (0-7) digits	value interpreted as base 8 (binary representation)	017
Decimal	non-zero digit followed by decimal (0-9) digits	value interpreted as base 10 (binary representation)	14
BCD	: followed by decimal digits	value interpreted as base 16 (Excess 3)	:14
Alphabetic	' (apostrophe) followed by any characters except ' followed by '	value of each character in corresponding position	'BOB'
Floating	decimal digits followed by . followed by decimal digits	values represented in internal floating point format (always double precision)	3.14
Field	field label followed by expression enclosed in parentheses	address of word selecting the field	OP ($ + 2)
Parameter	procedure label or procedure label followed by 1 or 2 expressions enclosed in parentheses	value of corresponding parameter as defined by the current reference (see Procedure Reference)	MAX (2, 1)
Line *	(followed by line followed by)	value of the word the line would generate	(J $ + 2)

All items in the above table will be right justified in their generated resultant field, and leading bit positions will be binary zeros.

* See description of line item.

F. MNEMONIC INSTRUCTIONS

The operation field may contain any of the mnemonic instruction names listed in Appendix 1. The instructions are of two types. Type 0 instructions have three expressions representing the "a", "m" and "b" fields of the instruction respectively. Type 1 instructions have two expressions representing the "m" and "b" fields of the instruction respectively. The absolute operation code is placed in the operation field of the instruction word and, if the instruction is type 1, the absolute "a" register code listed is placed in the "a" field of the instruction word. These fields are described by the format:

	24 21	20 15	14 11	10 1
S	b	op	a	m

The sign of the instruction will be + unless the first character of "m" is * (indirect address or field select) or an implied literal is generated (see Section I).

G. DATA WORD GENERATION

There are two methods of indicating a data word (other than an instruction).

1. Increment and Compare Word, ICW

This data generation operation is used to prepare a word suitable for incrementing and compating an index register (with the IX and IXC instructions). It is followed by two expressions: e_1 representing the comparison amount, and e_2 representing the increment.

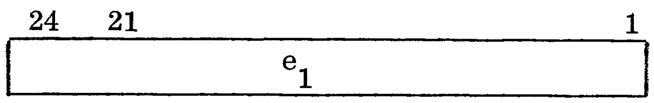
The format of the generated word is illustrated below:

		24 10	9 1
ICW	S	e_1	e_2

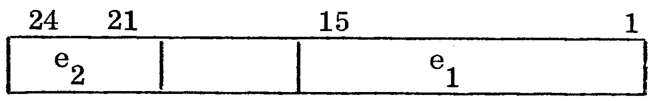
The sign of the word generated is the sign of e_2 and bits 9 to 1 contain the magnitude of e_2 mod 512.

2. <u>+ or - Operation Field</u>

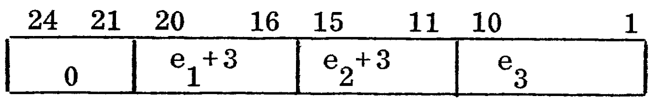
A + or - operation field causes generation of a one-word constant whose format depends upon the number of expressions in the operand field. The formats generated for the corresponding number of expressions are described below:

1 S


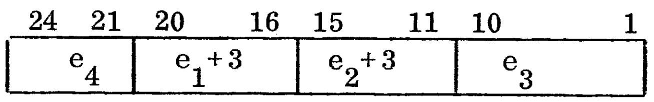
one-word datum

2 S


indirect address word

3 +


field select word

4 +


field select word

3. <u>Two Word Constant, TWC</u>

A TWC data generating word will actually generate two words. All floating point expressions should be preceded by TWC (except in the case of literals). The sign of both words will be the same and equal to the sign of the value of the expression given.

H. LINE ITEM

A line item is an instruction line, form reference line, or data word line without label field and without leading or trailing blanks, enclosed in parentheses. The line item has the value which the word generated by the line would have unless the line occurred in the address field of an IX or IXC instruction and has two expressions. In this latter case, it is evaluated as a data word with ICW in the operation field. If the line is a data word line, the leading + or - may be omitted. If an entire expression (except for possible leading *) consists of such an item, the value of the expression is the address of the cell containing the word generated by the line. The word generated is called a literal. If the literal is identical to any other literal, the location assigned is the location of the previous literal, thus eliminating duplication.

A literal will be double precision if the line was a "TWC" line or if it was a data line with one expression and the mode of the expression was floating.

An item within such an item can be of this type up to a level of 8 parentheses.

I. ADDRESSING

The programmer writes addresses as if they were 15-bit quantities and normally is not concerned with the fact that they are 10-bit quantities. The resultant object code generated depends upon which of the following cases is satisfied (where m represents the value of the address expression and b represents the value of the index expression of an instruction and x_i are the index registers assigned to the assembler by USE directives).

1. $m < 2^{10}$

	24 21	20 15	14 11	10 1
S	b	op	a	m

2. b = 0, and $m \geq 2^{10}$ and for some i $0 \leq m - (x_i) < 2^{10}$

	24 21	20 15	14 11	10 1
S	x_i	op	a	$m-(x_i)$

3. If neither 1 nor 2 is satisfied, the object code generated will be identical to that which would have been generated if the programmer had enclosed m, b in parentheses and preceded the left parenthesis by an *. (This is an implied literal.)

4. If the address addresses a literal location, y, (implied or otherwise) and does not satisfy $0 \leq y - (x_i) < 2^{10}$ for any i, a range error flag is set and the address contains y (mod 2^{10}).

Note: In 1 and 2, S is + unless the first character of m is *.

J. ASSEMBLER DIRECTIVES

Assembler directives supply information to the UTMOST assembler. There are several assembler directives as listed below and described on succeeding pages. Any labels referred to in an expression on a directive line must have been previously defined (i.e., they must have previously appeared in the label field).

1. EQU
2. RES
3. FLD
4. FORM
5. END
6. PROC
7. NAME
8. DO
9. USE
10. SET

1. EQU

The EQU assembler directive causes the label in the label field of its symbolic line to be equated to the value of the expression in the operand field of the symbolic line.

FORMAT: label EQU e_1

2. RES

The RES assembler directive causes the value of the expression in the operand field to be added to the location counter.

FORMAT: RES e_1

3. FLD

The FLD assembler directive is utilized to indicate the leftmost and rightmost bit limits of a field. It must have a label. The first expression represents the leftmost bit limit, the second expression, the rightmost bit limit.

FORMAT: label FLD e_1, e_2

USE FORMAT: op AR, label (m)

When a field reference item is used as an address, a field select literal selecting the field is generated and the address is the address of this literal. The sign of the instruction generating the literal is minus.

4. FORM

The FORM assembler directive is used to define arbitrary data formats. This directive must have a label in the label field, and the sum of the values of the expressions in the operand field must equal 25. A single expression equal to 25 is not permitted.

The FORM directive permits the programmer to define arbitrary word formats by calling upon the pattern specified with a line of coding having the associated label in the operation field and the appropriate number of expressions in the operand field.

FORMAT:	label FORM $e_1 \ldots e_1$;
REFERENCE:	label e_1, e_2, ... e_n

5. END

The END assembler directive indicates to the assembler that the last line of symbolic coding for the procedure or program has been read by the assembler. In the case of a procedure, the operand field is ignored. In the case of an entire program, the expression in the operand field represents the starting address.

FORMAT: END e_1.

6. PROC

A PROC directive line must have a label, and the expression in the operand field indicates the maximum number of lists of expressions associated with the procedure (if any). If no expression is given, the number of lists is indeterminate. (No expression is indicated by a period-blank. In this case, every reference to the PROC must have a period-blank following the last list).

A procedure must be defined previous to any references to the procedure.

The PROC line is (optionally) followed by NAME lines (see NAME directive) and any valid symbolic lines up to and including an END line. If there are n intervening PROC lines, the n + first END line will terminate the procedure.

Any labels defined within the procedure are considered not defined outside the procedure unless the label is followed by an "*", in which case the label is treated as if it appeared in the referencing procedure without an asterisk. If a label is referred to within the procedure and is not defined within the procedure, the definition of the label outside of the procedure (if any) is taken.

7. NAME

All NAME directives associated with a given procedure must follow the PROC line immediately. A NAME line must be given a label. Its operand field contains an expression.

FORMAT: label NAME e_1

A procedure may be referenced by placing any of the Procedure names (including the name on the procedure line) in the operation field of a line.

8. DO

The DO directive is used to generate a line a given number of times. If a label is present, the value of the label will be n the n'th time the line is done. The expression in the operand field indicates the number of times the line is to be done. The line may be any line of symbolic coding except EQU, FORM, PROC, NAME and END.

FORMAT: label DO e_1, line of coding

9. USE

This directive is followed by not more than 16 expressions which represent index registers. The first of these registers is assigned the current value of the location counter. Succeeding registers are assigned the value of the preceding register plus 2^{10}. These registers are loaded with their assigned values when the program is loaded and cannot be modified by the program unless a SET directive is given referring to the register. The same index register should not appear in more than one USE directive.

10. SET

The SET directive has two expressions. The first expression represents an index register and the second expression represents a memory address. The assembler will assume the value given is in the index register from the point the set is given until another set referring to the same register is given.

The register is essentially a "USE" register and the information supplied by the SET directive will be used for addressing purposes as explained under "ADDRESSING".

Note that the assembler will not cause the register to be loaded.

K. PROCEDURE REFERENCE LINE

Lists of variables may be submitted when referincing a procedure. Expressions within a list are separated by commas; lists are separated by blank columns.

If the name of the procedure is P, within procedure coding, P refers to the number of lists supplied by the current reference, P(e) refers to the number of expressions in the e'th list and P(e, f) refers to the value of the f'th expression of the e'th list (e and f are expressions). The list containing the procedure name (operation field) is considered list 0 and is always present. The procedure name may be followed by expressions. P (0, 0) refers to the value of the expression on the NAME line by which the procedure was referenced, and P (0, e) refers to the e'th expression in the name list (list 0).

L. INTER-PROGRAM COMMUNICATION

1. Definition

If a label in the label field is immediately followed by an "*" and the line is not within a procedure, this is an external label which can be referenced by other programs, assembled separately, when the set of programs is loaded. References to the external label in the program which defines it are the same as for any other label.

2. References

If an address expression consists of a label plus or minus a constant, and the label is not defined within this program, a reference to an external label will be generated.

Operating procedures will be specified later.

Section V is a reprint of UT 2465, the UNIVAC III Central Processor Manual, with illustrations changed to the UTMOST language and with notes brought up to date by the latest information on the hardware aspects of the computer. It is here included in order to make this manual as comprehensive as possible.

CENTRAL PROCESSOR

The Central Processor consists of five modules: the memory unit, the arithmetic and control unit, the general purpose channels, the power supply and the power control. The functions of the first three are described below.

Control Unit

The control unit contains a number of special registers and additional circuitry whose functions are to select in proper sequence, interpret, and initiate the execution of the individual instructions of the stored program governing the operations of the entire system. The instruction logic is 1–address and the instructions are executed sequentially.

In addition to the normal sequencing, addressing, and control registers, the control unit includes up to 15 index registers, and a Memory Address Adder. The Memory Address Adder is separate from the adder of the arithmetic unit. The index registers together with the special adder permit the system to make the indexing cycle an integral part of the instruction set-up cycle. Therefore, no additional memory cycles are required for indexing. The instruction execution cycle is explained in detail in Section 3.

Arithmetic Unit

The arithmetic unit contains an adder for both decimal and binary arithmetio, four arithmetic registers, and additional circuitry to permit a wide range of logical abilities.

Addition in the UNIVAC III System is parallel by bits of a digit and serial by digits. Because the digit rate through the adder is ½ microsecond, the serial additions of the six digits within a word are completed in the 4–microsecond basic memory cycle.

The four arithmetic registers can be linked in all processing operations to permit the handling of two- three - or four-word operands. Utilizing this feature, the programmer is able to reference, with a single instruction, 4, 8, 12 or 16 alphabetic characters; 6, 12, 18 or 24 decimal digits; or 24, 48, 72 or 96 binary digits.

All additions and subtractions are automatically checked by congruence arithmetic on a modulo 3 basis.

Magnetic Core Storage

The primary storage of the UNIVAC III System is a ferrite core storage unit of 8,192 UNIVAC III words. Additional modules of storage can be added to increase this capacity to 16,384; 24,576; or 32,768 UNIVAC III words.

The complete memory cycle including selection, read-out and regeneration of a word is 4 microseconds.

The basic unit of storage in the UNIVAC III Data-Processing System is a fixed-length word consisting of 27 binary bits. Twenty-five information bits represent data, instructions, or control words. A twenty-fifth bit is used to indicate the sign in a data word. The remaining two bits are used to check the accuracy of the transfer of all information to and from magnetic core storage.

UNISERVO III SYNCHRONIZER AND TAPE UNITS

The UNISERVO* III synchronizer serves as a communication device linking the system's core storage to its UNISERVO III tape units. When receiving or transmitting data, the Central Processor is never linked directly with the comparatively slower UNISERVO III tape units, but instead with the high-speed synchronizer.

Once a UNISERVO III input-output instruction is initiated by the Central Processor, the subsequent control of the operation is relegated to the synchronizer. This device automatically carries out the execution of the function specified, releasing the control unit so that the Central Processor continues with the execution of subsequent instructions.

Each UNISERVO III synchronizer has a pair of data channels with separate control circuitry. The result is that UNISERVO III tape reading and tape writing proceed in parallel with one another and with Central Processor computation (and with operations of the general purpose input-output channels which are introduced below). Data entering or leaving magnetic core storage through the high-speed tape channels requires a memory cycle of 4 microseconds per word.

In transfers from core storage, the tape synchronizer receives the 27–bit word and segments the word into three 9–bit groups, called frames.

Trademark of the Sperry Rand Corporation

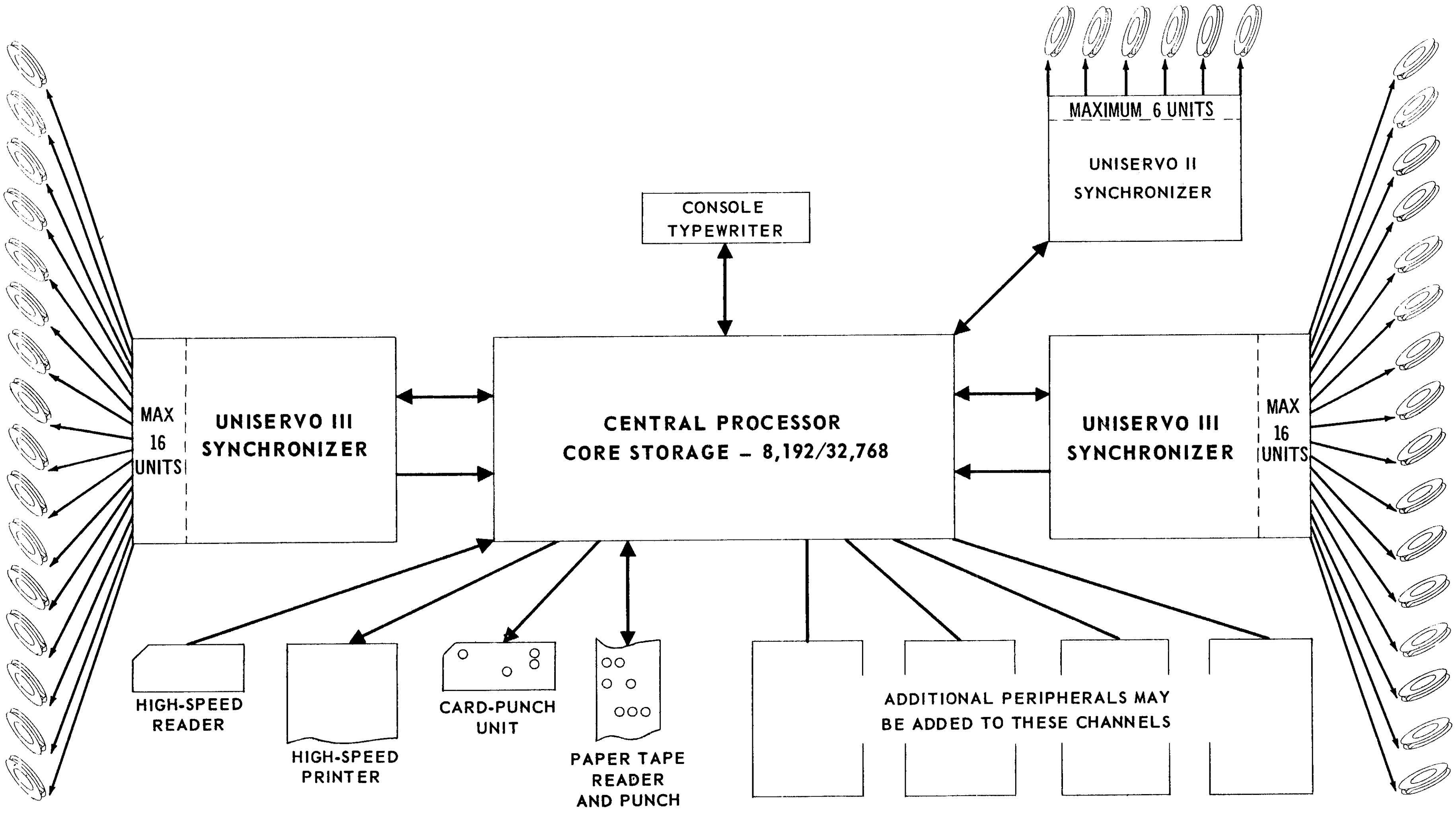

MAXIMUM 6 UNITS
UNISERVO II SYNCHRONIZER
CONSOLE TYPEWRITER
MAX 16 UNITS
UNISERVO III SYNCHRONIZER
CENTRAL PROCESSOR
CORE STORAGE – 8,192/32,768
UNISERVO III SYNCHRONIZER
MAX 16 UNITS
HIGH-SPEED READER
HIGH-SPEED PRINTER
CARD-PUNCH UNIT
PAPER TAPE READER AND PUNCH
ADDITIONAL PERIPHERALS MAY BE ADDED TO THESE CHANNELS

Figure 1–1. Maximum Configuration of the UNIVAC III System

The frames are transferred serially to the read-write head of the specified UNISERVO III tape unit. Each 9–bit frame is then written in parallel channels across the tape. On transfers into core storage the synchronizer essentially reverses its role. Nine-bit frames are sensed at the read-write head, transferred serially to the synchronizer, composed into a 27–bit word, and the entire word transferred to the magnetic core storage.

A single UNISERVO III synchronizer with associated power, control and switching circuitry can control up to 16 UNISERVO III tape units. Two UNISERVO III synchronizers can be attached to a UNIVAC III System, each operating independently of the other.

The pair of data channels on each UNISERVO III synchronizer is normally used to provide simultaneous read and write in parallel with internal computation. As an optional feature, the write channel may be enabled to read as well as write. With this read-read feature installed, the write channel will accept and execute read orders in all respects as if it were a read channel. This feature thus gives the UNIVAC III System, with a single UNISERVO III synchronizer, the ability to accommodate two simultaneous reads in parallel with computation.

The UNISERVO III tape units are the principal means of input and output to the UNIVAC III System and will be the only input-output devices used in the large majority of UNIVAC III processing runs. They employ as their storage medium MYLAR* base, oxide-coated magnetic tape of ½ inch width. The length of magnetic tape on a single reel is 2,400 feet.

As noted above, data is transferred from the synchronizer and recorded across the magnetic tape in 9 information channels. A single 9–position pattern of bits across the width of the tape represents one frame and three consecutive frames constitute a UNIVAC III word in magnetic core storage. The information-packing density on tape is in excess of 1,000 frames per inch, and, during reading or writing, tape speed under the read-write head is maintained at 100 inches per second. These specifications provide an instantaneous transfer rate in excess of 100,000 frames per second, representing over 800,000 binary digits, 200,000 decimal digits or 133,300 alphabetic characters per second.

*MYLAR is a registered trademark of E.I. du Pont de Nemours & Co., Inc.

Data may be grouped on magnetic tape in blocks varying in length, at the programmer's option, in multiples of three frames (one UNIVAC III word). The interblock spacing is approximately 0.7 inch. Assuming 2,000 word blocks, a fully recorded 2,400-foot reel of magnetic tape would contain from 34,000,000 characters (if the data was completely alphabetic) to 51,000,000 digits (if the data was completely in numeric form). A data file equivalent to 515,820 cards (assuming 50% numeric and 50% alpha-numeric data) occupying one full reel of UNISERVO III tape can be read, modified in the Central Processor and reproduced in updated form in less than 5 minutes.

The UNISERVO III tape unit employs a phase modulation recording and sensing technique to achieve high density packing with highest reliability reading. This form of data-recording on magnetic tape enables the UNISERVO III tape unit to discriminate bit patterns accurately at very high packing densities. The skew registers permit the UNISERVO III tape unit to accept, without fault, the normal skew associated with high-speed tape movement.

The detailed functional specifications and control operations for the UNISERVO III tape unit and the UNISERVO III synchronizer will be found in a separate technical bulletin.

GENERAL PURPOSE CHANNELS AND PERIPHERAL INPUT-OUTPUT DEVICES

In addition to the four high-speed data channels associated with the two UNISERVO III synchronizers (and a fifth associated with the UNISERVO II or compatible tape synchronizer), *eight* general purpose channels are attached to the UNIVAC III System. These channels serve as the communication circuits linking the Central Processor's magnetic core memory with the card, paper-tape and printing peripherals. (The term peripherals, as used in these technical bulletins, indicates the group of input-output devices exclusive of UNISERVO tape units.)

