

UNIVAC III UTMOST

Nº 07500

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
LQS	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.

August 22, 1962

The mnemonics for machine operation codes as printed in this manual are considered acceptable but non-standard by UNIVAC. This means that the UTMOST processor will accept programs written using either this set of mnemonics or the standard UNIVAC III set.

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

SALT (standard)	UTMOST (non-standard)	SALT (standard)	UTMOST (non-standard)
L	LA	ERS	AND
LCS	LAN	LX	LX
EXT	LF	STX	SX
ST	SA	IX	IX
STCS	SAN	ICX	IXC
---	SZ	TCI	TC
A	DA	RCI	RC
S	DS	TPE	TPE
AH	DAH	RPE	RPE
SH	DSH	TIO	TIO
M	DM	---	TW
D	DD	---	TR
BA	BA	---	TOV
BAH	BAH	---	TOP
BS	BS	RIO	RIO
BSH	BSH	---	RW
SR	DSR	---	RR
SL	DSL	AIO	AI
SAR	ASR	PIO	PI
SAL	ASL	TIOP	JIP
SBC	BRR	IOF	LC
CA	CM	---	LWC
C	C	---	LRC
CONE	CPA	NOP	NOP
CZRO	CPZ	STMC	SC
TEQ	JE	TR*	SCJ
THI	JG	---	SL
TLO	JL	---	SWC
TPOS	JP	---	SRC
TUN	J	STCR	ST
TR	SLJ	---	SRT
SSI	SS	---	SWT
RSI	RS	WAIT	HJ
TSI	JS	LT	RCK
ATD	LAD	DIS	WD
DTA	SAA	RT	RT
ZUP	LAE	WT	WT
SUP	OR	ACT	AT

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

UNIVAC III
UTMOST , U- 3520

August 24,1962

UPDATING PACKAGE A

CONTENTS: SECTION II , pages 3-4,9-10,21-26,29-34,37-55
SECTION IV , page 1
SECTION V , pages 1-4,23-24

The attached sheets are the first changes to the UTMOST manual.

There are three major programming modifications resulting from the implementation and testing of the UTMOST assembler:

1. The format of the DO directive (SECTION II,page 32)
2. The specification of a method and format for writing multiple-word alphanumeric constants useful in typewriter messages and High Speed Printer headings. (SECTION II, page 10)
3. The addition of the COR directive for source code corrections. (SECTION II,page 41)

Read these carefully.

Included,to be modified later, is the interim operating procedures.

Other changes correct typographical errors and omissions.

October 1, 1962

UPDATING PACKAGE B

CONTENTS: INDEX , pages 1-3
SECTION I , pages 1
SECTION II, pages 5-8, 11-12, 15-20, 29-30, 33-63
SECTION III, pages 1-4, 9-10
SECTION V , pages 31-32, 49-52, 63-64
Appendix 1, page 1

The attached sheets are additions and changes to the subject manual.

There are three additions of major importance:

1. The modes of the results using operators with items of mixed modes is specified in SECTION II, pages 17a and 17b,
2. In the reference section, SECTION III, the levels of the operators are specified.
3. Appendix 1 contains the error codes that are used and appear in an output listing of an UTMOST assembly.

The two PROCs, the \$ Editing PROC and the MOVE PROC, have been assembled and tested. The output listings of the assemblies and tests appear in SECTION II, pages 33 and following.

Other changes are minor in nature.

All pages replace the existing pages. Pages that are removed should be destroyed.

February 1, 1963

UPDATING PACKAGE C

CONTENTS:	INDEX	,	pages 1-3
	SECTION I	,	page 1
	SECTION II	,	pages 3-12, 15-42, 53-58
	SECTION III	,	pages 1-6, 9-10
	SECTION V	,	pages 23-24, 29-30, 33-48, 51-54, 69-70
	Appendix 2	,	pages 1-10

The attached sheets are additions and changes to the UTMOST manual.

There are five major revisions:

1. The addition of the GO directive (Section II, page 40).
2. The addition of the NACL directive (Section II, page 41).
3. The addition of Binary Card Formats (Appendix 2, pages 1-10).

Read these carefully.

4. The deletion of the COR directive from the UTMOST assembler. (See SUPPORT III, UPCO, ACCO and DECO).
5. Delete the operating procedure; this will be revised at a later date. (Section IV, page 1).

Other changes correct typographical errors and omissions.