The general purpose channels synchronize the operation of any combination of peripherals with the magnetic core storage and provide the same function of parallel operations for the peripherals that the tape synchronizer provides for the UNISERVO tape units. As a result, up to 13 input-output operations (plus unlimited rewinds of

1. UNIVAC III Data-Processing System

The UNIVAC®III System is a medium-cost, high performance electronic data-processing system designed and engineered to provide maximum productivity at minimal cost in a wide variety of business applications. The UNIVAC III System is modular in its major components and flexible in the variety and numbers of peripheral units which can be attached. These components utilize solid-state circuitry of proven reliability.

The high rate of basic internal speed in the UNIVAC III System is enhanced by advanced concepts of systems organization and design logic and it is matched with high-speed input-output units to permit extremely efficient, low-cost-per-unit perductivity in the broadest range of commercial applications.

A UNIVAC III Data-Processing System consists of a Central Processor with magnetic core storage and the arithmetic and control units, magnetic tape units, and varying types and numbers of peripheral devices. An expanded UNIVAC III System is schematically represented in Figure 1–1. The general specifications of these major components are discussed in this section. Detailed functional specifications and analysis of operations are covered in the separate technical bulletins on each component.

FEATURES

- Systems modularity providing the ability for smooth and efficient expansion by the addition of magnetic core storage, magnetic tape units and a full array of punched card, punched paper tape and printing peripherals.
- Sustained magnetic tape to magnetic tape processing with concurrent peripheral operations on-line.
- Up to 13 simultaneous input-output operations paralleling computer processing.
- The fastest magnetic tape system available, providing a tape transfer rate of 133,300 alphabetic and 200,000 numeric characters.
- Fast access, magnetic core storage available in memory sizes of 8,192; 16,384; 24,576; or 32,768 words.
- A 4-microsecond machine cycle providing internal processing speeds usually associated with computers designed for engineering and scientific applications (for example, LOAD, ADD, STORE, BRANCH, and so on, are all accomplished in 8 microseconds).
- A multiple-word operand feature plus field selection which allows the system to take full advantage of word addressable storage and of the high incidence of short fields in data-processing applications with no offsetting disadvantages.
- Bit-handling facilities which enable the UNIVAC III to be programmed to perform many types of special manipulations and allowing the system to utilize a variety of binary input-output codes.
- A powerful programming logic based on a comprehensive single-address instruction repertoire and including automatic index register modification, multiple word operands, field selection, indirect addressing, and scatter-read–gather-write tape operations.
- A completely integrated software package containing an executive routine capable of controlling concurrent peripheral operations on-line, a COBOL compiler, an advanced symbolic assembly system incorporating macro-instructions and an extensive library of common routines, and a sort/merge generator as well as the usual complement of service and diagnostic routines.

®*Registered trademark of the Sperry Rand Corporation*

UNISERVO tape units) could occur in parallel with one another and simultaneously with Central Processor operations.

High-Speed Reader

Both 80–column or 90–column card readers are available with the UNIVAC III System. Any number of card readers may be under simultaneous control of a single system up to the number of available general purpose channels.

Data is read into the system from punched cards at the maximum rate of 700 cards per minute. The data may be represented internally in either card code (a binary one per hole in the equivalent punch position) or in machine code (as the result of an automatic translation during the read-in of data).

The card transport system of the High-Speed Reader is unclutched and consists of: a 2,000–card input magazine; a read station for transfer of data to memory; a separate read station for check reading, providing automatic verification of sensing; and three program-selectable 1,000–card-capacity stackers.

Program controlled functions include:

- Feed Card
- Translate Image
- Select Stacker
- Select Memory Address
- Interrupt Program

Misfeeds, row misregistrations, card jams, full stackers and empty magazine are detected and indicated by signal to the program and to the operator.

Card-Punch Unit

Both 80–column or 90–column punch units are available with the UNIVAC III System and multiple punches may be operated simultaneously under the control of a single UNIVAC III System up to the number of available general purpose channels.

Data from magnetic core storage is punched into cards at the maximum rate of 300 cards per minute. As with the card reader, data may be transferred in either card code or machine code.

Under program control, cards move in a succession of 4 card cycles along a path composed of a 1,000–card input magazine; a clutched first wait station; a clutched second wait station; a clutched punch station; and a check-read station which provides automatic verification of card-punching. At the check-read station the card enters continuously driven eject rollers to be delivered to one of two program-selectable, 1,000–card-capacity stackers.

Program controlled functions include:

- Feed Card
- Move Card from Station to Station
- Translate Image
- Punch
- Select Stacker
- Interrupt Program

An empty input magazine, card jam, full stacker and full chip-box are detected and signalled to the program and to the operator.

High-Speed Printer

The High-Speed Printer of the UNIVAC III System has a line printing rate from a minimum of 700 lines per minute with alpha-numeric information and up to 922 lines per minute with completely numeric printing. Multiple High-Speed Printers may be operated simultaneously under the control of a single UNIVAC III System up to the number of available general purpose channels.

The printing span of a single line of print is 128 characters. Any of the 128 print positions can contain any of the 26 alphabetic characters, the ten digits 0 through 9, or one of 15 special symbols, as follows:

,	comma	/	solidus
.	period	'	apostrophe
=	equals sign	*	asterisk
<	less than	>	more than
;	semicolon	$	dollar sign
–	minus or hyphen	(	open parenthesis
+	plus	)	close parenthesis
:	colon		

The internally stored program specifies the 32 consecutive words of memory which will compose the print line. To satisfy the requirements of the particular format, each of the 128 consecutive print positions may contain printing characters to produce a solid line of type, or the positions may be subdivided into words or fields of various lengths. This completely variable format is under the control of an editing program.

The printed characters are spaced 10 per inch horizontally. Vertical spacing of 6 or 8 lines per inch may be selected by the operator. Skipping or advancing of paper proceeds at the rate of 22 inches per second.

The paper-feed mechanism accommodates continuous form, sprocket-fed paper ranging up to card stock in weight. The form may be either blank or preprinted, varying in over-all width from 4 to 22 inches.

Up to five carbon copies of the printing can be produced with paper between 11 and 13.5 pounds in weight. Further, impression control permits variation in the strength of the print-hammer stroke. Fine vertical adjustments of the paper position may be made while the printer is in operation.

No paper and paper runaway are detected and signalled to the operator.

The detailed functional specifications and the control of the operation for the peripheral input-output devices will be found in separate technical bulletins on each device.

SYSTEMS ORGANIZATION

It has long been a design objective of computer engineers to provide an EDP system which is able to co-ordinate and control all of the elements of data-processing and data conversion from a single set of electronic circuitry. Such a system would relieve the user of the expensive support of special purpose auxiliary equipment and provide him with a maximum processing power relative to his investment in electronic circuits.

The design of such a system is predicated upon:

- The existence of electronic components of sufficient reliability to insure against total systems failure.
- An input-output logic sufficiently flexible to permit a variety of input-output devices to operate in parallel with one another and with the Central Processor.
- The attainment of internal operating speeds considerably out of balance with top speeds obtainable from card, printing and paper tape peripherals.
- A transference from engineering to programming of the responsibility for systems control. Reducing the cost of computer development, and allowing for maximum flexibility through the creation of sophisticated and efficient control routines.

The UNIVAC III System, while basically a tape-to-tape system, provides for concurrent peripheral operations to proceed on-line through:

- The utilization of reliable solid-state equipment, proven in use on the UNIVAC Solid-State and and the UNIVAC LARC* Systems.
- The provision of eight fully-buffered general purpose channels (in addition to the five high-speed tape channels) and the automatic program interrupt feature.
- The seven-fold increase in internal operating speeds contrasted to the 1.1 to 2.8 increase obtainable within electromechanical limitations with peripheral equipment.
- The development of an executive routine, CHIEF, which controls error conditions, provides for input-output control, and allows itself to be modified to meet the specific requirements of an operating installation.

The UNIVAC III System from its inception was planned and designed to permit peripheral operations, which, while functionally "out of (the tape-to-tape processing) line," would proceed through peripherals controlled "in-line" through the Central Processor and concurrently with the tape-to-tape processing.

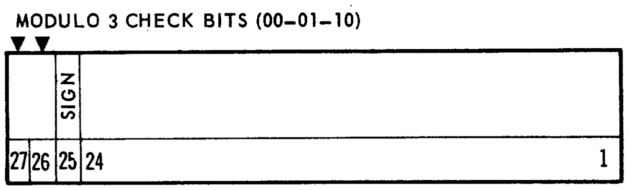
A simple application of the concept of concurrent peripheral operations on-line would require that a payroll run not use the printer for paychecks directly, but rather produce edited output data on

** Trademark of the Sperry Rand Corporation* magnetic tapes. This tape data would, in turn, be printed concurrently with a subsequent run. This approach has the added advantage that processing speed will not be limited to the speed of the printer. The magnetic tape will be used as a buffer between the high internal speeds and the slower printer speeds.

It should be noted that, when the edited payroll tape is printed, concurrently with a subsequent tape-to-tape run, during a half-hour of operation over 21,000 lines could be printed; however, high-speed storage would be required for a total of 45 seconds during the half-hour and the read channel of the UNISERVO III synchronizer would be required for a total of 28 seconds.

2. UNIVAC III Word

The UNIVAC III word is the basic unit of storage in the system. It is fixed in length and consists of 27 binary digits. Twenty-four bits are used to represent data, and a twenty-fifth bit denotes the sign. The remaining two bits are modulo 3 check bits required to produce a modulo 3 sum of zero for the 27 bits. They are used to automatically check the accuracy of word transfers and, by congruence arithmetic, to automatically check all addition and subtraction operations.



DATA WORD FORMATS

Data may be represented in any of the three formats shown in Figure 2–2, or in any combination. The processing circuits do not distinguish between data formats. This distinction is completely a function of the program.

Six decimal digits plus sign may be represented in a word. Each digit is expressed in excess-three binary coded decimal format. All decimal arithmetic operations assume the values to be in this format.

Four alphabetic or special characters may be represented in alpha-numeric data word format. Each character is composed of six bits, two bits for the zone (00 to 11) and four bits for the numeric portion (0000 to 1111); sixty-four different characters may therefore be represented.

See Figure 2–1 for the UNIVAC III Character Code.

Values may be expressed in pure binary with values up to $2^{24}-1$. All binary arithmetic operations assume the values to be in this format.

		ZONE			
		00	01	10	11
NUMERIC	0000	Δ	+		
	0001	;	)	*	(
	0010	–	.	$	,
	0011	0			"
	0100	1	A	J	/
	0101	2	B	K	S
	0110	3	C	L	T
	0111	4	D	M	U
	1000	5	E	N	V
	1001	6	F	O	W
	1010	7	G	P	X
	1011	8	H	Q	Y
	1100	9	I	R	Z
	1101	:	=		
	1110	<			
	1111	>			

Figure 2–1. UNIVAC III Character Code

DECIMAL WORD*

Six 4–bit numeric digits along with sign constitute a decimal word.

SIGN	DIGIT 6	DIGIT 5	DIGIT 4	DIGIT 3	DIGIT 2	DIGIT 1
25	24 21	20 17	16 13	12 9	8 5	4 1

S–Bit 25 indicates the sign, 1 for minus and 0 for plus.
Digits–6, 5, 4, 3, 2, 1–Each digit is expressed in excess-three code. See Figure 2–1.

ALPHA-NUMERIC WORD*

Four 6–bit alpha-numeric characters constitute an alpha-numeric word.

SIGN	CHARACTER 4	CHARACTER 3	CHARACTER 2	CHARACTER 1
25	24 19	18 13	12 7	6 1

S–Sign.
Characters–4, 3, 2, 1–Each character is represented by 6 bits.

BINARY WORD*

The entire 24–bit data portion of any memory location can be used to represent a binary value ranging from 0 through plus or minus 16,777,215.

SIGN	24–BIT BINARY VALUE
25	24 1

S–Bit indicates the sign, 1 for minus and 0 for plus.

Two check-bit positions are omitted for illustrative purposes.

Figure 2–2. Data Word Formats

	25	24 21	20 15	14 11	10 1
GENERAL INSTRUCTION FORMAT	I/A F/S	X	OP CODE	AR	m OPERAND ADDRESS
SHIFT INSTRUCTIONS	I/A	X	OP CODE	AR	SHIFT COUNT/m
INDEX REGISTER INSTRUCTION	I/A	X	OP CODE	XO	m OPERAND ADDRESS
INDICATOR INSTRUCTIONS	I/A	X	OP CODE	INDICATOR, CLASS, OR CHANNEL	INDICATOR/m
INITIATE I/O INSTRUCTION	I/A	X	OP CODE	CHANNEL	ADDRESS OF I/O FUNCTION SPECIFICATION

Figure 2–3. Instruction Word Formats

INSTRUCTION WORD FORMATS

UNIVAC III Central Processor Instructions are in five basic formats. In each format the functional grouping of bits is the same. Some bit groups perform the identical function regardless of the operation to be performed, while the functions of other groups vary, depending on the operation to be performed (Figure 2–3).

BIT POSITION 25

Indirect Addressing or Field Selection Option Designation. Indirect Addressing provides the ability to express an operand location, indirectly, through an intermediate control word. Nearly all instructions of the UNIVAC III repertoire are capable of utilizing this feature. In this form, the address in the basic instruction does not refer directly to the operand to be accessed but rather to a control word, which in turn contains the operand address. The word containing the operand address is termed the Indirect Address Control Word (INAD).

Field Selection provides the ability for an instruction to operate directly upon data fields that are not multiples of a word. This feature is available for processing instructions in which bit positions 1–10 would normally designate an operand address. When field selection is desired, bit positions 1–10 specify the location of a Field Select Control Word (FSEL). The FSEL provides the definition of the field size and specifies the address of the operand.

Either option is expressed by the presence of a 1–bit. The specific choice is determined by the format of the control word.

BIT POSITIONS 21–24

Binary Address (0001–1111) of the Index Register (X) Selected. The contents of the specified index register are used to increment bit positions 1–10 of the instruction. The m-address bits of all instructions, regardless of type, are automatically indexed while being staticized in the control unit – bits 1–10 + (X) produce m'. If 0000 is specified, $m = m'$. Neither the contents of the index register specified nor the instruction in memory is altered by the indexing.

BIT POSITIONS 15–20

Operation Code.

BIT POSITIONS 11–14

Depending on the operation to be performed the function of this group varies. The function of this group depends on the type of instruction. It will be the designation of the arithmetic register(s) selected, the binary address of the index register to be operated on, the indicator or group of indicators to be tested, or the selected input-output channel.

BIT POSITIONS 1–10

This bit group is always indexed (if only by 0's) and becomes a 15–bit group called m'. This is done in the Memory Address Adder during the instruction set-up cycle.

The function of m' varies with the operation performed as reflected in the above formats.

However, if position 25 is a 1–bit, positions 1–10 reflect the unindexed address of either an Indirect Address Control Word or a Field Select Control Word. The original function of positions 1–10 of the basic instruction will in these cases be relegated to the control words.

3. Control Unit

The functions of the control registers, a schematic of their relationship, and the control cycle of the UNIVAC III Processor are given in this section.

CONTROL COUNTER (CC)

This register is used to locate the next instruction to be accessed from memory for execution. On the last memory cycle of an instruction, the 15–bit value of the CC (the address of the instruction currently in progress) is incremented by 1 or 2 in the Memory Address Adder and returned to the Control Counter. The new value is also transferred to the Memory Switch Register in order to address memory for read-out of the instruction in the next memory cycle.

$$\begin{tikzcd} (CC) + 1 \text{ or } 2 \arrow[r] \arrow[dr] & MSR \\ & CC \end{tikzcd}$$

INDEX REGISTER (X)

These registers are used to develop the final operand address. When the instruction is read from memory into the Central Processor Register, the 10–bit m address (or 15–bit if it is a control word) is added to the contents of the selected index register. This addition is accomplished in the Memory Address Adder. The sum is then used by the Memory Switch Register to locate the operand to be accessed from memory in the next memory cycle. The modified storage address is also delivered to the Memory Address Register. Indexing occurs *during* the cycle in which the instruction was read from memory. The contents of the index register are not affected by the indexing.

$$\begin{tikzcd} m + (X) \arrow[r] \arrow[dr] & MSR \\ & MAR \end{tikzcd}$$

MEMORY SWITCH REGISTER (MSR)

This register contains the result of all additions of the Memory Address Adder. The Memory Switch Register addresses the magnetic core storage for read-in or read-out of all data, control words, and instructions.

MEMORY ADDRESS REGISTER (MAR)

This register contains the 15–bit result of m + (X). It will only be utilized if the instruction specifies a multi-word operand. In the event of a reference to a multi-word operand, the contents of the MAR will be decremented in the Memory Address Adder with the result used to address the next word of the operand to be read from memory. The result of (MAR) – 1 is also returned to the MAR.

$$\begin{tikzcd} (MAR) - 1 \arrow[r] \arrow[dr] & MSR \\ & MAR \end{tikzcd}$$

MEMORY ADDRESS COUNTERS (MAC)

These counters, one for each of the thirteen input-output channels, contain the 15–bit address of the last word of input-output data transferred to or from memory through the synchronizer circuitry of the related channel. When any channel is granted a memory access, the contents of its related MAC are read out and incremented through the Memory Address Adder. The result will then be used to access memory for read-in or read-out in the next memory cycle.

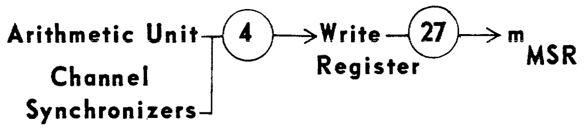
CENTRAL PROCESSOR REGISTER (CPR)

Operands, instructions and their associated control words, when accessed, are read from memory directly into the CPR register. If an instruction is read, the OP Code, the AR portion, and the X portion are read out and stored in decoders, in order to alert the designated AR and X and to build up function table signals for the execution of the instruction. The m address is added to the contents of the selected index register to produce the effective operand address. During multiplication or division it has the special requirement of retaining the multiplicand or divisor.

Input-output data and input-output function specifications do not utilize this register.

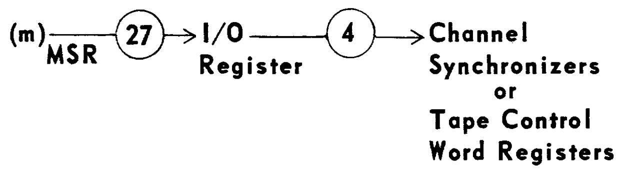
WRITE REGISTER

All data transferred to memory is routed through the Write Register. Its function is to accept information from a 4–bit parallel transmission line and to transfer it to the memory location specified by the MSR over a 27-bit parallel line.



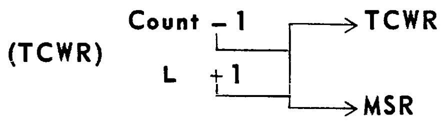
INPUT-OUTPUT REGISTER

When read from memory, all output, including tape control words and input-output function specifications, pass through this register. Its function is to convert the 27–bit parallel transmission from memory to a 4–bit parallel transmission to the channel synchronizers.



TAPE CONTROL WORD REGISTERS (TCWR)

The four TCWR's (one for each UNISERVO III channel) are used in conjunction with the scatter-reading and gather-writing features. When memory access is granted to any of the four channels (and control words for scatter-read or gather-write are being used), the contents of the appropriate TCWR are transferred through the Memory Address Adder where the word-count portion is decremented by one and the address portion is incremented by 1. The new address is then used to access memory for the read-in or read-out of the input-output data in the next memory cycle. The adjusted control word is also returned to the TCWR. When control words are used for tape reading or writing, the Memory Address Counters for the UNISERVO III Read and Write Channels are used to access the next control word when required. If control words are not used, the UNISERVO III Memory Address Counters are used to access memory for input or output data.



MEMORY PRIORITY CIRCUITS (MPC)

The MPC circuits govern access to the magnetic core storage by controlling the selection of the contents of the CC, the MAR, an MAC, or a TCWR to be transferred to the MSR through the Memory Address Adder.

The selection is based, in the case of the MAC and TCWR, on the transfer speed of the related peripheral unit. As each peripheral unit's synchronizer circuitry determines a memory access requirement, a request is sent to the MPC. At every 4–microsecond memory cycle all memory requests are evaluated and the channel with the highest priority will be selected. The contents of the MAC for the selected channel will be sent to the Memory Address Adder and memory read-in or read-out performed according to the new setting of the MSR. The request is then eliminated from the MPC.

This action will be repeated as long as any channel synchronizer requests memory access. At the time when all requests from the channel synchronizer have been accommodated, either the Control Counter or the Memory Address Register will be given access to memory.

The general order of priority for memory access is as follows:

UNISERVO III Channel Synchronizer

UNISERVO II Channel Synchronizer

General Purpose Channels

Accessing Multi-Word Operands

Accessing Instructions

UNIVAC III PROCESSOR BLOCK DIAGRAM

The functional relationship of the elements of the control unit are schematically represented by the UNIVAC III Processor Block Diagram, Figure 3–1, on page 3–3.

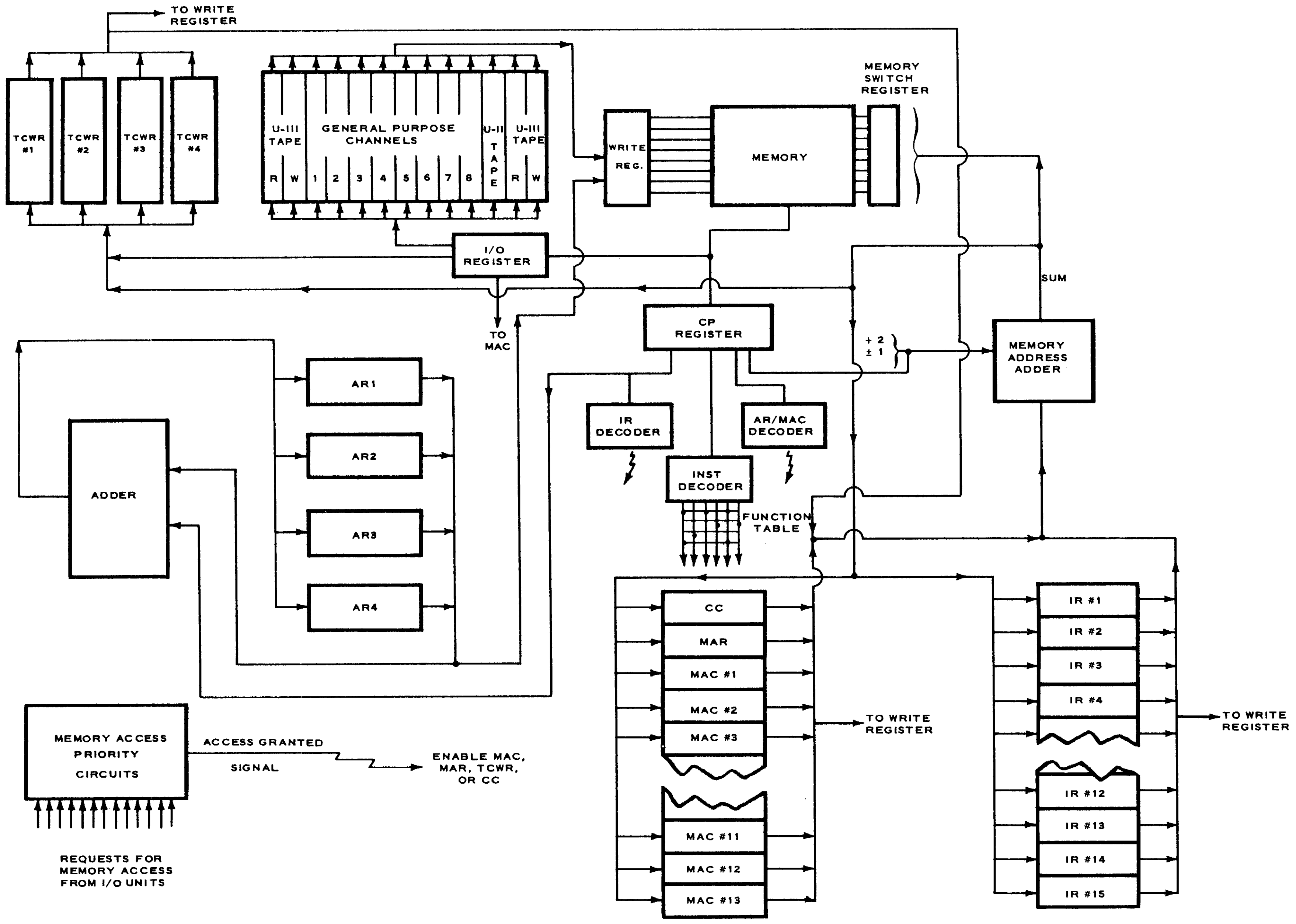


Figure 3-1. UNIVAC III Processor Block Diagram

THE CONTROL CYCLE

The major function of the control unit is to sequentially select each instruction from memory, interpret it, and perform all of the operations necessary for its execution.

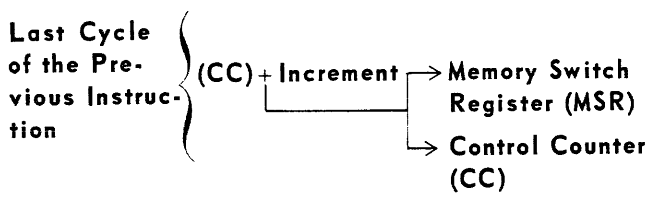
The sequencing of instructions is a function of the Control Counter (CC). The CC contains the memory address of the instruction being executed in a 15-bit binary format.

The control unit sequence is divided into 4-microsecond memory cycles. The description of the control cycle will be in terms of these cycles rather than in microseconds.

Single-Word Operand

During the final Execution Cycle of the preceding instruction, the 15-bit address currently contained in the CC Register is transferred to the Memory Address Adder. The other input to the adder, the increment amount, is specified as a function of the nature of the previous instruction. Most instructions generate an increment of 1 and step the program to the next sequential location. General branching operations may replace the CC reading with a new address rather than increment the current address. Special test operations cause the CC to be incremented by either 1 or 2, depending on the set of the conditions tested.

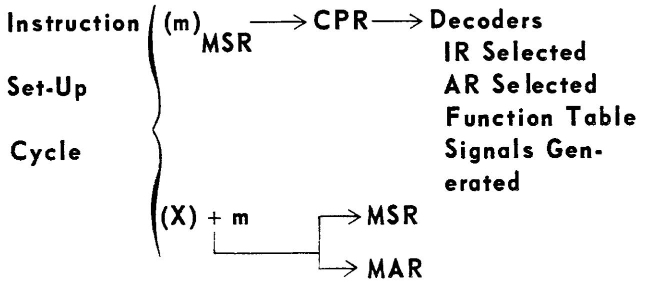
The address fabricated by the Memory Address Adder is sent to the Memory Switch Register (MSR) and returned to the CC Register replacing its previous contents.


Last Cycle of the Previous Instruction
(CC) + Increment
Memory Switch Register (MSR)
Control Counter (CC)

Instruction Set-Up Cycle

During the Instruction Set-Up Cycle, the 27 bits at the storage location selected by the Memory Switch Register are sent to the Central Processor Register (CPR) where they are staticized. During the initial part of this cycle, the instruction being received from memory is decoded through the Index Register, the Arithmetic Register, and Instruction Decoders. The appropriate index register and AR are selected and function table signals are generated which will affect the execution of the instruction.

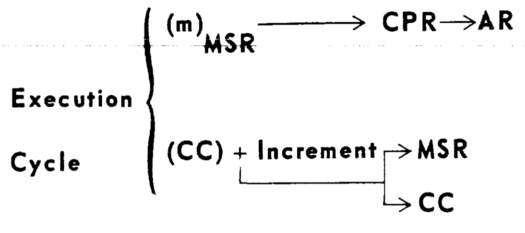
During the latter part of the Instruction Set-Up Cycle, the contents of the index register specified by the instruction, and the memory address (from the CPR) are combined in the Memory Address Adder, and the result is sent to the Memory Switch Register and the Memory Address Register. The MSR, which now contains the full 15-bit address of the operand, is used to address memory.



Execution Cycle

During the Execution Cycle, the contents of the Memory Switch Register select the memory location which contains the data to be used in the operation. This data will be routed through the Central Processor Register to the specified AR(s) which have been alerted by the decoding of the AR portion of the instruction on the previous cycle.

During this Execution Cycle, the contents of the CC are being read out and are being adjusted by a selected increment. Thus, there is a continuous overlap between the Execution Cycle of the previous instruction and the fabrication of the location of the next instruction.



Multi-Word Operand

The incrementing of the CC during the execution of an operation employing a multi-word operand is delayed until the final Execution Cycle of the operation. The control unit is required during all other Execution Cycles to decrement the contents of the Memory Address Counter to select in turn the other words of the operand.

Last Cycle of Previous Instruction

$$\begin{tikzcd} (CC) + \text{Increment} \arrow[r] \arrow[dr] & MSR \\ & CC \end{tikzcd}$$

Instruction Set-Up Cycle

$$\begin{tikzcd} (m)_{MSR} \arrow[r] & CPR \arrow[r] & \begin{array}{l}\text{Decoders}\\ \text{IR Selected}\\ \text{AR Selected}\\ \text{FT Signals Generated}\end{array} \\ (X) + m_{CPR} \arrow[r] \arrow[dr] & MSR & \\ & MAR & \end{tikzcd}$$

Execution Cycle (first word)

$$\begin{tikzcd} (m)_{MSR} \arrow[r] & CPR \arrow[r] & AR \\ (MAR) - 1 \arrow[r] \arrow[dr] & MSR & \\ & MAR & \end{tikzcd}$$

Execution Cycle (last word)

$$\begin{tikzcd} (m)_{MSR} \arrow[r] & CPR \arrow[r] & AR \\ (CC) + \text{Increment} \arrow[r] \arrow[dr] & MSR & \\ & CC & \end{tikzcd}$$

4. UNIVAC III Command Repertoire

PROGRAMMING FEATURES

The UNIVAC III System provides a number of programming features greatly expanding the power of its basic command repertoire and providing additional flexibility to the systems designer as well as to the programmer.

Index Registers

In the UNIVAC III System nine or fifteen index registers make possible address modification, program loop control, and the setting of counters without additional time being spent on the execution of an instruction. This occurs as all instructions (and control words) go through an indexing phase in order to develop the final operand address. The net result of this feature is an expansion of the memory.

Index registers may be used effectively to reduce the number of instructions required for any application. Their basic function is to permit the modification of referenced data locations. They do this by changing the "effective" address sought, without altering the "base" address itself. Therefore, the entire processing routine remains unaltered in memory available for application to any set of data.

Modifying the base operand address of any instruction without reference to the arithmetic registers has also eliminated the need to handle each variable individually.

Each index register contains a 15–bit unsigned binary value and is specified in binary (0001-1111) in bits 21–24 of the instruction word.

During the access of each instruction from memory, bit positions 1–10 of the instruction and the contents of the specified index register are automatically added in binary [m + (X)]. A 15–bit effective operand address, m′, is produced. Address modification in the UNIVAC III System does not require an additional cycle. Any carry beyond bit 15 is ignored. The instruction in memory and the index register addressed are not affected as a result of the indexing.

If 0000 is specified in bit positions 21–24 of the instruction, no effective indexing occurs.

Multi-Word Operands

The UNIVAC III System contains four one-word arithmetic registers – AR1, AR2, AR4, and AR8. The arithmetic register involved in the execution of the instruction is designated by a 1–bit in bit positions 11–14 of the instruction word as shown below:

Bit Positions

14	13	12	11	
1	0	0	0	AR8
0	1	0	0	AR4
0	0	1	0	AR2
0	0	0	1	AR1

Through any combination of these bit designations it is possible to manipulate operands of from one to four words with a single instruction. The number and position of 1–bits control the size of the operand and its placement within the arithmetic registers. AR's not specified will not be affected by the instruction execution (Figure 4-1).

The AR's selected may be adjacent or non-adjacent and in either case they will act as a single extended register. Multi-word operands in memory, however, must be from adjacent locations.

The contents of the memory location specified in the instruction (m′) are considered the least significant word of the operand and are used in conjunction with the lowest numbered AR designated. The balance of the operand in the lower ordered memory location(s) are related to the higher numbered designated AR's.

The sign of the least significant word of a multi-word operand is treated as the sign of the entire operand regardless of the sign of the more significant words. After arithmetic operations the correct algebraic sign will be placed in all AR's involved, regardless of their previous signs.

A carry from the least significant AR is propagated to the next higher numbered register designated in the instruction. Only a carry beyond the most significant AR designated causes the Arithmetic Overflow Indicator to be set and a Contingency Interrupt to occur.

Generally, when a multi-word operand is specified an additional machine cycle for each word beyond one should be added to the basic execution time.

Indirect Addressing

In some programming instances, it is valuable to be able to specify the location where the address of an operand is stored rather than to specify the location of the operand directly. This method of addressing an operand is called *indirect addressing*. It is of use in writing compilers, sort and merge routines, manipulating subroutines, and in the formation of various control words for the UNIVAC III System. Indirect addressing has therefore proven valuable in reducing programmer effort, processing time and instruction storage area.

Indirect addressing is specified by placement of a 1–bit in bit position 25 of the instruction word. The indexed address of the instruction word in this case will not be the location of the operand, but rather the location of an Indirect Address Control Word (INAD). The indexed address of the INAD will specify the location of the data.

I/A	X	000	Unass'g.	L-Addr.
25	24 21	20 18	17 16	15 1

I/A	*Indirect address/field selection option*
X	*Binary address of index register, 1 to 15*
Bits 18–20	*Must be 0's*
Bits 16–17	*Unassigned*
L-Address	*If I/A is a 1–bit, the L-address specifies the unindexed location of another INAD or a Field Select Control Word (FSEL).*
	If I/A is a 0–bit, the L-address specifies the unindexed address of the data.

If it is desired to delay the expression of the operand address through another level, a 1–bit should be placed in bit position 25 of the first level INAD and its indexed L-address made the location of the second INAD. In this way, indirect addressing can be made to extend through several

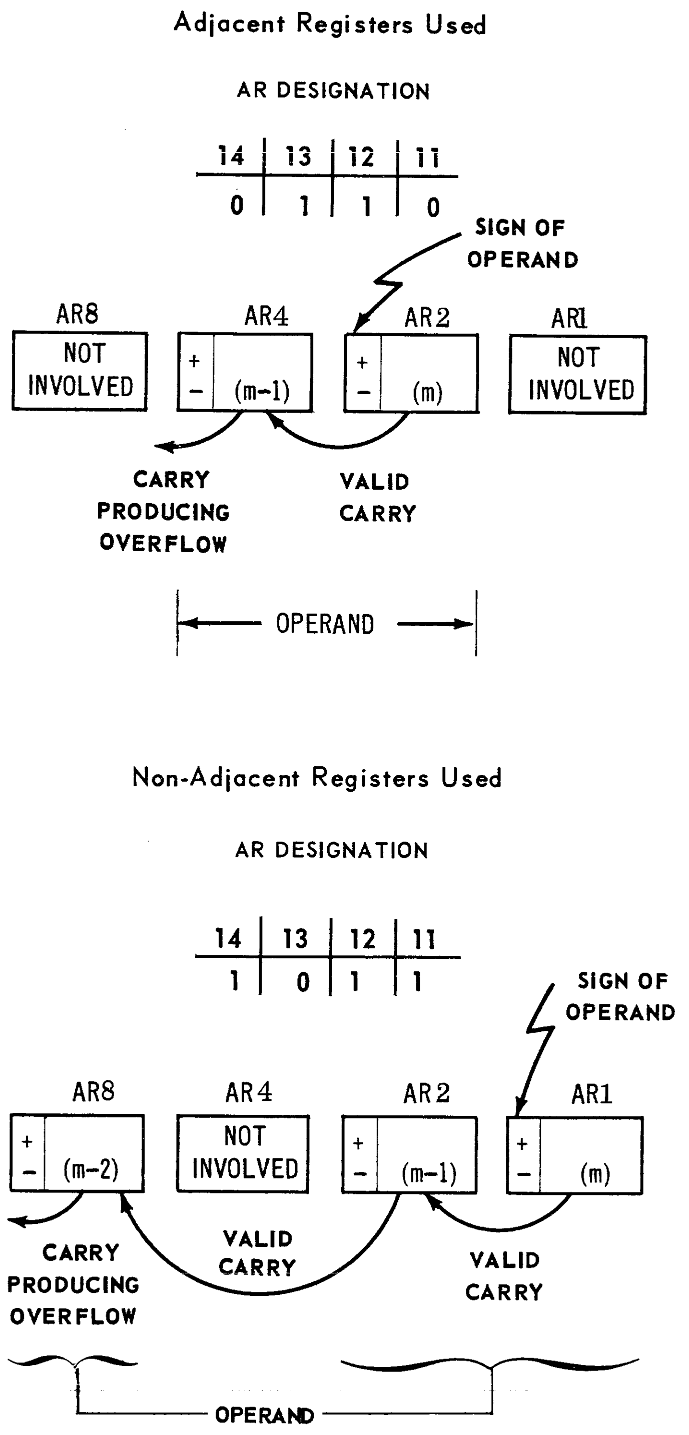


Figure 4-1. Examples of Multi-Word Operands

levels until an INAD with a 0–bit in bit position 25 is accessed. The original instruction will then be executed, using the operand address of the last INAD. There is no arbitrary limit to the possible levels of "cascading."

Indirect addressing is not restricted to referencing data.

Instructions utilizing indirect addressing are executed in the following manner:

a. The basic instruction word is set-up in the Instruction Register, an indexed address developed m + (X) and bit 25 is examined.

b. If bit 25 is a 1–bit, execution of the instruction is delayed and the contents of the indexed address are accessed. Again an indexed location is developed L + (X) and bit position 25 is again examined.

   (If bit position 25 is a 1–bit, Step b is repeated until the word accessed contains a 0–bit.)

c. If bit position 25 is a 0–bit, the control word is further examined. If bit positions 18–20 contain binary 0's the developed L-address is the address of the data.* The instruction is then executed.

Though the Control Counter is not altered, indirect addressing will require an additional memory cycle for each INAD accessed.

Illustration

Load the contents of DATA (0651) into Arithmetic Register 4 using the indirect address option. The operand address is stored in the 15 least significant bits of the Indirect Address Control Word located at 0700 and tagged CONTROL.

LA 4, * **CONTROL,**

I/A	X	OP Code	AR	m
1	0000	12	0100	0700

**A 1–bit in position 25 may also indicate field selection; however, field selection is specified by the presence of bits other than 0–bits in positions 16–20 of the control word (FSEL).*

(0700) CONTROL + **DATA**

I/A	X			L
0	0000	000	00	0651

(0651) DATA

25	24 21	20 17	16 13	12 9	8 5	4 1

Field Selection

When a data field is not a multiple of a word, field selection should be employed in order to isolate only those bits, digits or characters to be operated on during the instruction execution. The position of the field to be selected is defined in a Field Select Control Word (FSEL) as is the field's address.

The indexed m address of the basic instruction word is made the location of the FSEL and bit 25 records a 1–bit. The FSEL has the following format:

	X	Left Boundary Bit	Right Boundary Bit	m
25	24 21	20 16	15 11	10 1

Bit 25	*Always 0*
X	*Binary address of index register 0–15*
Left Boundary Bit	*Most significant bit position of field to be selected. The bit position is specified in excess-three and ranges from 4 (LSB of word) to 27 (MSB of word).* *If a multi-word operand is specified in the instruction, the Left Boundary Bit Designator must be within the most significant word of the operand.*

Right Boundary Bit	*Least significant bit position of the field to be selected. The bit position is specified in excess-three and ranges from 4 (LSB of word) to 27 (MSB of word). If a multi-word operand is specified in the instruction, the Right Boundary Bit must be within the least significant word of the operand.*
m	*Unindexed address of the word containing the least significant bit of the field*

Notes

1. The sign bit(s) will not be selected; the signs of all fields selected will be positive.
2. Portions of the word(s) beyond the boundaries specified are binary 0's. If decimal add or decimal subtract is specified, these binary 0's are treated as excess-three 0's.
3. Field Selection from memory affects or acts in conjunction with the same relative bit positions of the arithmetic register(s) unless a carry results beyond the most significant bit or digit within the register. Such carries may be propagated up to the limits of the most significant arithmetic register designated. Beyond this limit overflow will occur.
4. When a multi-word operand is specified in the basic instruction the arithmetic registers may be non-adjacent but the bits of the operand from memory must be contiguous.
5. The FSEL may be indirectly addressed. But indirect addressing may not extend beyond the field select cycle. Hence bit position 25 of a FSEL must be 0.
6. One machine cycle is required to access and analyze the FSEL. The Control Counter is not affected by this accessing.

Illustration

Arithmetic Register 1 contains a value of 770111. Add to it the three least significant digits of the value 99933 in LOC B (0739). The FSEL is located in CONTROL (0266).

INSTRUCTION

DA 1, * CONTROL

I/A	X	OP Code	AR	m
1	0000	20	0001	0266

CONTROL (0266)

+ 12, 1, LOC B,

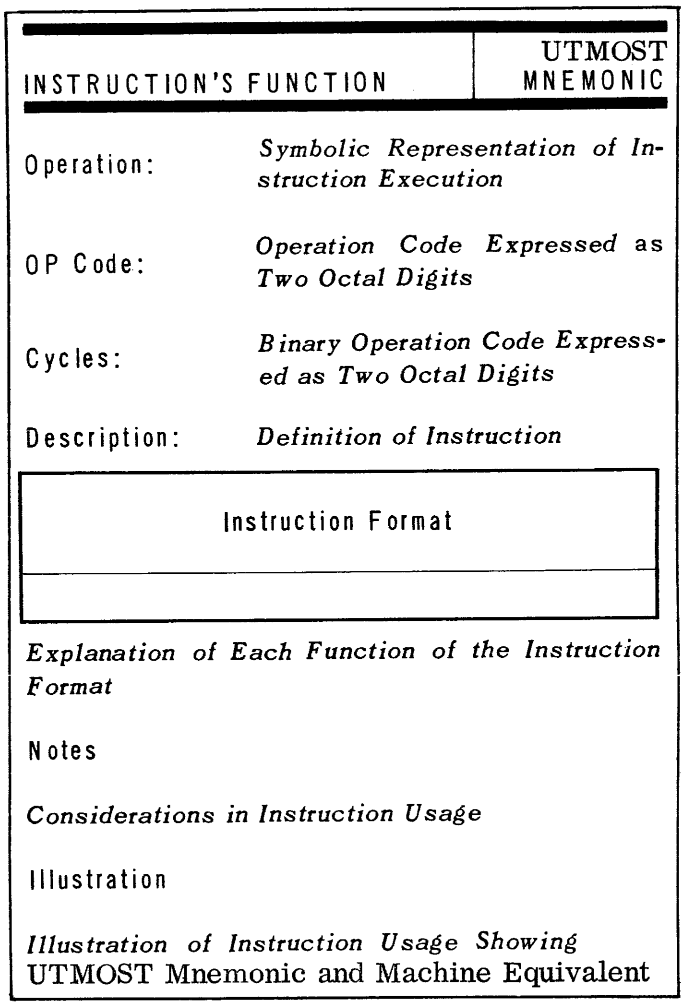
I/A	X	Left Boundary Bit	Right Boundary Bit	m
0	0	15	4	0739

RESULT IN AR1 = 770444

INSTRUCTION FORMAT

The purpose of this section is to provide the reader with a complete summary of the UNIVAC III Central Processor command repertoire as well as a knowledge of the subtle considerations applicable to each instruction.

Each instruction description contains a symbolic representation of the operation as well as its format (Figure 4–2). This format is further elaborated upon by the use of an example illustrating the operation described. Each example is illustrated in two ways. One illustration will be in the equivalent of machine representation. That is, the coded instruction will contain the machine binary equivalent when applicable, or its decimal equivalent, in various segments of the instruction word. (For example, the index registers will be designated by a 4–bit binary configuration ranging from 0000-1111.) The same illustration will be coded in UTMOST (UNIVAC Three Machine Oriented Symbolic Translator).



INSTRUCTION'S FUNCTION	UTMOST MNEMONIC

Operation:	*Symbolic Representation of Instruction Execution*
OP Code:	*Operation Code Expressed as Two Octal Digits*
Cycles:	*Binary Operation Code Expressed as Two Octal Digits*
Description:	*Definition of Instruction*

Instruction Format

Explanation of Each Function of the Instruction Format

Notes

Considerations in Instruction Usage

Illustration

Illustration of Instruction Usage Showing UTMOST Mnemonic and Machine Equivalent

Figure 4-2. Instruction Layout

SYMBOLOGY AND ABBREVIATIONS USED

( )	*The contents of*
( )x	*The contents of, as specified by x*
$a \longrightarrow b$	*a is transferred to b*
m	*A 10–bit unindexed address*
m'	*A 15–bit indexed address*
ARi	*One of the four arithmetic registers*
X_i	*One of the fifteen index registers used to modify m*
XO_i	*One of the fifteen index registers to be affected*
CC	*The Control Counter*
L	*A 15–bit unindexed address*
L'	*A 15–bit indexed address*
MAC_i	*One of thirteen Memory Address Counters*
MAR	*Memory Address Register*
SL_i	*One of thirteen stand-by locations*
TBR	*Typewriter Buffer Register*
$TCWR_i$	*One of four Tape Control Word Registers*
ICW	*Index Register Modification Control Word*

OPERAND TRANSFER INSTRUCTIONS

These instructions transfer operands from memory to the arithmetic registers or from the arithmetic registers to memory.

Reading from memory or the arithmetic registers does not alter their contents. Reading into memory locations or the arithmetic registers will replace the original contents with the operand read in.

LOAD	LA

Operation:	$(m') \longrightarrow AR_i$
OP Code:	*12*
Cycles:	*2*

Description: *Transfer an operand from the indexed memory location(s) to the arithmetic register(s) designated.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Arithmetic register(s) are first automatically cleared to binary 0's.*
2. *Contents of memory location(s) accessed are not altered.*
3. *Indirect addressing, field selection and multi-word operand(s) may be employed.*

Illustration

Transfer the operand, FIELD (0689), to AR2.

LA 2, FIELD A,

I/A	X	Op Code	AR	m
0	0000	12	0010	0689

LOAD A NEGATIVELY — LAN

Operation:	*(m)⟶ARi*
OP Code:	*13*
Cycles:	*2*

Description: *Transfer an operand from the indexed memory location(s) to the arithmetic register(s) designated, reversing* each of the signs.

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Arithmetic register(s) are first automatically cleared to binary 0's.*
2. *Contents and sign of memory location(s) accessed are not altered.*
3. *If field selection is used, the sign of the AR will always be negative.*
4. *Indirect addressing, field selection, and multi-word operands may be employed.*

Illustration

Transfer the operand, FIELDB (1002), to AR8 reversing the sign(s) of the operand.

LAN 8, FIELDB

I/A	X	OP Code	AR	m
0	0000	12	1000	1002

LOAD FIELD INTO REGISTER — LF

Operation:	*$(m')_{FSEL} \longrightarrow ARi$*
OP Code:	*14*
Cycles:	*3*

Description: *Selectively replace consecutive bits within the arithmetic register(s) designated with the bits from corresponding positions of the memory location(s) specified.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed location of Field Select Control Word*

Notes

1. *Bit positions to be replaced and operand address are specified in a Field Select Control Word. (FSEL).*

2. *If the field selection option is not exercised, the instruction functions as the Load instruction except that the sign of AR remains unchanged.*

3. *Bits outside the limits specified remain unchanged.*

4. *The sign of the arithmetic register(s) will not be affected.*

5. *Indirect addressing and multi-word operands may be employed.*

6. *See* Field Selection, *page 4–4.*

Illustration

Extract bit position 1–12 of FIELDA (0789) into AR1. The Field Select Control Word is located in 0289.

LF 1, * (12, 1, FIELDA)

I/A	X	OP Code	AR	m
1	0000	14	0001	0289

FSEL (0289)

I/A	X	Left Boundary Bit	Right Boundary Bit	L
0	0000	15	4	0789

STORE — ST

Operation: *(ARi) ⟶ m'*
OP Code: *10*
Cycles: *2*

Description: *Transfer the contents of the arithmetic register(s) designated to the indexed memory location(s).*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *The indexed memory location(s) are first automatically cleared to binary 0's.*

2. *Contents of the arithmetic register(s) are not altered.*

3. *Indirect addressing, multi-word operands, but not field selection, may be employed.*

Illustration

Transfer the contents of AR2 and 4 to FIELDB (0551–0552). SA 6, FIELDB

I/A	X	OP Code	AR	m
0	0000	10	0110	0552

STORE A NEGATIVELY — SAN

Operation: *(ARi) ⟶ m'*
OP Code: *11*
Cycles: *2*

Description: *Transfer the contents of the arithme-register(s) designated to the indexed memory location(s) reversing the sign(s) of the operand.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *The indexed memory location(s) are first automatically cleared to binary 0's.*
2. *Contents of the arithmetic register(s) are not altered.*
3. *Indirect addressing and multi-word operands, but not field selection, may be employed.*

Illustration

Transfer the contents of AR4 to FIELDC (0482) reversing the sign.

SAN 4, **FIELDC**

I/A	X	OP Code	AR	m
0	0000	11	0100	0482

ARITHMETIC INSTRUCTIONS

All arithmetic operations are performed in the adder. One input to the adder, the primary, always comes from some combination of the four arithmetic registers: AR1, AR2, AR4, AR8. The other input, the secondary, is from the indexed location specified by the instruction. The result of an arithmetic operation is usually returned to the same arithmetic register or registers from which the primary operand was secured; this return of the result replaces the original operand in the arithmetic register(s). However, the result may be placed in some other arithmetic register, in which case the primary operand is unchanged. The rule is: *The result of an arithmetic operation will be located in one place and one place only.* In decimal or binary subtractions and additions, the Equal Comparison Indicator (ECI) is set, if the result is decimal or binary 0; if the result is non-zero, the ECI is reset.

DECIMAL ADD — DA

Operation:	*(ARi) + (m') ⟶ AR*
OP Code:	*20*
Cycles:	*2*

Description: *Algebraically add in decimal the operand (augend) in the indexed memory location(s) and the value (addend) in the designated arithmetic register(s). The result is placed in the same arithmetic regis'er(s).*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index registers, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the augend*

Notes

1. *Binary 0's (0000) in either the addend or augend will be treated as decimal excess-three 0's (0011). See Appendix for treatment of non-numeric binary codes.*
2. *Indirect addressing, field selection, and multi-word operands may be employed.*
3. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.

4. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Add FIELDA (0525) to AR8.

DA 8, FIELDA

I/A	X	OP Code	AR	m
0	0000	20	1000	0525

DECIMAL ADD HIGHER | DAH

Operation: *(ARi) + (m') ⟶ ARi where i' < i*
OP Code: *22*
Cycles: *2*

Description: *Algebraically add, in decimal, the operand (augend) in the indexed memory location(s) and the value (addend) in the higher arithmetic register(s), placing the result in the designated arithmetic register(s).*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the augend*

Notes

1. *The addend will be undisturbed.*
2. *Pure binary 0's (0000) in either the addend or augend will be treated as decimal excess-three 0's (0011). See Appendix for treatment of non-numeric binary codes.*
3. For single-word operands, all possible cases of i and i' are:
   if i is 8, i' may be 4, 2 or 1.
   if i is 4, i' may be 2 or 1.
   if i is 3, i' may be 1 only.
   i may not be 1.
4. *Multi-word usage is restricted to Arithmetic Register 12. The sum will always appear in Arithmetic Register 3. Bits 11–14 of the instruction word in this case should be all 1's.*
5. *Indirect addressing and field selection may be employed.*
6. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in the* Multi-Word Operands, *and* Field Selection Sections.
7. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Add FIELDD (0585) to AR8 and place the sum in AR2.

DAH 10, FIELDD

I/A	X	OP Code	AR	m
0	0000	22	1010	0585

DECIMAL SUBTRACT | DS

Operation: *(ARi) – (m') ⟶ ARi*
OP Code: *21*
Cycles: *2*

Description: *Algebraically subtract in decimal the operand (subtrahend) in the indexed memory location(s) from the value (minuend) in the designated arithmetic register(s), placing the result in the same arithmetic register(s).*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the subtrahend*

Notes

1. *Pure binary 0's (0000) in either the subtrahend or minuend will be treated as decimal excess-three 0's (0011). See Appendix for treatment of non-numeric binary codes.*
2. *Indirect addressing, field selection and multi-word operands may be employed.*
3. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.
4. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Subtract FIELDA *(0565) from AR*1.

DS 1, **FIELDA**

I/A	X	OP Code	AR	m
0	0000	21	0001	0565

DECIMAL SUBTRACT HIGHER — DSH

Operation:	*(ARi) – (m′)⟶ARi′, where i′<i*
OP Code:	*23*
Cycles:	*2*

Description: *Algebraically subtract in decimal the operand (subtrahend) in the indexed memory location(s) from the value (minuend) in the designated arithmetic register(s), placing the result in a higher designated arithmetic register(s).*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the subtrahend*

Notes

1. *The minuend will be undisturbed.*
2. *Pure binary 0's (0000) in either the subtrahend or minuend will be treated as decimal excess-three 0's (0011). See Appendix for treatment of non-numeric binary codes.*
3. For single-word operands, all possible cases of i and i' are:
   if i is 8, *i′ may be* 4, 2 *or* 1.
   if i is 4, *i′ may be* 2 *or* 1.
   if i is 2, *i′ may be* 1 *only.*
   i may not be 1.
4. *Multi-word usage is restricted to Arithmetic Register* 12. *The difference will always appear in Arithmetic Register 3. Bits 11–14 of the instruction word in this case should be all 1's.*
5. *Indirect addressing, and field selection may be employed.*
6. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.
7. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Subtract FIELDS (0782) from AR4 placing the difference in AR2.

DSH 6, **FIELDS**

I/A	X	OP Code	AR	m
0	0000	23	0110	0782

DECIMAL MULTIPLY — DM

Operation: *(m') x (AR8) ⟶ AR4 and AR2*
OP Code: *30*
Cycles: *12 to 31 Depending on multiplier digits.*

Description: *Algebraically multiply the contents of the indexed memory location (multiplicand) by the contents of Arithmetic Register 8 (multiplier), placing the six most significant digits of the product in Arithmetic Register 4 and the six least significant digits in Arithmetic Register 2.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	Will designate AR14 *(1110)*
m	*Unindexed address of the multiplicand*

Notes

1. *The multiplier and the multiplicand will not be disturbed.*
2. *All 0's in the multiplier (AR8) and the multiplicand (m) must be excess-three (0011).*
3. *Indirect addressing but not field selection may be employed.*
4. *Multi-word operands may not be used, but note that a 12–digit product is produced.*
5. *See* Arithmetic Modes *for determination of signs and Appendix for timing.*

Illustration

Multiply the contents of AR8 by FieldB (0538).

	DM			FIELDB
I/A	X	OP Code	AR	m
0	0000	30	1110	0538

DECIMAL DIVIDE — DD

Operation: (AR12) ÷ *(m')* ⟶ *AR4(quotient)*, ⟶ *AR8(remainder)*

OP Code: *31*
Cycles: *17–36 Depending upon quotient digits*

Description: *Algebraically divide the contents of Arithmetic Register* 12 *(dividend) by the contents of the indexed memory location (divisor) placing the 6–digit quotient in Arithmetic Register 4 and the 6–digit remainder in Arithmetic Register 8 .*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, to 15*
AR	Will designate AR12 *(1100)*
m	*Unindexed address of the divisor*

Notes

1. *Decimal 0's in the divisor (m) and the dividend* AR12 *must be excess-three (0011).*
2. *If the absolute magnitude of the divisor (m) is less than or equal to that of AR8, the Overflow Indicator will be set and a Contingency Interrupt will occur.*
3. *The sign of the remainder will be that of the dividend.*
4. *Indirect addressing but not field selection may be employed.*
5. *See* Arithmetic Modes *for determination of signs and timing.*

Illustration

Divide the contents of AR12 *by FIELDD (0685).*

	DD			FIELDD
I/A	X	OP Code	AR	m
0	0000	31	1100	0685

BINARY ADD — BA

Operation: $(ARi) + (m') \longrightarrow ARi$
OP Code: 24
Cycles: 2

Description: *Algebraically add in binary the operand (augend) in the indexed memory location(s) and the value (addend) in the designated arithmetic register(s) placing the result in the same arithmetic register(s).*

I/A	X		OP Code		AR		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the augend*

Notes

1. *Indirect addressing, field selection and multi-word operands may be employed.*
2. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.
3. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Add in binary, FIELDA (0789) to AR2.

BA 2, **FIELDA**

I/A	X	OP Code	AR	m
0	0000	24	0010	0789

BINARY ADD HIGHER — BAH

Operation: $(ARi) + (m') \longrightarrow ARi'$ *where* $i' > i$
OP Code: 26
Cycles: 2

Description: *Algebraically add in binary the operand (augend) in the indexed memory location(s) and the value (addend) in the designated arithmetic register(s), placing the result in a higher designated arithmetic register(s).*

I/A	X		OP Code		AR		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the augend*

Notes

1. *The addend will be undisturbed.*
2. For single-word operands, all possible cases of i and i' are:
   if i is 8, i' may be 4, 2 *or* 1.
   if i is 4, *i' may be* 2 *or* 1.
   if i is 2, *i' may be* 1 *only.*
   i may not be 1.
3. *Multi-word usage is restricted to Arithmetic Register* 12. *The sum will always appear in Arithmetic Register* 4. *Bits 11–14 of the instruction in this case should be all 1's.*
4. *Indirect addressing and field selection may be employed.*
5. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.
6. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration:

Add FIELDD (0832) to AR4 and place sum in AR1.

BAH 5, FIELDD

I/A	X	OP Code	AR	m
0	0000	26	0101	0832

BINARY SUBTRACT — BS

Operation: *(ARi) – (m′) ⟶ ARi*
OP Code: *25*
Cycles: *2*

Description: *Algebraically subtract in binary the operand (subtrahend) in the indexed memory location(s) from the value (minuend) in the designated arithmetic register(s), placing the result in the same arithmetic register(s).*

I/A	X		OP Code		AR		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designator of arithmetic register(s)*
m	*Unindexed address of the subtrahend*

Notes

1. *Indirect addressing/field selection and multi-word operands may be employed.*
2. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.
3. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Subtract in binary FIELDD (0823) from AR2.

BS 2, FIELDD

I/A	X	OP Code	AR	m
0	0000	25	0010	0823

BINARY SUBTRACT HIGHER — BSH

Operation: *(ARi) – (m′) ⟶ ARi′ where i′ > i.*
OP Code: *27*
Cycles: *2*

Description: *Algebraically subtract in binary the operand (subtrahend) in the indexed memory location(s) from the value (minuend) in the designated arithmetic register(s), placing the result in a* lower *designated arithmetic register(s).*

I/A	X		OP Code		AR		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the subtrahend*

Notes

1. *The minuend will be undisturbed.*
2. For single-word operands, all possible cases of i and i′ are:
   if i is 8, i′ may be 4, 2 or 1.
   if i is 4, i′ may be 2 or 1.
   if i is 2, i′ may be 1 only.
   i may not be 1.
3. *Multi-word usage is restricted to Arithmetic Register 12. The result will always appear in Arithmetic Register 3. Bits 11–14 of the instruction in this case should be all 1's.*

4. *Indirect addressing and field selection may be employed.*

5. *Additional considerations if the operand is multi-word, or if field selection is to be employed, are discussed in* Multi-Word Operands, *and* Field Selection Sections.

6. *See* Arithmetic Modes *for a discussion of recomplementation and determination of signs.*

Illustration

Subtract in binary FIELDD (0930) from AR8 placing the difference in AR4.

BSH 12 FIELDD

I/A	X	OP Code	AR	m
0	0000	27	1100	0930

SHIFT INSTRUCTIONS

The contents of the arithmetic registers may be altered by the shift instructions. Three distinct methods of shifting, a separate method for each of the three types of data format, may be designated.

DECIMAL SHIFT RIGHT — DSR

OP Code: *40*
Cycles: *4*

Description: *Shift the contents of the arithmetic register(s) designated right the number of decimal digit positions specified in bit positions 1–10 of the instruction.*

I/A	X	OP Code	AR	Shift Count
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
Shift Count	*Unindexed number of places to be shifted expressed in pure binary*

Notes

1. *Digits shifted to the right of the least significant digit position of the operand are lost, and decimal 0's (0011) are inserted in the vacated most significant decimal digit positions.*

2. *The sign bit(s) are not shifted.*

3. *A maximum of a 2–word operand, in adjacent or non-adjacent arithmetic registers may be shifted. The results in either case will always appear in the same registers, leaving the other registers undisturbed.*

4. *Two-word operands cannot be shifted right from one register into another with a* higher *numerical designation, for example, shifting right AR1 and AR4.*

5. *A shift count greater than that of the operand size will result in an error, for example, shifting a 1-word operand nine places. The shift will occur* with Modulo 3 check error which causes a processor error interrupt.

6. *Indirect addressing, but not field selection, may be employed.*

Illustration

Shift the contents of AR6 *four decimal places right.*

DSR 6, 4

I/A	X	OP Code	AR	Shift Count
0	0000	40	0110	0004

DECIMAL SHIFT LEFT — DSL

OP Code: *41*
Cycles: 3

Description: *Shift the contents of the arithmetic register(s) designated left the number of decimal digit positions specified in bit positions 1–10 of the instruction.*

I/A	X	OP Code	AR	Shift Count
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
Shift Count	*Unindexed number of places to be shifted expressed in pure binary*

Notes

1. *Digits shifted to the left of the most significant digit position of the operand are lost and decimal 0's (0011) are inserted in the vacated least significant decimal digit positions of the operand.*
2. *The sign bit(s) are not shifted.*
3. *A maximum of a 2–word operand in adjacent or non-adjacent arithmetic registers may be shifted. The results in either case will always appear in the same registers leaving the other registers undisturbed.*
4. *Two-word operands cannot be shifted left from a register into another with a higher numerical designation, for example, shifting left AR4 and AR1.*
5. *A shift count greater than that of the operand size will result in an error, for example, shifting a 1–word operand nine digits. The shift will occur, causing* a modulo 3 (parity) error and a processor error interrupt.
6. *Indirect addressing, but not field selection, may be employed.*

Illustration

Shift the contents of AR4 three decimal positions left.

DSL 4, 3

I/A	X	OP Code	AR	Shift Count
0	0000	41	0100	0003

ALPHABETIC SHIFT RIGHT — ASR

OP Code: *42*
Cycles: *4*

Description: *Shift the contents of the arithmetic register(s) designated right the number of alphanumeric character positions specified in bit positions 1–10 of the instruction.*

I/A	X	OP Code	AR	Shift Count
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
Shift Count	*Unindexed number of places to be shifted expressed in pure binary*

Notes

1. *Characters shifted to the right of the least significant character position are lost and binary 0's (000000) are inserted in the vacated most significant character positions of the operand.*
2. *The sign bit(s) are not shifted.*
3. *A maximum of a 2–word operand in adjacent or non-adjacent arithmetic registers, may be shifted. The results in either case will always appear in the same registers, leaving the other registers undisturbed.*
4. *Two-word operand cannot be shifted right from a register into another with a lower numerical designation, for example, shifting right AR1 and AR4.*
5. *A shift count greater than the operand size will result in an error, for example, shifting a 1–word operand nine character positions. The shift will occur, causing* a modulo 3 (parity) error and a processor error interrupt.

6. *Indirect addressing, but not field selection may be employed.*

Illustration

Shift the contents of AR8 two character positions right.

ASR 8, 2

I/A	X	OP Code	AR	Shift Count
0	0000	42	1000	0002

ALPHABETIC SHIFT LEFT — ASL

OP Code: *43*
Cycles: *3*

Description: *Shift the contents of the arithmetic register(s) designated left the number of alphanumeric character positions specified in bit positions 1–10 of the instruction.*

I/A	X	OP Code	AR	Shift Count
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
Shift Count	*Unindexed number of places to be shifted expressed in pure binary*

Notes

1. *Characters shifted to the left of the most significant character position of the operand are lost. Binary 0's (000000) are inserted in the vacated least significant character positions of the operand.*
2. *The sign bits are not shifted.*
3. *A maximum of a 2–word operand, in adjacent or non-adjacent arithmetic registers, may be shifted. The results in either case will always appear in the same registers leaving the other registers undisturbed.*
4. *Two-word operands cannot be shifted left from a register into another with a* lower *numerical designation, for example, shifting left AR4 and AR1.*
5. *A shift count greater than the operand size will not result in any error, for example, shifting a 1–word operand nine character positions. The shift will occur, causing* a modulo 3 (parity) error and a processor error interrupt.
6. *Indirect addressing, but not field selection may be employed.*

Illustration

Shift the contents of AR2 'wo character positions left.

ASL 2, 2

I/A	X	OP Code	AR	Shift Count
0	0000	43	0010	0002

BINARY ROTATE RIGHT — BRR

OP Code: *44*
Cycles: *4*

Description: *Shift circularly the contents of the arithmetic register designated right the number of bit positions specified in bit positions 1–10 of the instruction.*

I/A	X	OP Code	AR	Shift Count
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register*

Shift Count	*Unindexed number of places to be shifted expressed in pure binary*

Notes

1. *Bits shifted beyond the least significant bit position re-enter in the most significant bit positions of the same register so that no bits are lost.*
2. *The sign bit* is *shifted.*
3. *The maximum size of the operand is one word.*
4. *A shift count greater than 25 will result in an error.*
5. *Indirect addressing, but not field selection, may be employed.*

Illustration

Shift the contents of AR8 twenty bit positions right.

BRR 8, **020**

I/A	X	OP Code	AR	Shift Count
0	0000	44	1000	0020

COMPARISON INSTRUCTIONS

These instructions perform four distinct types of comparisons. In each case the contents of the arithmetic register is compared to the contents of the indexed address. Each of these instructions sets one of the comparison indicators reflecting the relationship of the contents of the arithmetic register(s) to those of the indexed memory location. The setting of the individual indicators may later be tested and a logical branch operation executed as a result. If Field Selection is employed, only the selected bits are compared.

COMPARE MAGNITUDE	CM

Operation: $|(ARi)| : |(m')|$
OP Code: *55*
Cycles: *2*

Description: *Compare the absolute magnitude of the arithmetic register(s) designated with the absolute magnitude of an operand in memory. Set the appropriate comparison indicator according to the following:*

if $|(ARi)| > |(m')|$, *set* Greater Comparison Indicator
if $|(ARi)| < |(m')|$, *set* Less*Comparison Indicator*
if $|(ARi)| = |(m')|$, *set Equal Comparison Indicator*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Prior to the setting of the appropriate indicator all comparison indicators are automatically reset.*
2. *The operands are not altered.*
3. *Comparison is based on the binary value of the operands regardless of word format. See Figure 2-1.*
4. *Indirect addressing, field selection and multiword operands may be employed.*

Illustration

*Compare the absolute magnitude of AR2 with the absolute magnitude of FIELD*A *(0732).*

CM 2, **FIELDA.,**

I/A	X	OP Code	AR	m
0	0000	55	0010	0732

COMPARE ALGEBRAIC	C

Operation: *(ARi) : (m')*
OP Code: *54*
Cycles: *2*

Description: *Algebraically compare the contents of the arithmetic register(s) designated with an operand in memory. Set the appropriate comparison indicator according to the following:*

If (ARi) > (m'), set Greater Comparison Indicator
If (ARi) < (m'), set Less *Comparison Indicator*
If (ARi) = (m'), set Equal Comparison Indicator

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Prior to the setting of the appropriate indicator, all comparison indicators are automatically reset.*
2. *Plus 0 will compare greater than a minus 0.*
3. *The operands are not altered.*
4. *Comparison is based on the binary value of the operands regardless of word format. See Figure 2-1.*
5. *Only the sign of the least significant word of a multi-word operand is considered. All other signs are ignored.*
6. *Indirect addressing, field selection and multi-word operands may be employed.*

Illustration

*Compare algebraically the contents of AR*1 *with FIELD*A *(0835).*

C 1, FIELDA

I/A	X	OP Code	AR	m
0	0000	54	0001	0835

COMPARE PRODUCT WITH A REGISTER — CPA

Operation: *(ARi) 1–bits : (m') 1–bits*
OP Code: *57*
Cycles: *2*

Description: *Compare the 1–bits of the arithmetic register(s) designated with the 1–bits of the operand in memory. If the latter contains a 1–bit in every bit position in which the arithmetic register(s) contains a 1–bit, set the Equal Comparison Indicator; otherwise set the High Comparison Indicator.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Sign bits are included in the comparison.*
2. *Before setting the appropriate indicator all comparison indicators are automatically reset.*
3. *The operands are unaltered.*
4. *Indirect addressing, field selection and multi-word operands may be employed.*

Illustration

*Compare the 1–bits of AR*4 *with the 1–bits of FIELD*D *(0823).*

CPA 4, FIELDD,

I/A	X	OP Code	AR	m
0	0000	57	0100	0823

COMPARE PRODUCT WITH ZERO — CPZ

Operation: *(ARi) 1–bits : (m′) 0–bits*
OP Code: *56*
Cycles: *2*

Description: *Compare the 1–bits of the arithmetic register(s) designated with the 0–bits of the operand in memory. If the latter contains a 0–bit in every bit position in which the arithmetic register(s) contains a 1–bit, set the Equal Comparison Indicator; otherwise set the High Comparison Indicator.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Sign bits are included in the comparison.*
2. *Before setting of the appropriate indicator all comparisons indicators are reset.*
3. *The operands are unaltered.*
4. *Indirect addressing, field selection and multi-word operands may be employed.*

Illustration

Compare the 1–bits of AR2 with the 0–bits of FIELDD (0834).

CPZ 2, **FIELDD**

I/A	X	OP Code	AR	m
0	0000	56	0010	0834

LOGICAL BRANCHING INSTRUCTIONS

The sequence of execution of instructions may be altered depending upon the state (set or reset) of the indicators affected by previous instructions. Thus a branch in the program or a conditional transfer of control may be accomplished. If the indicator tested is reset, the next instruction in sequence will be accessed and executed. If the indicator is set, control will be transferred to any point in the program desired. Control may also be transferred unconditionally.

JUMP IF EQUAL — JE

Operation: *Test Indicator:*
If set, $m' \longrightarrow CC$.
If reset, $(CC) + 1 \longrightarrow CC$
OP Code: *60*
Cycles: *1 if set; 2 if reset*

Description: *Test the Equal Comparison indicator. If set, transfer control to the indexed memory address. Otherwise, access the next instruction in sequence.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Indicator	*0110*
m	*Unindexed address of the next instruction to be accessed if indicator is set*

Notes

1. *The condition of the indicator will not be affected by the test.*
2. *The state of this indicator may also be affected by addition and subtraction instructions. If a zero result is produced, it will be set. It will be reset if a non-zero result is produced.*
3. *Indirect addressing may be employed.*

Illustration

Transfer control to LOCC (0932) if the Equal Comparison Indicator is set.

JE LOCC

I/A	X	OP Code	Indicator	m
0	0000	60	0110	0932

Illustration

Transfer control to LOCD (0839) if the Greater *Comparison Indicator is set.*

JG LOCD

I/A	X	OP Code	Indicator	m
0	0000	60	0111	0839

JUMP IF HIGH — JG

Operation:	*Test Indicator:*
	If set, $m' \longrightarrow CC$
	If reset, $(CC) + 1 \longrightarrow CC$
OP Code:	*60*
Cycles:	*1 if set; 2 if reset*

Description: *Test the* Greater Comparison Indic. *If set, transfer control to the indexed memory address. Otherwise, access the next instruction in sequence.*

I/A	X		OP Code		Indicator		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Indicator	*0111*
m	*Unindexed address of the next instruction to be accessed if indicator is set*

Notes

1. *The condition of the indicator will not be affected by the test.*
2. *Indirect addressing may be employed.*

JUMP IF LESS — JL

Operation:	*Test Indicator:*
	If set, $m' \longrightarrow CC$
	If reset $(CC) + 1 \longrightarrow CC$
OP Code:	*60*
Cycles:	*1 if set; 2 if reset*

Description: *Test the* Less *Comparison Indicator. If set, transfer control to the indexed memory address. Otherwise access the next instruction in sequence.*

I/A	X		OP Code		Indicator		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Indicator	*0101*
m	*Unindexed address of the next instruction to be accessed if indicator is set*

Notes

1. *The state of the indicator will not be affected by the test.*
2. *Indirect addressing may be employed.*

Illustration

Transfer control to LOCB (0938) if the Less Comparison Indicator is set.

JL **LOCB**

I/A	X	OP Code	Indicator	m
0	0000	60	0101	0938

JUMP IF POSITIVE — JP

Operation: *Test Indicator:*
If set, $m' \longrightarrow CC$
If reset, $(CC) + 1 \longrightarrow CC$
OP Code: *60*
Cycles: *1 if set; 2 if reset*

Description: *Test the Sign Indicator of the arithmetic register addressed. If set, transfer control to the indexed address. Otherwise, access the next instruction in sequence.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Indicator	*Designation (See below.)*
m	*Unindexed address of the next instruction to be accessed if indicator is set*

Notes

1. *Each Sign Indicator will be set or reset depending on the sign of the word currently in the respective arithmetic register. If the sign is positive the indicator will be set, if negative it will be reset.*

2. *The designations of the Sign Indicators are:*

AR8	*0001*	*1*
AR4	*0010*	*2*
AR2	*0011*	*3*
AR1	*0100*	*4*

3. *Indirect addressing may be employed.*

Illustration

Transfer control to LOCD (0659) if the sign of AR2 is positive.

TPOS 3, LOCD ,

I/A	X	OP Code	Indicator	m
0	0000	60	0011	0659

JUMP — J

Operation: *$m' \longrightarrow CC$*
OP Code; *06*
Cycles: *1*

Description: *Replace the contents of the Control Counter with the indexed address of the instruction.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect address option*
X	*Binary address of index register, 0 to 15*
AR	*Not relevant*
m	*Unindexed address of the next instruction to be accessed*

Notes

1. *Indirect addressing but not field selection may be employed.*

Illustration

Transfer control to LOCC (0783).

J **LOCC**

I/A	X	OP Code	AR	m
0	0000	06	0000	0783

STORE LOCATION AND JUMP	SLJ
STORE CHANNEL AND JUMP	SCJ

Operation: $(CC) + 1 \longrightarrow m'$ and $m' + 1 \longrightarrow CC$

OP Code: *07*

Cycles: *3*

Description: *Transfer the contents of the Control Counter, incremented by 1 (or if specified the MAC incremented by 1) into bit positions 1–15 of the indexed memory location and replace the contents of the Control Counter with the indexed memory address incremented by 1.*

I/A	X	OP Code	CC/MAC	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
CC/MAC	*Normally 0001 (See note 2 below.)*
m	*Unindexed address minus 1 of the next instruction to be accessed*

Notes

1. *Bit positions 16–25 of the indexed location will be binary 0's.*

2. *If a Memory Address Counter plus 1 is desired, the designations are:*

UNISERVO III Basic Write	*0011*	3
UNISERVO III Basic Read	*0100*	4
General Purpose #1	*0101*	5
General Purpose #2	*0110*	6
General Purpose #3	*0111*	7
General Purpose #4	*1000*	8
General Purpose #5	*1001*	9
General Purpose #6	*1010*	10
General Purpose #7	*1011*	11
General Purpose #8	*1100*	12
Compatible Tape Read-Write	*1101*	13
UNISERVO III Additional Write	*1110*	14
UNISERVO III Additional Read	*1111*	15

3. *The contents of the Memory Address Register (15 bits) plus 1 may also be transferred to memory by placement of 0010 in bit positions 11–14 of the instruction.*

4. *Indirect addressing but not field selection may be employed.*

Illustration

Store the contents of the Control Counter incremented by 1 in LOCB (0839) and transfer control to 0840.

SLJ LOCB

I/A	X	OP Code	CC/MAC	m
0	0000	07	0001	0839

SENSE INDICATOR INSTRUCTIONS

The following instructions refer to eight indicators that may be used for program control. Each may be set, or reset and tested, with branching occurring if the indicator is set.

SET SENSE INDICATOR	SS

OP Code: *62*

Cycles: *2*

Description: *Set the Sense Indicator (1–8) specified in bits 11–14 of the instruction.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Not relevant*
X	*Not relevant*
Indicator	*Designation*
m	*Not relevant*

Notes

1. *The designations of the Sense Indicators are:*

Sense Indicator	*#1*	*1000*	8
Sense Indicator	*#2*	*1001*	9
Sense Indicator	*#3*	*1010*	10
Sense Indicator	*#4*	*1011*	11
Sense Indicator	*#5*	*1100*	12
Sense Indicator	*#6*	*1101*	13
Sense Indicator	*#7*	*1110*	14
Sense Indicator	*#8*	*1111*	15

2. *Indirect addressing, field selection and multiword operands are not applicable.*

Illustration

Set Sense Indicator #8.

SS , 15,

I/A	X	OP Code	Indicator	m
0	0000	62	1111	0000

RESET SENSE INDICATOR — RS

OP Code: *61*
Cycles: *2*

Description: *Reset the Sense Indicator (1–8) specified in bits 11–14 of the instruction.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Not relevant*
X	*Not relevant*
Indicator	*Designation*
m	*Not relevant*

Notes

1. *See Note 1 above for Sense Indicator designations (bits 11–14).*

2. *Indirect addressing, field selection and multiword operands not applicable.*

Illustration

Reset Sense Indicator #4.

RS , 11,

I/A	X	OP Code	Indicator	m
0	0000	61	1011	0000

JUMP IF SENSE INDICATOR SET — JS

Operation: *Test Indicator:*
If set, $m' \longrightarrow CC$
If reset, $(CC) + 1 \longrightarrow CC$

OP Code: *60*
Cycles: *1 if set; 2 if reset*

Description: *Test the Sense Indicator designated. If set, transfer control to the indexed address. Otherwise access the next instruction in sequence.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Indicator	*Designation*
m	*Unindexed address of the next instruction*

Notes

1. *The condition of the indicator is not affected by the test.*

2. *See Note 1 above for sense indicator designations (bits 11–14).*

3. *Indirect addressing may be employed.*

Illustration

Transfer control to LOCC (0832) if Sense Indicator #3 is set.

JS, 10, LOCC,

I/A	X	OP Code	Indicator	m
0	0000	60	1010	0832

CONVERSION INSTRUCTIONS

These instructions provide the facility to convert data in decimal format to alpha-numeric format or data in alpha-numeric format to decimal format, and to convert non-significant characters to non-printing codes. Such instructions may be used to prepare input data for processing and/or output.

LOAD A CONVERTING TO DECIMAL — LAD

Operation:	$(m'-2,\ m'-1,\ m') \rightarrow AR_i - 1,\ AR_i$
OP Code:	*72*
Cycles:	*7*

Description: *Transfer the contents of three consecutive memory locations of alpha-numeric format into two adjacent arithmetic registers in decimal format by removing the zone bits.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand in alpha-numeric format*

Notes

1. *A 3–word alpha-numeric operand in memory is "compressed" into a 2–word decimal operand in the arithmetic registers.*

2. *It is assumed that the operand in memory is a numeric (in 6–bit code) rather than alphabetic representation. There is no check for the presence of zone bits.*

3. *The signs of the result in the arithmetic registers will be that of the least significant word of the operand in memory.*

4. *The operand in memory will not be altered.*

Illustration

Convert FIELDB (0830–0832) from alpha-numeric format to decimal format and locate the result in AR12.

LAD, 12, FIELDB + 2,

I/A	X	OP Code	AR	m
0	0000	72	1100	0832

STORE A CONVERTING TO ALPHA-NUMERIC — SAA

Operation:	$(AR_i - 1,\ AR_i) \rightarrow m' - 2,\ m' - 1,\ m'$
OP Code:	*71*
Cycles:	*8*

Description: *Transfer the contents of two adjacent arithmetic registers of decimal format into three consecutive indexed memory locations in alpha-numeric format by inserting zero zone bits.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand in alpha-numeric format*

Notes

1. *A 2–word decimal operand in the arithmetic registers is "expanded" to a 3–word alpha-numeric operand in memory.*

2. *The signs of the result in memory will be that of the least significant word of the operand in the arithmetic registers.*

3. *The contents of the arithmetic registers are not altered.*

4. *Indirect addressing, but not field selection may be employed.*

Illustration

Convert to alpha-numeric format a decimal operand located in AR12, *storing it in FIELD*B *(0681–0683).*

SAA, **12,** **FIELD**B + 2

I/A	X	OP Code	AR	m
0	0000	71	1100	0683

ZERO SUPPRESS — LAE

OP Code: *73*
Cycles: *2*

Description: *Transfer the contents of the indexed memory location(s) to the arithmetic registers designated replacing alpha-numeric 0's (00 0011) and commas (11 0010) to the left of the first significant non-zero character with non-printing space codes (00 0000).*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand (See #3 below.)*

Notes

1. *The operand in memory is unaltered.*

2. *The original sign(s) are retained.*

3. *A multi-word operand must be located in consecutive memory locations, but the suppressed result may be in non-adjacent arithmetic registers.*

4. *When the operand is multi-word, the indexed memory location must be the address of its* ***most*** *significant word.*

5. *Indirect addressing, but not field selection may employed.*

Illustration

*Zero suppress FIELD*B *(0689–0690) placing the result in* AR12.

LAE, **12,** **FIELD**B

I/A	X	OP Code	AR	m
0	0000	73	1100	0689

LOGICAL INSTRUCTIONS

These instructions allow bit manipulation in the UNIVAC III System. The operation table which applies to each affected bit of the arithmetic register(s) has the following form:

$m \backslash AR_i$	*(AR_i) before execution*
(m')	*(AR_i) after execution*

OR — OR

Operation: $(m') \longrightarrow AR_i$
OP Code: *15 1–bits*
Cycles: *2*

Description: *Transmit all 1–bits in the indexed memory location(s) to the corresponding bit positions in the arithmetic register(s) designated.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Bit positions in the arithmetic register(s) that correspond to 0-bits in the operand are not altered.*
2. *The operand in memory is not altered.*
3. *A logical "or" operation is performed on the entire operands, including sign bits. The truth table is:*

m \ *AR*	*0*	*1*
0	*0*	*1*
1	*1*	*1*

4. *Indirect addressing, field selection and multi-word operands may be employed.*

Illustration

LOGICAL "OR" *FIELD*B *(0823)* with AR2

OR, 2, **FIELD**B

I/A	X	OP Code	AR	m
0	0000	15	0010	0823

AND

AND

Operation: $(m') \xrightarrow[\text{0-bits}]{} AR_i$

OP Code: *16*

Cycles: *2*

Description: *Transmit all 0-bits in the indexed memory location(s) to the corresponding bit positions in the arithmetic register(s) designated.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*
AR	*Positional designation of arithmetic register(s)*
m	*Unindexed address of the operand*

Notes

1. *Bit positions in the arithmetic register(s) that correspond to 1-bits in the operand are not altered.*
2. *The operand in memory is not altered.*
3. *A logical "and" operation is performed on the entire operand, including sign bits, for which the truth table is:*

m \ *AR*	*0*	*1*
0	*0*	*0*
1	*0*	*1*

4. *Indirect addressing, field selection and multi-word operands may be employed.*

Illustration

LOGICAL "AND" *FIELD*E *(0832)* with AR1

AND 1, **FIELD**E

I/A	X	OP Code	AR	m
0	0000	16	0001	0832

INDEX REGISTER INSTRUCTIONS

The following instructions provide for the loading, storing, incrementing and comparing of index register contents used for the indexing of all instructions.

LOAD INDEX REGISTER — LX

Operation:	*(m′) ⟶ XO_i bits 1–15*
OP Code:	*51*
Cycles:	*3*

Description: *Transfer bits 1–15 of the indexed memory location to the index register designated in bit positions 11–14 of the instruction.*

I/A	X	OP Code	XO	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, to 15*
XO	*Binary address of index register (1 to 15) operand*
m	*Unindexed address of value to be loaded*

Notes

1. *Indirect addressing may be employed. Field selection and multi-word operands do not apply.*

Illustration

Load index register 12 with the value found in AMTA (0389).

LX 12, AMTA

I/A	X	OP Code	XO	m
0	0000	51	1100	0389

STORE INDEX REGISTER — SX

Operation:	*(XOi) ⟶ m′*
OP Code:	*50*
Cycles:	*3*

Description: *Transfer the contents of the index register designated in bit positions 11–14 of the instruction to bit positions 1–15 of the indexed memory location.*

I/A	X	OP Code	XO	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address to index register, 0 to 15*
XO	*Binary address of index register (1 to 15) operand*
m	*Unindexed address of storage location*

Notes

1. *Bit positions 16–25 of the indexed memory location will be binary 0's.*
2. *If XO is 0000, bit positions 1–25 of m′ will contain binary 0's.*
3. *Indirect addressing may be employed. Field selection and multi-word operands do not apply.*

Illustration

Store Index Register 10 in AMTB (0834).

SX, 10, AMTB

I/A	X	OP Code	XO	m
0	0000	50	1010	0834

INCREMENT INDEX REGISTER — IX

Operation:	*$(XO_i) + (m')$ ⟶ XO_i bits 1–9*
OP Codes:	*52*
Cycles:	*3*

Description: *Algebraically add in binary bit positions 1–9 (augend) of the indexed memory location to the index register designated (addend) in bits 11–14 of the instruction.*

I/A	X	OP Code	XO	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address to index register, 0 to 15*
XO	*Binary address of index register (1 to 15) operand*
m	*Unindexed address of increment*

Notes

1. *If the sign of the indexed memory location is negative, the addition to the index register is in effect a decrementation.*
2. *Any carry beyond the most significant bit position of the index register is ignored.*
3. *Indirect addressing may be employed. Field selection and multi-word operands do not apply.*

Illustration

Increment Index Register 12 by the value in AMTB (0772).

IX 12, AMTB

I/A	X	OP Code	XO	m
0	0000	52	1100	0772

INCREMENT INDEX REGISTER AND COMPARE — IXC

Operation: $(XO_i) + (m')_{\text{bits } 1-9} \longrightarrow XO_i$

$|(XO_i)| : |(m')|_{\text{bits } 10-24}$

OP Code: *53*

Cycles: *4*

Description: *Algebraically add in binary bit positions 1–9 (increment amount) of the indexed* Increment and Compare word (ICW) to the *index register designated in bits 11–14 of the instruction. Compare in absolute the new contents of the index register with bit positions 10–24 (comparison amount) of the* ICW *and set the appropriate comparison indicator according to the following:*

if $|(XOi)| > |(m')|$ *bits 10–24, set* Greater Comparison *Indicator.*

if $|(XOi)| < |(m')|$ *bits 10–24, set* Less Comparison *Indicator.*

if $|(XOi)| = |(m')|$ *bits 10–24, set Equal Comparison Indicator*

I/A	X	OP Code	XO	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
XO	*Binary address of index register (1 to 15) operand*
m	*Unindexed address of XMOD*

Notes

1. *The* ICW *is in the following format:*

Sign	Comparison Amount	Increment Amount
25	24 10	9 1

2. *If the sign bit (25) of the* ICW *is one, the increment amount is added as a negative value, in effect decrementing the index register.*
3. *Any carry beyond the most significant bit position of the index register is ignored.*
4. *Prior to the setting of the appropriate indicator, all comparison indicators are reset.*
5. *Indirect addressing may be employed. Field selection and multi-word operands do not apply.*

Illustration

Increment Index Register 5 by 3 and compare the contents to the value 45. The ICW is located in INCR (0489).

IXC, 5, INCR

I/A	X	OP Code	XO	m
0	0000	53	0101	0489

(0489)
INCR ICW 45,3,

Sign	Comparison Amount	Increment Amount
0	45	3

PROCESSOR INTERRUPT INSTRUCTIONS

The cause of two classes of automatic program interrupt, *Processor Error* and *Contingency,* may be determined by these instructions. When the condition is rectified, the affected indicator may then be reset, and normal processing may continue.

TEST CONTINGENCY INDICATOR — TC

Operation:	*Test Indicator:* *If set, (CC) + 1 ⟶ CC* *If reset, (CC) + 2 ⟶ CC*
OP Code:	*64*
Cycles:	*2*

Description: *Test the contingency indicator(s) specified in bit positions 1–10. If one or more is set, access the next instruction in sequence. Otherwise, skip the next instruction in sequence.*

I/A	X	OP Code	Class	Indicator
25	24 21	20 15	14 11	10 1

I/A	*Indirect address option*
X	*Binary address of index registers, 0 to 15*
Class	*0010*
Indicator(s)	*Positional designation of specific indicator(s)*

Notes

1. *Any number of indicators may be tested by placement of 1–bits in bit positions 1–10. If an indicator is set, the next instruction in sequence will be accessed, (CC) + 1 ⟶ CC.*

2. *The condition of the indicator(s) will not be affected by the test.*

3. *Indicators are designated by 1–bits in the following bit positions. (Bit positions 7–10 should be 0's.)*

	ADDRESSES	
Overflow	000001	01
Invalid OP Code	000010	02
Console Typewriter Interrupt	000100	04
Keyboard Request	001000	010
Keyboard Release	010000	020
Contingency Stop	100000	040

4. *The location immediately following the instruction will normally be an unconditional transfer.*

5. *Indirect addressing may be employed.*

Illustration

Test the Contingency Stop Indicator.

TC 040

I/A	X	OP Code	Class	Indicator
0	0000	64	0010	0000100000

RESET CONTINGENCY INDICATORS — RCI

OP Code:	*65*
Cycles:	*2*

Description: *Reset the Contingency Indicator(s) specified in bit positions 1–10 of the instruction.*

I/A	X		OP Code		Class		Indicator	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Class	*0010*
Indicator(s)	*Positional designation of specific indicator(s)*

Notes

1. *Any number of indicators may be reset. The inclusion of several 1–bits will result in the resetting of all indicators designated.*
2. *Indicators are designated in the same way as for Test Contingency Indicator Note 3*
3. *Any attempt to reset an indicator in a reset condition will not result in an error.*
4. *Resetting of any indicator will automatically reset the Contingency Interrupt Mode Indicator and inhibit all interrupts until after execution of the following instruction.*
5. *Indirect addressing may be employed.*

Illustration

Reset the Overflow Indicator.

		RC		1
I/A	X	OP Code	Class	Indicator
0	0000	65	0010	0000000001

TEST PROCESSOR ERROR INDICATOR(S) — TPE

Operation:	*Test Indicator:* *If set, (CC) + 1 → CC* *If reset, (CC) + 2 → CC*
OP Code:	*64*
Cycles.	*2*

Description: *Test the Process error indicator(s) specified in bit positions 1–10. If one or more is set, access the next instruction in sequence. Otherwise, skip the next instruction in sequence.*

I/A	X		OP Code		Class		Indicator	
25	24	21	20	15	14	11	10	1

I/A	*Indirect address option*
X	*Binary address of index register, 0 to 15*
Class	*0001*
Indicator(s)	*Positional designation of specific indicator(s)*

Notes

1. *Any number of indicators may be tested by placement of 1–bits in bit positions 1–10. If an indicator is set, the next instruction in sequence will be accessed; (CC) + 1 → CC.*
2. *The condition of the indicator(s) is not affected by the test.*
3. *Indicators are designated by* the following address:

	UTMOST
Memory Address Error during:	
Instruction Access	1
Operand Access	2
Synchronizer Access by:	
UNISERVO III Basic Write	3
UNISERVO III Basic Read	4
General Purpose #1	5
General Purpose #2	6
General Purpose #3	7
General Purpose #4	8
General Purpose #5	9
General Purpose #6	10
General Purpose #7	11
General Purpose #8	12
Compatible Tape	13
UNISERVO III Additional Write	14
UNISERVO III Additional Read	15
Modulo 3 Check on Instruction	16
Modulo 3 Check on Operand	32
Adder Error Check	64

4. *The location immediately following the instruction will normally be an unconditional transfer.*

5. *Indirect addressing may be employed.*

Illustration

Test the Modulo 3 Check On Instruction Indicator.

TPE 16,

I/A	X	OP Code	Class	Indicator
0	0000	64	0001	0000010000

RESET PROCESSOR ERROR INDICATOR(S) — RPE

OP Code: *65*
Cycles: *2*

Description: *Reset the Processor Error Indicator(s) specified in bit positions 1–10 of the instruction.*

I/A	X	OP Code	Class	Indicator
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register 0 to 15*
Class	*0001*
Indicator(s)	*Positional designation of specific indicator(s)*

Notes

1. *Any number of indicators may be reset. The inclusion of several 1–bits will result in the resetting of all the indicators designated.*

2. *Indicators are designated in the same way as for Test Processor Error Indicators, Note 3*

3. *Any attempt to reset an indicator already in a reset condition will not result in an error.*

4. *Resetting of any indicator will automatically reset the Processor Error Interrupt Mode Indicator and inhibit all interrupts until after execution of the following instruction.*

5. *Indirect addressing may be employed.*

Illustration

Reset the Adder Error Check Indicator.

RPE 64

I/A	X	OP Code	Group	Indicator
0	0000	65	0001	0001000000

INPUT-OUTPUT INTERRUPT INSTRUCTIONS

The third class of automatic program interrupt, *Input-Output,* is handled by these instructions. The channel synchronizer originating the interrupt and the specific cause of it may be determined. Normal processing will be resumed when the affected indicators are reset.

TEST INPUT-OUTPUT INDICATORS — TIO

Operation: *Test Indicator:*
If set, (CC) + 1 $\longrightarrow$ CC
If reset, (CC) + 2 $\longrightarrow$ CC
OP Code; *64*
Cycles: *2*

Description: *Test the Input-Output Indicator(s) specified in bit positions 1–10 for the channel specified in bit positions 11–14. If one or more is set, access the next instruction in sequence. Otherwise, skip the next instruction in sequence.*

I/A	X	OP Code	Channel	Indicator
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing/field selection option*
X	*Binary address of index register, 0 to 15*

Channel	*Designator (See below.)*
Indicator	*Positional designation of specific indicator*

Notes

1. *Any number of indicators may be tested by placement of 1–bits in positions 1–10. If an indicator is set, the next instruction in sequence will be accessed; (CC) + 1 → CC.*
2. *The condition of the indicator(s) will not be affected by the test.*
3. *The location immediately following this instruc- should normally contain an unconditional transfer.*
4. *Any attempt to reset an undefined indicator for a given channel or an indicator already in a reset condition will not result in an error.*
5. *Indirect addressing may be employed.*
6. *Channel designations (bits 11–14) are as follows:*

UNISERVO III Basic Write	*0011*	3
UNISERVO III Basic Read	*0100*	4
General Purpose #1	*0101*	5
General Purpose #2	*0110*	6
General Purpose #3	*0111*	7
General Purpose #4	*1000*	8
General Purpose #5	*1001*	9
General Purpose #6	*1010*	10
General Purpose #7	*1011*	11
General Purpose #8	*1100*	12
Compatible Tape Read Write	*1101*	13
UNISERVO III Additional Write	*1110*	14
UNISERVO III Additional Read	*1111*	15

7. *Indicators are designated by 1–bits in the following bit positions (bits 8–10 should be 0):*

	Bit Positions
Stand-by Location Interlock Indicator	*1*
Completion/Initiation Interrupt	*2*
Error A (UNISERVO Units Only)	*3*
Busy (UNISERVO Units Only)	*4*
Error B	*5.*
Error for General Purpose Channels	*5*
End of File (727 Tape)	*5*
End of Tape (UNISERVO III Unit Only)	*6*
Out-of-paper (High-Speed Printer)	*6*
Wired Stop Character (Paper Tape)	*6*
Fault	*7*
Low on Paper (Paper Tape)	*2 and 6*
Bad Line Printed	*5 and 7*

Illustration

Test the Stand-by Location Interlock Indicator for UNISERVO III Basic Write Channel.

TIO 3, 1

I/A	X	OP Code	Channel	Indicator
0	0000	64	0011	0000000001

RESET INPUT-OUTPUT INDICATOR(S) — RIO

OP Code: *65*
Cycles: *2*

Description: *Reset the input-output indicator(s) specified in bit positions 1–10 for the channel specified in bit positions 11–14.*

I/A	X	OP Code	Channel	Indicator
25	24 21	20 15	14 11	10 1

I/A	*Indirect address option*
X	*Binary address of index register, 0 to 15*
Channel	*Designator (See below.)*
Indicator(s)	*Positional designation of specific indicator(s)*

Notes

1. *Any number of indicators may be reset. The inclusion of several 1–bits will result in the resetting of all indicators designated.*

2. *For channel designations (bits 11–14) see Note 6 of preceding instruction.*

3. *Indicators are designated by 1–bits as specified in Note 7 of the preceding instruction.*

4. *Any attempt to reset an undefined indicator for a given channel or an indicator already in a reset condition will not result in an error.*

5. *Resetting of any indicator will automatically reset the Input-Output Interrupt Mode Indicator and inhibit all interrupts until after execution of the following instruction.*

6. *Indirect addressing may be employed.*

Illustration

Reset the Stand-by Location Interlock Indicator for UNISERVO III Basic ReadChannel.

RIO 4, 1

I/A	X	OP Code	Channel	Indicator
0	0000	65	0100	0000000001

PREVENT INPUT-OUTPUT INTERRUPT	PI

OP Code: *62*
Cycles: *2*

Description: *Set the Inhibit Input-Output Interrupt Indicator thereby preventing all subsequent Input-Output Interrupts from occurring.*

I/A	X		OP Code		Indicator		m	
25	24	21	20	15	14	11	10	1

I/A	*Should be 0*
X	*Not relevant*
Indicator	*Should be 0000*
m	*Not relevant*

Notes

1. *Storage of the Control Counter reading and transfer of control to location 0020 will be blocked as long as the indicator is set.*

2. *The setting of the indicator will not affect any subsequent setting or resetting of the Input-Output Indicators.*

3. *Indirect addressing and field selection are not applicable.*

Illustration

Inhibit all Input-Output Interrupts from occurring.

PI 0

I/A	X	OP Code	Indicator	m
0	0000	62	0000	0000

ALLOW INPUT-OUTPUT INTERRUPT	AI

OP Code: *61*
Cycle: *2*

Description: *Reset the Inhibit Input-Output Interrupt Indicator thereby allowing the occurrence of all subsequent input-output interrupts.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Should be 0*
X	*Not relevant*
Indicator	*Should be 0000*
m	*Not relevant*

Notes

1. *An Input-Output Interrupt or Input-Output Error Indicators may be set during the time Input-Output Interrupts are inhibited. A normal Input-Output Interrupt will occur when this indicator is reset.*

2. *Indirect addressing and field selection are not applicable.*

Illustration

Allow input-output interrupts to occur.

AI 0

I/A	X	OP Code	Indicator	m
0	0000	61	0000	0000

JUMP IF INPUT-OUTPUT INTERRUPT PREVENTED — JJP

Operation:	*Test Indicator:*
	If set, $m' \rightarrow CC$
	If reset, $(CC) + 1 \rightarrow CC$
OP Code:	*60*
Cycles:	*1 if set; 2 if reset*

Description: *Test the Inhibit Input-Output Indicator. If set, transfer control to the indexed address. Otherwise access the next instruction in sequence.*

I/A	X	OP Code	Indicator	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Indicator	*Should be 0000*
m	*Unindexed address of the next instruction to be accessed if indicator is set*

Notes

1. *The condition of the indicator is not affected by the test.*

2. *Indirect addressing may be employed.*

Illustration

Transfer control to LOCE (0839) if input-output interrupt is inhibited.

JIP **LOC**E

I/A	X	OP Code	Indicator	m
0	0000	60	0000	0839

INITIATE INPUT-OUTPUT INSTRUCTION

Input-output function specifications, denoting the particular input-output operations to be performed, are not decoded and executed in the Central Processor. Execution of Initiate Input-Output Instruction makes the input-output function specification available to the appropriate channel synchronizer which executes it.

LOAD CHANNEL STANDY REGISTER — LC

Operation:	*$(m') \rightarrow SLi$ and set appropriate Stand-by Location Interlock Indicator*
OP Code:	*70*
Cycles:	*3*

Description: *Transfer the function specification from the indexed memory location to the fixed stand-by location in memory associated with the channel designated in bit positions 11–14 and set the respective Stand-by Location Indicator.*

I/A	X	OP Code	Channel	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Channel	*Channel designator*
m	*Unindexed address of the function specification*

Notes

1. *Input-output operations, except those pertaining to the Console Typewriter, are executed by means of two instructions – the* initiate input-output *instruction and a* function specification *(FS). The latter serves to direct the peripheral unit to perform a specific function – read a card, read a block, print a line, and so on. Function specifications have the following formats.*

TAPE FUNCTIONS

0	Servo Number	Function Code	I	L-Addr.
25	24 21	20 17	16	15 1

HIGH-SPEED PRINTER FUNCTIONS

0	Number of Lines Paper Advance	FUNCT'N CODE	I	L-Addr.
25	24 19	18 17	16	15 1

HIGH-SPEED CARD READER AND CARD-PUNCH FUNCTIONS

00000	Function Code	I	L-Addr.
25 21	20 17	16	15 1

PAPER-TAPE READER AND PUNCH FUNCTIONS

0	No. of Words	FUNCT'N CODE	I	L-Addr.
25	24 19	18 17	16	15 1

The initiate input-output function places the FS in the memory location associated with the channel so that it may be picked up by the channels control circuitry, decoded, and executed. To inform the channel circuitry that a FS is available, the Stand-by Location Indicator is set.

Operation of the initiate input-output function and the input-output function specification is as follows:

Execution of the initiate input-output function places an input-output function specification into the stand-by location for the synchronizer designated and sets the corresponding Stand-by Location Indicator.

When the related synchronizer successfully completes the execution of a previous instruction, the synchronizer requests access to its stand-by location if its Stand-by Location Interlock Indicator is set. When the Memory Priority Circuits grant the Synchronizer the requested access, and the contents of the stand-by location are transferred to the Channel Control Circuitry where the function is defined. During the transfer, bit functions 1–15 are loaded into the synchronizer's Memory Address Counter. The Stand-by Location Interlock Indicator will be reset when the operation is successfully initiated and the instruction execution begins (when the instruction applies to the tape units and to the Printer.)

If the Stand-by Location Interlock Indicator is set, and an initiate input-output function is executed, the associated, input-output function specification will replace the one in the stand-by location. In normal use the indicator should be tested and found reset prior to the execution of an initiate input-output function. If the Indicator is found set, and the initiate I-O command is executed, there is the possibility that the instruction already in the stand-by location will not be executed while the new one is being entered. Pesetting of the Indicator may be accomplished by the RIO instruction.

Whenever input-output functions cannot be successfully completed because of error or abnormal conditions, the stand-by location Interlock Indicator for the appropriate synchro izer remains reset. The instruction in its stand-by location will therefore not be transferred for execution.

2. *The address of the memory locations associated with the channel is the binary value of the channel designator.*

3. *Indirect addressing but not field selection may be employed.*

4. *See Note 2 of for channel addresses.*

Illustration

*Initiate a tape operation for the Basic Read Channel. The function specification is located in LOC*B *(0839).*

LC 4, **LOC**B

I/A	X	OP Code	Channel	m
0	0000	70	0100	0839

MISCELLANEOUS INSTRUCTIONS

NO OPERATION	NOP

Operation: *(CC) + 1 $\longrightarrow$ CC*
OP Code: *00*
Cycles: *2*

Description: *No operation is performed. Access the next instruction in sequence.*

I/A	X		OP Code		AR		m	
25	24	21	20	15	14	11	10	1

I/A	*0*
X	*Not Relevant*
OP Code	*00*
AR	*Not Relevant*
m	*Not Relevant*

Notes

1. *Memory, arithmetic registers and indicators are not affected.*

STORE LOCATION STORE CHANNEL	SL SC

Operation: *(MACi) $\longrightarrow$ m'*
OP Code: *04*
Cycles: *3*

Description: *Transfer the contents of the Memory Address Counter (MAC) for the channel specified in bit positions 11–14 (or the Control Counter if specified) into bit positions 1–15 of the indexed memory location.*

I/A	X		OP Code		MAC/CC		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
MAC/CC	*Normally channel designator (See below.)*
m	*Unindexed address*

Notes

1. *Bit positions 16–25 of the indexed location will be binary 0's.*

2. *If the Control Counter is desired, bit positions 11–14 should be 0001 (*SL*) If a Memory Address Counter is desired, the channel designations are:*

UNISERVO III Basic Write	*0011*
UNISERVO III Basic Read	*0100*
General Purpose #1	*0101*
General Purpose #2	*0110*
General Purpose #3	*0111*
General Purpose #4	*1000*
General Purpose #5	*1001*
General Purpose #6	*1010*
General Purpose #7	*1011*
General Purpose #8	*1100*
Compatible Tape Read-Write	*1101*
UNISERVO III Additional Write	*1110*
UNISERVO III Additional Read	*1111*

3. *The Memory Address Counter for the channel designated at the time of transfer will contain:*

UNISERVO III Unit Scatter Read or Gather Write	– *Address of the Tape Control Word currently effective in the UNISERVO III Read or Write Synchronizer.*
Compatible Servos or UNISERVO III Unit Read W/O Control Word	– *Address to or from which the last data word transfer took place.*
High-Speed Printer	– *Address of last word transferred to the Printer Synchronizer Buffer.*
Card-Punch Unit	– *Address of the last word transferred from Punch Synchronizer.*
High-Speed Reader	– *Address of the last word transferred from High-Speed Reader Synchronizer.*

4. *The contents of the Memory Address Register (15 bits) may also be transferred to memory by placement of 0010 in bit positions 11–14 of the instruction.*

5. *Indirect addressing but not field selection may be employed.*

Illustration

*Store the MAC for the Basic Read Channel in LOC*B *(0839).*

SC 4, **LOC**B

I/A	X	OP Code	Channel	m
0	0000	04	0100	0839

STORE TAPE CONTROL REGISTER — ST

Operation: *(TCRi)* $\longrightarrow$ *m'*
OP Code: *05*
Cycles: *3*

Description: *Transfer the contents of the Tape Control Word Register (TCWR) for the UNISERVO III synchronizer channel specified in bits 11–14 to the indexed memory location.*

I/A	X		OP Code		Channel		m	
25	24	21	20	15	14	11	10	1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Channel	*For channel designation, see Note 2 below.*
m	*Unindexed memory location*

Notes

1. *The indexed memory location will contain the following information:*

Bits 1–15	*Binary address of the last word transferred to or from the synchronizer channel*
Bits 16–24	*Original count as contained in the Scatter Read/Gather Write Control Word, decremented by one for each word transferred*
Bit 25	*Sign; Positive*

2. *The UNISERVO III Read or Write Channel Synchronizer Designations*

	BITS 11–14
Basic Write	*1000*
Basic Read	*0100*
Additional Read	*0010*
Additional Write	*0001*

Note: The above designations apply to this instruction only.

3. *Indirect addressing, but not field selection may be employed.*

Illustration

*Store the TCWR of the Basic Write Synchronizer Channel in FIELD*D *(0832).*

ST 4, FIELDD

I/A	X	OP Code	Channel	m
0	0000	05	1000	0832

HALT AND JUMP — HJ

Operation: *m′ ⟶ CC and Stop Arithmetic and Control Unit*
OP Code: *77*
Cycles: *2*

Description: *Replace the contents of the Control Counter with the indexed address of the instruction and stop the arithmetic and control unit.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect address option*
X	*Binary address of index register, 0 to 15*
AR	*Not relevant*
m	*Unindexed address of the next instruction to be accessed*

Notes

1. *When the Start Key on the console is depressed, the program is resumed at the location specified by the Control Counter reading.*

2. *The arithmetic and control unit ceases to request memory access. All peripheral operations in progress continue to request memory until they are completed. Any function specifications in stand-by locations will be accessed and executed.*

3. *Indirect addressing but not field selection, may be employed.*

Illustrations

Stop the arithmetic and control unit. Then resume the program with the instruction located in LOCB (0839).

HJ   **LOC**B

I/A	X	OP Code	AR	m
0	0000	77	0000	0839

READ CLOCK — RCK

Operation: *(Clock) ⟶ ARi*
OP Code: *76*
Cycles: *2*

Description: *Transfer the reading of the clock to the arithmetic register designated.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Should be 0*
X	*Should be 0*
AR	*Positional designation of arithmetic register*
m	*Should be 0's*

Notes

1. *If the clock is cycling, one-half second every six seconds, an invalid time is transferred to ARi and the next instruction in sequence is accessed; (CC) + 1 ⟶ CC.*

2. *If the clock is not cycling, a valid time is transferred to bit positions 1–20 of ARi with 21–25 binary 0's and the next instruction in sequence is skipped; (CC) + 2 ⟶ CC.*

3. *The valid time is expressed in five 4–bit excess-three digits in the following format:*

00000					
25 20	20 17	16 13	12 9	8 5	4 1
	Hour	Hour	Minute	Minute	Tenth of Minute

4. *If more than one arithmetic register is designated, the clock reading will be transferred to the highest arithmetic register designated.*

5. *If the UNIVAC III System does not include the clock and the instruction is executed, ARi will receive binary 0's and the next instruction in sequence will be accessed.*

6. *The clock, modulo 24 hours, is located inside the Console and is not normally visible to the operator. Knobs are provided on the clock housing to set the hour and minute hands. Power is supplied directly from a 115-volt AC, 60-cycle line.*

7. *If the power to the clock was disrupted, any Load Time instructions executed will set the Overflow Indicator resulting in a Contingency Interrupt. The operator must reset the clock to prevent further Contingency Interrupts when accessing the clock. This is accomplished by depression of a button located on the clock housing.*

8. *Indirect addressing, field selection and multiword operands are not applicable.*

Illustration

Store the clock reading in AR1.

	RCK		L	0
I/A	X	OP Code	AR	m
0	0000	76	0001	0000

WRITE DISPLAY | WD

Operation: *(m')⟶Display*
OP Code: *03*
Cycles: 2

Description: *Transfer the 27–bits of the indexed memory location to the visual display on the Maintenance Panel.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*0*
X	*Binary address of index register, 0 to 15*
AR	*Not relevant*
m	*Unindexed address of operand*

Notes

1. *The Display switch on the panel must be set to position 0.*

Illustration

Display the contents of LOCB (0839).

		WD		LOCB
I/A	X	OP Code	AR	m
0	0000	03	0000	0839

5. Operator's Console

The UNIVAC III Operator's Console contains, in addition to the Console Typewriter and Keyboard and its controls, buttons and lights to control the Central Processor and monitor the peripheral equipment.

AC On-Off Button-Light

Depression of this button when in the off-state, will supply power to the system. If this button is depressed when in the on-state, power will be lost. Use of this button is controlled by a key lock located under the Console apron.

Ready Light

When lit, it indicates that power has been supplied and the Central Processor is ready to operate. There will normally be some lag in its lighting after power has been supplied.

General Clear Button

Depression of the General Clear Button causes the following indicators to be reset:

- Processor Error Interrupt Indicators
- Contingency Interrupt Indicator
- Input-Output Interrupt Indicators
- Interrupt Mode Indicators
- Inhibit Input-Output Interrupt Indicator
- Sense Indicators

Depression of the General Clear Button also causes the following registers to be cleared to binary 0's:

- Control Counter
- Index Registers
- Memory Address Counters

Load Button

Depression of this button causes logical UNISERVO III 0000 to read forward one block without control word. The starting address of the transfer is determined by the Memory Address Counter of the UNISERVO III Read Synchronizer. The Stop Light must be lit for the button to be effective.

Rewind Button

Depression of this button causes the logical UNISERVO III 0000 to rewind without interlock. The button will only be effective if the Stop Light is lit.

Program Run Button-Light

Depression of this button causes the Central Processor to begin execution of instructions the location of which is specified by the Control Counter. The light is lit only during the execution of instructions.

Processor Error Stop/Program Stop

This is a two-section button-light. When the top section, Processor Error Stop, is lit, it indicates that a second Processor Error occurred while in a Processor Error Interrupt Mode causing a stop condition. When the lower section is lit, it indicates that the stop resulted from the execution of a Halt Instruction. When this button is depressed, the Contingency Stop Indicator will be set causing a Contingency Interrupt.

Prevent I/O Interrupt

This light is lit when the Inhibit I/O Interrupt Indicator is set.

Monitor Panel

Eight pairs of lights, indicate the line status of the general purpose channels (lit if off-line) and whether an abnormal (fault) condition exists in any unit requiring operator intervention. Two additional pairs of lights indicate the same conditions for the servo power supplies and the Central Processor.

If an abnormal condition such as no airflow, overheat, power supply failure, and so on, occurs, the appropriate light will be lighted and sound a buzzer. The buzzer may be turned off by depressing the Buzzer Override Button which is on the panel. The indicator light is extinguished when the abnormal condition is corrected. (This panel is not illustrated.)

CONSOLE TYPEWRITER

The UNIVAC III Operator's Console contains in addition to lights and buttons for the operation of the Central Processor, a Console Typewriter and Keyboard.

The typewriter and keyboard are used for the following purposes:

- Typing out data or the contents of the addressable registers, for control purposes under program control.
- Changing the contents of memory location addressable registers by program controlled type-ins.
- Manual typing independent of program control when in an off-line condition.

Specifications:

CHARACTERS

Fifty-one printing alpha-numeric (6-bit) characters as programmed input or output (Figure 5-2).

FORMAT CONTROL

Programmed typewriter actions are controlled by 6-bit non-printing characters. They are:

- Tab Stop (advance carriage to next tab stop).
- Return carriage and space one or two lines.
- Form Feed will advance paper to the pre-set first printing line of the next 5½" or 11" form.
- Bell Ring.

SPEED

Ten characters printed per second.

SPACING

Ten characters per inch horizontal spacing and six lines per inch vertical spacing.

FORM FEED

Sprocket Fed

PAPER WIDTH

Eight and one-half inches including sprocket holes.

NUMBER OF COPIES

Up to five copies plus the original may be produced.

MODES

On-line typewriter functions under program control. Off-line functions as a conventional electric desk typewriter.

		ZONE			
		00	01	10	11
NUMERIC	0000	Δ	&		
	0001	)	:	*	%
	0010	–	.	$	,
	0011	0	CARRIAGE RETURN AND LINE FEED	RING BELL	+
	0100	1	A	J	/
	0101	2	B	K	S
	0110	3	C	L	T
	0111	4	D	M	U
	1000	5	E	N	V
	1001	6	F	O	W
	1010	7	G	P	X
	1011	8	H	Q	Y
	1100	9	I	R	Z
	1101	,	#		
	1110	;		HORIZONTAL TAB	FORM FEED
	1111	(			

Figure 5-2. UNIVAC III Console Typewriter Code

On-Line Mode of Operation

Input from the keyboard and output to be printed is accomplished character-by-character through the 6–bit Typewriter Buffer Register (TBR).

Execution of a Write Typewriter Character (WT) will transfer from memory one 6–bit printable or non-printing typewriter character and initiate a typewriter cycle. Once this is accomplished the Central Processor accesses the next instruction. The character is then printed or the non-printing function executed. At this time, the Console Typewriter Interrupt Indicator is set, causing a Contengency Interrupt.

In order to use the keyboard for input, the Acti-Typewriter (AT) instruction must be executed, before depressing a character key. Depressing a character key will enter in the TBR the proper 6–bit code and set the Console Typewriter Interrupt Indicator causing a Contingency Interrupt. Execution of a Read Typewriter Character (RT) instruction will then transfer the character to the arithmetic register designated. Depression of a character key will not result in a printing or typewriter controlled function.

Typewriter Control Buttons and Associated Indicators

In addition to the keyboard with its printing and non-printing character keys, the following buttons and testable indicators are associated with the Console Typewriter:

KEYBOARD REQUEST BUTTON

Depression will set the Keyboard Request Indicator and cause a Contingency Interrupt to occur. The indicator is tested and reset by programming.

This button is inactive when the typewriter is off-line.

KEYBOARD RELEASE BUTTON

Depression will set the Keyboard Release Indicator and a Contingency Interrupt will occur. The indicator is tested and reset by programming.

This button is inactive when the typewriter is off-line.

KEYBOARD ACTIVE LIGHT

Lit by the execution of an Activate Typewriter (AT) instruction. It is extinguished when either a key or the Keyboard Release Button is depressed. There is no associated program testable indicator.

TYPEWRITER ON-OFF LINE BUTTON-LIGHTS

Indicates the status of the typewriter by the section lit. If on-line, the typewriter is under the direct control of the program. Depression of the button when on-line will put it off-line. The typewriter may then be used manually with printing or non-printing functions occurring when a key is depressed. Depression of the On-Off Line button-light when off-line will put the typewriter on-line.

CONSOLE TYPEWRITER INTERRUPT INDICATOR

This indicator is set when the typewriter is on-line by the depression of a character key or the execution of a printing or non-printing function initiated by a WT instruction.

There is no light indicating the status of this indicator; it is testable and resettable by program only.

Console Typewriter Instructions

The UNIVAC III Console Typewriter will function under program control utilizing these instructions.

WRITE TYPEWRITER CHARACTER	WT

Operation: *If Typewriter on-line:* $(m') \longrightarrow TBR$ *one character*
Then print and $(CC)+2 \longrightarrow CC$
If Typewriter off-line: $(CC)+1 \rightarrow CC$

OP Code: *02*

Cycles: *2*

Description: *If the Console Typewriter is on-line, transfer the alpha-numeric character or function code specified in bit positions 11–14 of the instruction from the indexed memory location to the Typewriter Buffer Register (TBR), initiate a Typewriter Print Cycle, and skip the next instruction in sequence.*

I/A	X	OP Code	Character	m
25	24 21	20 15	14 11	10 1

I/A	*Indirect addressing option*
X	*Binary address of index register, 0 to 15*
Character	*Designation of character position to be printed, 0000–0011. See Note 1.*
m	*Unindexed Address of character to be printed*

Notes

1. *The character to be transferred and printed is designated in bits 11–12 as shown below. Bits 13–14 are not examined and therefore may be 1 or 0.*

CHAR.	4	3	2	1
BITS	24 19	18 13	12 7	6 1
AR BITS	12 11 / 1 1	12 11 / 1 0	12 11 / 0 1	12 11 / 0 0
SALT	3	2	1	0

2. *When the character is printed or function performed, the Typewriter Interrupt Indicator is set causing C6ntingency Interrupt.*

3. *If the Typewriter is off-line the instruction is aborted and the next instruction (normally an unconditional transfer) is accessed.*

Illustration:

Print character 4 from FIELDB (0683).

WT 3, **FIELDB**

I/A	X	OP Code	Character	m
0	0000	02	0011	0683

ACTIVATE TYPEWRITER	AT

OP Code: 66
Cycles: 2

Description: *Allow one alpha-numeric character to be typed in the Typewriter Buffer Register.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Should be 0*
X	*Should be 0's*
AR	*Should be 0's*
m	*Should be 0's*

Notes

1. *The Keyboard Activate Light on the Console will be turned on when the instruction is executed. When a character key is depressed, the light will be extinguished and the Typewriter Interrupt Indicator will be set causing a Contingency Interrupt.*

2. *Depression of a character key will not result in the character being printed.*

3. *The Central Processor will not be interlocked while the character is being typed.*

4. *Indirect addressing, field selection and multi-word operands are not applicable.*

Illustration

Activate the Console Typewriter.

AT 0

I/A	X	OP Code	AR	m
0	0000	66	0000	0000

READ TYPEWRITER CHARACTER	RT

Operation: $(ARi) + (TBR) \longrightarrow ARi$
OP Code: *01*
Cycles: *2*

Description: *Add the alpha-numeric character in the Typewriter Buffer Register (TBR) to bit positions 1–6 of the designated Arithmetic Register.*

I/A	X	OP Code	AR	m
25	24 21	20 15	14 11	10 1

I/A	*Should be 0*
X	*Should be 0's*
AR	*Positional designation of arithmetic register*
m	*Should be 0's*

Notes

1. *Bits 7–25 of the designated arithmetic register will not be affected.*
2. *Indirect addressing, field selection and multi-word operands are not applicable.*
3. *The rules for binary addition apply for bit positions 1–5. For bit position 6, the rules are:*
   - *If a carry from bit position 5 exists, the result in bit position 6 is a 1.*
   - *If a carry from bit position 5 does not exist, the rules for binary addition apply to bit position 6.*
   - *In any case, no carry from bit position 6 is propagated to bit position 7.*

Illustration

Unload the Typewriter Buffer Register into AR2.

RT 2, 0

I/A	X	OP Code	AR	M
0	0000	01	0010	0000

6. Arithmetic Modes

The purpose of this section is to explain briefly the operation of each arithmetic process so that details of the individual instructions may be more fully appreciated.

All arithmetic operations exclusive of those relative to the control unit are accomplished by the arithmetic unit which consists of the adder, arithmetic registers, Central Processor register, and their related circuitry. Each of the five registers involved performs a unique function during all of the arithmetic processes as shown in Figure 6–1.

ADDITION

Signs Equal – True Addition

In either a binary or decimal add with like signs, the operands are transferred to the adder four bits in parallel, the augend from memory and the addend from the arithmetic register(s) specified. The addition is actually binary with any carries resulting from a 4–bit group retained and added to the next higher 4–bit group entering the adder. If a binary add were specified, the result of the addition would be read into the arithmetic register designated. A decimal addition will require the binary sum produced to be corrected prior to its being read in the designated arithmetic registers. This adjustment, requiring no additional time, is the addition of correction factors to each 4–bit group and the ignoring of decimal carries, since the decimal values expressed were in excess-three.

Unequal Signs – Addition with Complementation

Addition with complementation takes place if the signs of both quantities are unequal. In an addition with unequal signs, the data word from memory entering the adder is automatically converted to its 10's complement.** A normal addition then takes place.

The result will take the sign of the input with the greater absolute value. If it is a decimal add, the result would have been corrected for excess-three notation.

Addition with Complementation:

AR (addend)	+226385	−226385
m (augend)	−214360	+214360
Effective Addend (AR)	226385	226385
Complemented Augend (m)	785640	785640
	+1 012025	−1 012025

the carry is ignored

** *In complementing, a 0 remains a 0, a 1 becomes a 9, a 2 becomes an 8, a 3 becomes a 7, and so on. For all digits after the first least significant non-zero digit the 9's complement is used. Therefore in complementing 214360 the following takes place:*

	9	9	9	9	10	
	2	1	4	3	6	0
10's complement	7	8	5	6	4	0

	ADDITION*	SUBTRACTION*	MULTIPLICATION	DIVISION
AR1	ADDEND AND SUM	MINUEND AND DIFFERENCE	MULTIPLIER	6 MSD OF DIVIDEND AND REMAINDER
AR2	ADDEND AND SUM	MINUEND AND DIFFERENCE	6 MSD OF PRODUCT	6 LSD OF DIVIDEND AND QUOTIENT
AR3	ADDEND AND SUM	MINUEND AND DIFFERENCE	6 LSD OF PRODUCT	NEVER INVOLVED
AR4	ADDEND AND SUM	MINUEND AND DIFFERENCE	NEVER INVOLVED	NEVER INVOLVED
CPR	AUGEND	SUBTRAHEND	MULTIPLICAND	DIVISOR

**Only those AR's specified in the instruction will be involved.*

Figure 6-1. Functions of Arithmetic Registers in Arithmetic Processes

Addition with complementation ignores the carry from the most significant digit position and takes the sign of the input with the greater absolute value. Although complementation will occur in an addition with unequal signs, no additional execution time will be expended.

Addition with Recomplementation

In an addition with unequal signs recomplementation will be necessary if the result will change the sign of the addend. Recomplementation will be necessary if the absolute value of the quantity in the AR is less than the absolute value of the quantity from memory. This relationship will necessitate a change in the sign of the AR(s) with recomplementation automatically taking place.

Addition with Recomplementation:

AR (addend)	+218684	−218684
m (augend)	−221896	+221896
Effective Addend (AR)	218684	218684
Complemented Augend (m)	778104	778104
	996788	996788

This is the 10's complement of the correct result and must be recomplemented to

−003212 taking the sign of the input with the greater absolute value. +003212

In these examples, the result of the addition with complementation alone is, in reality, the 10's complement of the true result. This complemented result will be sent through the adder and be recomplemented. Because recomplementation is necessary, a minimum of one additional cycle time will be needed to complete the execution of the instruction. In addition, one cycle time must be added for each word of the result to be recomplemented.

Recomplementation will therefore take place in an addition with unequal signs, if the absolute value of the contents of the AR(s) are less than the absolute value of the contents of the data word from memory.

The three factors which affect the sign and the result of an addition are:

The sign of the AR
The sign of the data word from memory
The absolute value of the operands

		AR	
		+	−
m	WITH EQUAL SIGNS +	+ *	SIGN OF THE GREATER IN ABSOLUTE VALUE**
	WITH UNEQUAL SIGNS −	SIGN OF THE GREATER IN ABSOLUTE VALUE**	− *

* *So long as the signs are equal, the result is a sum even if the signs are both negative.*

** *Although the command is for addition, the presence of unequal signs makes the operation effectively a subtraction. The result is, in reality, a difference.*

Note: If a zero result is developed, its sign is always positive and the Equal Comparison Indicator is set. If the result is not zero, the indicator will be reset.

SUBTRACTION

The same rules which apply to addition apply to subtraction. However, because subtraction affects the sign of the subtrahend (m), the rules are the converse of those for addition.

In a subtraction the sign of the operand from memory is reversed and an addition is performed. If the signs were originally equal, the sign of the subtrahend would change and an algebraic addition occurs. This addition would then involve two quantities with unequal signs. The rules governing complementation and recomplementation take effect if the sign of the AR will change because of the absolute values of the input. In this case, recomplementation automatically occurs.

The factors which will affect the sign and the result of a subtraction are:

The sign of quantity in the AR
The sign of the quantity from memory
The absolute values of the operands

		AR	
		+	−
m	+	SIGN OF THE GREATER IN ABSOLUTE VALUE**	+ *
	−	− *	SIGN OF THE GREATER IN ABSOLUTE VALUE

* *The result of this subtraction is, in reality, a sum because the subtraction operation changes the sign of the subtrahend (m) before the execution of the operation. A true addition would then take place without complementation.*

** *The result of this operation is a difference. The reversing of the sign of the subtrahend would make this operation an addition with unequal signs. This type of operation necessitates complementation. Recomplementation would be necessary if the absolute value of the quantity in the AR were less than the absolute value of the quantity from memory because the relationship would force a change in the sign of the AR(s).*

Note: If a zero result is developed, its sign is always positive.

MULTIPLICATION

Multiplication is accomplished by repeated additions of multiples of either the multiplicand or its tens complement to AR4 (initially cleared to binary 0's.) The selection of the value and number of times it is to be used is governed by the value of each multiplier digit as determined by the value of the multiplier digit to its immediate right. A 12–digit product is produced; the six most significant digits in AR4 and the six least significant digits in AR2.

		MULTIPLIER		
		+	−	
MULTIPLICAND	+	+	−	SIGNS OF THE PRODUCT
	−	−	+	

During the execution of a multiplication, no accesses to memory are required since the multiplier is held in the Central Processor Register and the multiplicand digits in AR4 during the process.

7. Automatic Program Interrupt

Automatic program interrupt in the UNIVAC III Data-Processing System causes, upon automatic recognition of special conditions in the system, the automatic interruption of the program in progress. Depending on the cause of the interrupt, the contents of the Control Counter will be stored in a specific location and control transferred to the succeeding location where the reason for the interruption may be investigated and suitable action taken. Return to the point in the program at which the interrupt occurred may be accomplished by use of the stored Control Counter reading.

The three main causes or classes of interrupt in decending order of priority are *Process Error, Contingency* and *Input-Output.*

When a condition which calls for interrupt arises, the following occurs within the Central Processor:

- A program testable indicator, or group of indicators, is set to specifically identify the cause of the interrupt. The special indicators set will generally belong to the same class of interrupt.

- For each of the three classes of interrupt there is an Interrupt Mode Indicator. These indicators cannot be program set, reset or tested; their functions are automatically controlled. If one is set, interrupts of its respective class or of any class of a lower priority are inhibited; those of a higher class are not.

  The setting of any Mode Indicator will not inhibit the setting of any specific indicator when the appropriate conditions arise.

  In general, when an ending pulse is generated at the end of the execution of each instruction in the Central Processor, the indicators are automatically probed in groups according to the class of interrupt in decending order of priority. In the case of certain Processor Errors, the respective indicators are examined every 4 microseconds. If any specific indicator is found to be set, and if the interrupt Mode Indicator for its class or for classes of higher priority is not set, interrupt will take place. At this time the appropriate Interrupt Mode Indicator is automatically set.

- Depending on the class of interrupt to which the specific indicator found set belongs the current contents of the Control Counter is stored in one of three addressable fixed memory locations; bit positions 1–15 containing the Control Counter reading and bit positions 16–25 containing binary 0's. Control is then transferred to one of three fixed memory locations depending on the class of interrupt.

The specific locations associated with each class of interrupt is as follows:

Class of Interrupt	Storage Location of Control Counter	Transfer of Control to
Processor Error	0016	0017
Contingency	0018	0019
Input-Output	0020	0021

Transfer in thus effected to one of three locations where JUMP to a program may be initiated to determine the exact nature of the interrupt. This determination is made by testing the condition of the specific indicators related to the class of interrupt. During this time the specific indicators are probed as above. When it is known, appropriate action may then be taken, and the specific indicators reset. The reset instruction (**RIO, RPE** or **RC**) will automatically reset the Interrupt Mode Indicator for the class of interrupt involved. Interrupts of *all* classes will then be inhibited, provided *all* the specific indicators are reset, until the completion of the instruction following the reset instruction.

- After the execution of the J instruction, and before the next instruction is accessed, the specific indicators for the class of interrupt just effective, as well as those of a lower class, are again automatically tested for a set condition. If any is found set, the appropriate Interrupt Mode Indicator is set and the Control Counter, containing the return address of the previous interrupt, is stored in the fixed location associated with the class of interrupt of higher priority for which a specific indicator was found set. Control is then transferred to the location associated with the class of interrupt.

- During the course of operation within an Interrupt Mode, that is, an Interrupt Mode Indicator is set, occurrence of an interrupt of a higher priority is always possible and cannot be prevented. Interrupts for all classes will be inhibited until the instruction following the interrupt reset instruction has been executed.

PROCESSOR ERROR INTERRUPT

At the completion of every instruction, regardless of whether any Mode Indicator is set, the Processor Error Indicators are probed for a set condition. If any is set, and the Processor Error Interrupt Mode Indicator (**PEIMI**) is not set, a Processor Error Interrupt will always result immediately without regard to the condition of the lower priority Interrupt Mode Indicators. The **PEIMI** will be set, the Control Counter reading stored in memory location 0016 and control transferred to memory location 0017. If any other Processor Error Indicator is set when the PEIMI is set, the Central Processor will stop. The Control Counter will contain the address plus one of the instructions which caused the error.

During the time the **PEIMI** is set, the setting of specific indicators for the same or lower priority interrupts will not be inhibited. Their action, though, will not be effective until the instruction following the instruction resetting the specific Processor Error Indicator has been executed.

If a Processor Error Indicator is set during the time when either (or both) of the lower priority Interrupt Mode Indicators is set, a Processor Interrupt will occur.

The conditions causing a Processor Interrupt and the special indicator addresses in bit positions 1–10 of the Test (**TPE**) and Reset (**RPE**) instructions are listed below.

Memory Address Check

Incorrect memory addressing of internal and external instructions or operands by the Central Processor (accessed in current instruction cycle) or channel synchronizer (accessed during previous instruction cycle). If the error occurs during a synchronizer access a specific Input-Output Interrupt is set after the Processor Error Interrupt Mode Indicator has been reset.

Depending on when the error occurred, the following designation in bit position 1–4 will test or reset this indicator:

During access of an internal instruction	0001
During access of an internal operand	0010
During access of an input-output data word or function specification by the channel addresses specified (See descriptions of **RPE** and **TPE**.)	0011 to 1111

Modulo 3 Check On Instruction

The instruction or function specification failed the modulo 3 check when accessed from memory. This error is detected after the instruction execution begins.

The indicator is designated by a 1–bit in bit position 5 of the **TPE** and **RPE** instructions.

Modulo 3 Check On Operand

The operand or input-output data word failed the modulo 3 check when transferred to or from memory.

The instruction will be partially executed before the error is detected. An ending pulse is then generated and an interrupt will occur. This error cannot occur on instructions in which a transfer of control is involved.

The indicator is designated by a 1–bit in bit position 6 of the **TPE** and **RPE** instructions.

Adder Error Check

The results of certain instructions failed the modulo 3 check. The check bits of the operand are used to determine the check bits of the result which, in turn, are compared with check bits generated from the bits of the result. If the two pair of check bits are not equal, an error will result. The instructions checked are all Add and Subtracts, Load and Compare, and Compare Absolute.

The indicator is designated by a 1–bit in bit position 7 of the **TPE** and **RPE** instructions.

CONTINGENCY INTERRUPT

The Contingency Interrupt Indicators are probed on the completion of the execution of an internal instruction when an ending pulse is produced. If any is set and neither the Processor Error Interrupt Mode Indicator nor Contingency Interrupt Mode Indicator (**CIMI**) is set, a Contingency Interrupt will result without regard to the state of the Input-Output Interrupt Mode Indicator. The **CIMI** will be set, the Control Counter reading stored in memory location 0018 and control transferred to memory location 0019.

Any specific indicators for the same or lower priority set subsequent to the setting of the **CIMI** and prior to it being reset, will not effect another interrupt, on this or a lower class. If a Processor Error Indicator is set during this time a Processor Error Interrupt will occur.

The conditions resulting in a Contingency Interrupt and the specific indicator addresses in bit positions 1–10 of the test (**TC**) and reset (**RC**) instructions are listed below.

Overflow

A carry beyond the most significant bit or digit was detected in an add or subtract operation, or in a division, when the absolute magnitude of the divisor in memory is less than that of the most significant half of the dividend in AR8 or it is equal to 0.

This indicator will also be set if power to the Program Clock has been dropped at any time prior to the execution of a Load Time instruction without subsequently resetting the clock.

The indicator is designated by a 1–bit in bit position 1 of the **TC** and **RC** instructions.

Invalid Op Code

Attempted execution of an instruction whose operation code is not part of the repertoire immediately producing an ending pulse. No registers or memory locations will be affected by this condition.

The indicator is designated by a 1–bit in bit position 2 of the **TC** and **RC** instructions.

Console Typewriter

The release of a character key on the Console Typewriter Keyboard or a character printed by the Console Typewriter will set the indicator.

The indicator is designated by a 1–bit in bit position 3 of the **TC** and **RC** instructions.

Keyboard Request

This indicator will be set when the Keyboard Request Button is depressed.

The indicator is designated by a 1–bit in bit position 4 of the **TC** and **RC** instructions.

Keyboard Release

This indicator will be set when the Keyboard Release Button is depressed.

The indicator is designated by a 1–bit in bit position 5 of the **TC** and **RC** instructions.

Contingency Stop

Depression of the Stop Button will result in this indicator being set.

The indicator is designated by a 1–bit in bit position 6 of the **TC** and **RC** instructions.

INPUT-OUTPUT INTERRUPT

The Input-Output Interrupt Indicators for all channels are probed by an ending pulse produced by the completion of an internal operation. If any is set, and the Processor Error Interrupt Mode Indicator, Contingency Interrupt Mode Indicator and Inhibit Input-Output Indicator are reset an Input-Output Interrupt will occur. The Input-Output Interrupt Mode Indicator will be set, the Control Counter reading stored in memory location 0020 and control transferred to memory location 0021.

Since this is the lowest priority interrupt any specific indicators of a higher priority interrupt set while the Input-Output Interrupt Mode Indicator is set will immediately result in another interrupt, of the higher class.

The subsequent setting of specific indicators for other channels will not be affected during the time that the Input-Output Interrupt Mode Indicator is set.

Input-Output Interrupt will occur as a result of the following conditions:

- Successful completion or initiation of an input-output operation if called for in the function specification.

- Occurrence of an error or some condition requiring manual instruction when the synchronizer attempts to perform an operation.

See the appropriate bulletin for the specific causes of interrupt and indicators effected.

8. SPECIAL CONSIDERATIONS

The following shift instructions

Decimal Shift Right	DSR
Decimal Shift Left	DSL
Alphabetic Shift Right	ASR
Alphabetic Shift Left	ASL

will cause a stall when executed if more than two AR's are specified.

The following instructions

Decimal Add Higher		DAH
Decimal Subtract Higher	1	DSH
Binary Add Higher		BAH
Binary Subtract Higher		BSH

will cause a stall when executed, if one or three AR's are specified.

The conversion instructions

Load A Converting to Decimal	LAD
Store A Converting to Alphanumeric	SAA

will cause a stall when executed if one, three or four AR's are specified.

Reference to arithmetic register zero can result in a processor error. It should not be used.

Multiplication involving zero generates as a result a properly signed zero.

A store memory address counter instruction specifying the control counter will store the current value rather than the current value plus one.

TIMING OF MULTIPLICATION

Terminology

The multiplier is the factor in Arithmetic Register 8. Each digit is a number from 0 through 9, represented as *n*. Each digit has a position within the multiplier, from 1 through 6, represented as a subscript *i* to the number *n*. The value of the number varies according to the value of the digit on its right, except for the number in position 1, and this digit on the right is represented by the subscript *i–1*. The final value of the number for timing of multiplication purposes is represented by n'. The following formulae state the method of computing n', and the following table gives the number of 4–microsecond cycles required for multiplication according to the value of n'.

For $i = 1$, $n'_i = n_1$.

For $i > 1$, $n'_i = n_i$ *if* $n'_{i-1} < 5$.

For $i > 1$, $n'_i = n_i + 1$ *if* $n'_{i-1} \geq 5$; but if $n_i + 1 = 10$,

$n'_i = 0$, and $n'_{i+1} = n_{i+1} + 1$.

The n_7 is a constructive digit position created to allow for the "righthand" value of n'_6.

$$n'_7 = 0 \text{ if } n'_6 < 5$$

$$n'_7 = 1 \text{ if } n'_6 \geq 5$$

Execution time in 4 μ cycles $= 5 + \sum_{i=1}^{7} T_i$, where T_i is found in the following table:

n'i	T
0	2
1, 2	2
3, 4	3
5	4
6, 7	3
8, 9	2

Thus, for example, if the multiplier is 945270, the execution time is determined as follows:

i	n_i	n'_i	T
1	0	0	2
2	7	7	3
3	2	3	3
4	5	5	4
5	4	5	4
6	9	0	1
7	0	1	2
		$\Sigma\ T_i$ =	19

Multiplication time = 5 + 19 = 24 cycles.

Note: If $n'_i \geq 5$, the ten's complement of the multiplicand is used.

TIMING OF DIVISION

Terminology

Timing of division is computed in a fashion analogous to timing of multiplication. Each digit is a number from 0 through 9, represented as *n* , but the time for execution of division depends entirely upon the digits of the *quotient*. Each digit has a position within the quotient, from 1 through 6, represented as a subscript *i* to the number *n*; but the value of the number varies according to the value of the digit on its left, except for the number in position 6. The digit on the left is represented by the subscript *i + 1*. The final value of the number for timing of division purposes is represented by *n*. The following formulae state the method of computing *n* , and the following table gives the number of 4 μ cycles required for division according to the value of *n* .

For $i = 6$, $n'_i = n_6$.

For $i < 6$, $n'_i = n_i$ *IF* n'_{i+1} is *ODD*.

For $i < 6$, $n'_i = 9 - n_i$ *IF* n'_{i+1} is *EVEN*.

Execution time in 4 μ cycles = $5 + \sum_{i=1}^{6} T_i$, where T_i is found in the following table:

n'_1	T
0, 1	2
2, 3	3
4, 5	4
6, 7, 8, 9	5

Thus, for example, if the *quotient* is 806491, the execution time is determined as follows:

i	n_i	n'_i	T
6	8	8	5
5	0	9	5
4	6	6	5
3	4	5	4
2	9	9	5
1	1	1	2
		ΣT_i =	26

Division Time = 5 + 26 = 31 cycles

MODULO 3 CHECKING IN UNIVAC III SYSTEM

The Parity Bits

The UNIVAC III fixed word consists of twenty-seven bits, two of which are parity bits. These parity bits can be used for two purposes:

1. Checking the transmission of the word to determine if any bits were lost, picked up, or transposed as a result of this process.
2. Checking the result of arithmetic operations without the necessity for programmed checks or duplicated circuitry.

Casting Out of Elevens '

The *casting out of elevens* used to check arithmetic is analogous to modulo 3 congruence arithmetic.

The modulo 11 check value for any number is its remainder when it is divided by 11. As a result of this division, the greatest number of 11's are "cast out" (the quotient) leaving a value less than 11 to be used as the check value. We determine the modulo 11 check value for the following numbers thus:

$$\begin{array}{r} 251 \\ 11\overline{)2762} \\ 2762 \end{array} \quad 1 = \text{check value} \qquad \begin{array}{r} 312 \\ 11\overline{)3438} \\ 3438 \end{array} \quad 6 = \text{check value}$$

Another way the check value may be determined is to subtract the sum of the even numbered digits from the sum of the odd numbered digits.* The units digit is considered odd; the tens digit, even and so on, to the left.

	Sum of Odd Numbered Digits	Sum of Even Numbered Digits	Check Value
2762	2 + 7 = 9	6 + 2 = 8	9 − 8 = 1
3438	8 + 4 = 12	3 + 3 = 6	12 − 6 = 6

We may determine whether the sum of two quantities is correct by adding the modulo 11 check values of the operands and comparing it to the check value of the sum.

	Check Value
2762	1*
+ 3438	+ 6
6200	7

If the sum of the even numbered digits is greater than the sum of the odd numbered digits, a multiple of 11 is added to the latter. When the difference is obtained, the largest multiple of 11 is subtracted.

From the above computation it can be seen that the sum arrived at is correct. The above relationship is always valid no matter how many digits there are in the operands or how many operands there are.

The same theory can also be used for other arithmetic processes. In the case of multiplication, for example, instead of adding the check values of the two operands, we would multiply them and compare it to the check value of the product. They should be equal when the multiplication is correct.

When numbers are copied, digits may often be dropped or inverted. For example, if we were to read the number 2762 and record it, it might be recorded as 2726. Without the original number with which to compare the copy we would never know that the unit and ten digits were transposed. However, if we determine a modulo 11 check value and carry it with the number, any transposition of the original number as 2726 would indicate an error in "transmission."

check value		check value		
1	2762	9	2726	check value incorrect, therefore transmission incorrect.

In conclusion, the check value determined by congruence arithmetic, in the above case modulo 11, can be used to check arithmetic functions and transcriptions of numbers.

Modulo 3 Checking

Using the principles outlined above, we may examine a binary number and develop a method of checking its transmission and arithmetic functions.

Two bits are used in the UNIVAC III System for checking. These two bits may represent values: 00, 01, 10 and 11, or 0, 1, 2, and 3. Since a modulo 3 check is used, the value 3(11) is not possible.

Let us determine the parity or check value, modulo 3, for the following binary configuration:

111101

The decimal value is 61. Since a modulo 3 check value is desired, the quantity is divided by 3, and its remainder becomes its modulo 3 check value.

```
     20
  3)61        1 = check value
    60       01 = binary check value
    --
     1
```

The modulo 3 check value may also be determined by subtracting the total number of the even numbered bits from the total number of the odd numbered bits.

Number of Odd Numbered Bits		Number of Even Numbered Bits	
3	—	2	= 1

As a result of this subtraction, the parity would be 01.

The binary configuration would carry its modulo 3 check value and would appear as:

Modulo 3 Parity	Value
01	111101

In any transmission, a bit which is lost or transposed, would be revealed by the modulo 3 check.

Just as the modulo 11 check value was used to check the results of a decimal addition, so the modulo 3 parity bits may also be used to check a binary addition. For example:

```
Modulo 3
 Parity      Value
   01       011001 = 25
 + 10       001110 = 14
 ----       -----------
   11       100111 = 39
   or
   00
```

Advantages of Modulo 3 Checking

1. The loss of an odd number of bits will be detected.
2. The loss of an even number of non-consecutive bits will be detected.
3. The check bits can be "crossfooted" in addition and subtraction giving a reliable check through the adder.

RESULTS OF DECIMAL ARITHMETIC WITH NON-NUMERIC OPERANDS

A procedure follows for determining the results of decimal add which involves non-numerics (sum with like signs, difference with unlike signs).

A1. Calculate the results of a binary add, retaining carry information from bits 4 to 5, 8 to 9, 12 to 13, 16 to 17, 20 to 21 and 24 to overflow.

A2. Group the result according to decimal format (1–4, 5–8,...21–24)..

A3. Note each 4–bit group with a carry from its most significant bit of the same group.

A4. Convert the 4–bit result according to the following table:

	Decimal Character	
4-bit Group	No Carry	Carry
0000 a	0101 2	0011 0
0001 b	0110 3	0100 1
0010 c	0111 4	0101 2
0011 0	0000 a	0110 3
0100 1	0001 b	0111 4
0101 2	0010 c	1000 5
0110 3	0011 0	1001 6
0111 4	0100 1	1010 7
1000 5	0101 2	1011 8
1001 6	0110 3	1100 9
1010 7	0111 4	1101 f
1011 8	1000 5	1110 g
1100 9	1001 6	1111 h
1101 f	1010 7	1000 5
1110 g	1011 8	1001 6
1111 h	1100 9	1010 7

A5. The result is the final result of an add. Overflow will cause a Contingency Interrupt.

The following procedure is to be followed for subtract (add unlike signs, subtract like signs):

S1. Complement the contents of ARi, and binary add 00...001 to(m′). Use the results as the contents of ARi and m′ for the next step.

S2. Follow add steps A1 through A4.

S3. If overflow results, the answer has been obtained, and will be negative.

S4. If no overflow results, the answer will be positive and must be recomplemented. Repeat subtract step 1 and add steps 1–2 with the contents of m′ assumed to be binary 0's.

S5. This result is the answer.

The following example will illustrate:

```
Decimal add  + f 3 7 b 2 8  = (ARi)
             - a 1 f 3 6 h  = (m′)

Step S1.      (ARi) = 0  1101  0110  1010  0001  0101  1011
Complement    (ARi) = 0  0010  1001  0101  1110  1010  0100
              (m′)  = 1  0000  0100  1101  0110  1001  1111
Binary add 1            0  0000  0000  0000  0000  0000  0001
                        ------------------------------------
                        1  0000  0100  1101  0110  1010  0000
Step S2. A1.               0010  1001  0101  1110  1010  0100
Binary add.                0000  0100  1101  0110  1010  0000
                           ---------------------------------
Step S2. A2.               0010  1110  0011  0101  0100  0100
Step S2. A3.  carry   0      0     1     1     1     0
Step S2. A4.               0111  1011  0110  1000  0111  0001
Step S3.      No carry, therefore S4 applies
Step S4.      (ARi) = 0  0111  1011  0110  1000  0111  0001
              (m′)  = 0  0000  0000  0000  0000  0000  0000
Complement    (ARi) =    1000  0100  1001  0111  1000  1110
      Add to  (m′)  =    0000  0000  0000  0000  0000  0001
Step S4. A1. A2.         1000  0100  1001  0111  1000  1111
Step S5.      (ARi) = + 5 1 6 4 5 h.
```

Communications with the executive system (BOSS III) will be specified later.

MNEMONIC INSTRUCTIONS

Instruction is type 0 unless an A value is listed

Octal OP Code	A Field		Instructions' Function	Timing
61	00	AI	Allow Interrupt	2
16		AND	AND	2
43		ASL	Alphabetic Shift Left	3
42		ASR	Alphabetic Shift Right	4
66	00	AT	Activate Typewriter	2
24		BA	Binary Add	2
26		BAH	Binary Add Higher	2
44		BRR	Binary Rotate Right	4
25		BS	Binary Subtract	2
27		BSH	Binary Subtract Higher	2
54		C	Compare	2
55		CM	Compare Magnitude	2
57		CPA	Compare Product with A	2
56		CPZ	Compare Product with Zero	2
20		DA	Decimal Add	2
22		DAH	Decimal Add Higher	2
31	14	DD	Decimal Divide	17-36
30	16	DM	Decimal Multiply	12-31
21		DS	Decimal Subtract	2
23		DSH	Decimal Subtract Higher	2
41		DSL	Decimal Shift Left	3
40		DSR	Decimal Shift Right	4
77	00	HJ	Halt and Jump	2

Octal OP Code	A Field		Instructions' Function	Timing
52		IX	Increment indeX	3
53		IXC	Increment indeX and Compare	4
06		J	Jump	1
60	06	JE	Jump if Equal	1-2
60	07	JG	Jump if Greater	1-2
60	00	JIP	Jump if Interrupt Prevented	1-2
60	05	JL	Jump if Less	1-2
60		JP	Jump if Positive	1-2
60		JS	Jump if Sense indicator set	1-2
12		LA	Load A	2
72		LAD	Load A converting to Decimal	7
73		LAE	Load A Edited	2
13		LAN	Load A Negatively	2
70		LC	Load Channel	3
14		LF	Load Field	3
70	04	LRC	Load Read Channel	3
70	03	LWC	Load Write Channel	3
51		LX	Load indeX	3
00		NOP	No OPeration	2
15		OR	OR	2
62	00	PI	Prevent Interrupt	2
65	02	RC	Reset Contingency	2
76		RCK	Read ClocK	2
65		RIO	Reset Input-Output	2
65	01	RPE	Reset Processor Error	2
65	04	RR	Reset Read	2
61		RS	Reset Sense	2
01		RT	Read Typewriter character	2

Octal OP Code	A Field		Instructions' Function	Timing
65	03	RW	Reset Write	2
10		SA	Store A	2
71		SAA	Store A in Alphanumeric	8
11		SAN	Store A Negatively	2
04		SC	Store Channel	3
07		SCJ	Store Channel and Jump	3
04	01	SL	Store Location	3
07	01	SLJ	Store Location and Jump	3
04	04	SRC	Store Read Channel	3
05	04	SRT	Store Read Tape control	3
62		SS	Set Sense	2
05		ST	Store Tape control	3
04	03	SWC	Store Write Channel	3
05	10	SWT	Store Write Tape control	3
50		SX	Store indeX	3
50	00	SZ	Store Zero	3
64	02	TC	Test Contingency	2
64		TIO	Test Input-Output	2
64	01	TPE	Test Processor Error	2
64	04	TR	Test Read	2
64	03	TW	Test Write	2
03	00	WD	Write Display	2
02		WT	Write Typewriter character	2

U
T
M
O
S
T

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

PRINTED IN U.S.A. U-3520