UNIVAC III UTMOST

REVISION:

1

SECTION:

Index

DATE: Oct. 1, 1962

PAGE:

1

TABLE OF CONTENTS

I.	INTRODUCTION	I - 1
II.	BASIC INTRODUCTION TO UTMOST ASSEMBLER LANGUAGE	II - 1
1.	Computers and Languages	II - 1
2.	The UTMOST Assembler	II - 2
3.	Symbolic Coding Format	II - 2
a.	Label Field	II - 3
b.	Operation Field	II - 4
c.	Operand Field	II - 5
d.	Line Control	II - 6
4.	Expressions	II - 7
a.	Elementary Items	II - 7
b.	Operators	II - 12
c.	Mode	II - 13
5.	Data Word Generation	II - 15
a.	ICW, Increment and Compare Word	II - 15
b.	TWC, Two Word Constants	II - 18
c.	+ or - Operation Fields	II - 19
6.	Mnemonic Instructions	II - 22
7.	Line Items	II - 23
8.	Assembler Directives	II - 25
a.	EQU	II - 26
b.	RES	II - 26
c.	USE	II - 27
d.	FORM	II - 28
e.	SET	II - 30
f.	FLD	II - 31

UNIVAC III UTMOST

REVISION: 2

SECTION: Index

DATE: Feb. 1, 1963

PAGE: 2

TABLE OF CONTENTS (Cont'd)

g.	END	II - 31
h.	DO	II - 32
i.	PROC	II - 33
j.	NAME	II - 35
k.	Procedure Reference Lists	II - 36
l.	GO	II - 40
m.	NACL	II - 41
9.	Sample Problem	II - 41
10.	Sample Floating Dollar Sign Editing Procedure	II - 44
11.	Sample MOVE Procedure	II - 56
III.	PROGRAMMER'S REFERENCE GUIDE	III - 1
A.	Line Control	III - 1
B.	Label Field	III - 1
C.	Operation Field	III - 1
D.	Operand Field	III - 2
E.	Expressions	III - 2
F.	Mnemonic Instructions	III - 5
H.	Line Item	III - 5
I.	Addressing	III - 7
J.	Assembler Directives	III - 7
K.	Procedure Reference Line	III - 11
L.	Inter-Program Communication	III - 11

UNIVAC III UTMOST

REVISION:

2

SECTION:

Index

DATE:

Feb. 1, 1963

PAGE:

3

TABLE OF CONTENTS (Cont'd)

IV. OPERATING PROCEDURES	IV - 1
V. UNIVAC III CENTRAL PROCESSOR	V - 1
VI. BOSS III COMMUNICATIONS	VI - 1
VII. MNEMONIC INSTRUCTIONS	VII - 1
Appendix 1 - Error Codes	Appendix 1
Appendix 2 - Binary Card Formats	Appendix 2

UNIVAC III UTMOST

REVISION:

SECTION:

Notes

DATE:

July 1, 1962

PAGE:

UNIVAC III UTMOST

REVISION:

2

SECTION:

I

DATE:

Feb. 1, 1963

PAGE:

1

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 twelve 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.

UNIVAC III UTMOST

REVISION:

SECTION:

Notes

DATE:

July 1, 1962

PAGE:

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.

REVISION: 2	SECTION: II
DATE: Feb. 1, 1963	PAGE: 3

[illegible]

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 sixteen characters. Of these sixteen 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:

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. No special characters may be used in a label field. 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.

4

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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

5

1	LABEL	Δ	OPERATION	Δ	OPERAND
ONE	LA		15, 3, 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 has been encountered (or the maximum number of lists in the case of a procedure reference) or by the end of card, whichever applies to the case in question. 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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

7

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 sixteen alphanumeric, 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.

CONST	+	24
LA	1,	CONST

In the example above, AR1 will be loaded with the value of the label CONST, which is a decimal 24.

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.

LOC	+	\$
-----	---	----

In the example above, if the current value of the location counter was 5280, the integer value 5280 would be produced as a one word constant in decimal, right justified, with preceding binary zeros and a positive sign.

- 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.

0117

07007

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.

9
1024

In the examples above:

9 is equivalent to 000000000000000000001001

1024 is equivalent to 0000000000000000100000000000

- 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.

: 9
 : 1024

In the examples above:

:9 is equivalent to 0000 0000 0000 0000 0000 1100
:1024 is equivalent to 0000 0000 0000 0000 0000 0000

:1024 is equivalent to

10

3. 14 is equivalent to 513140000000 in 4 bit BCD digits.

REVISION:	2	SECTION:	II
DATE:	Feb 1, 1963	PAGE:	11

- EXT(VALUE)

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

- (DON')

('DON') would generate the constant DON in six bit 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.

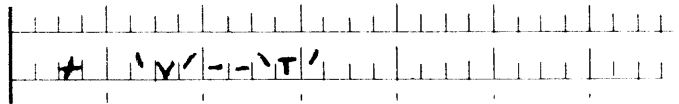
- 2) - Arithmetic Difference : The arithmetic difference operator may be used to subtract one item from another. The assembler will subtract the integer value of the second item from that of the first, and the resultant integer difference will be substituted in the expression.

13

- 4) / Arithmetic Quotient: 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.

14

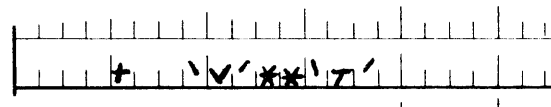
6) -- Logical Difference (EXCLUSIVE OR): The logical difference operator may be used to obtain the logical difference between the integer values of two items. The assembler will perform an EXCLUSIVE OR on the two items (where a bit is present in corresponding position in both items, the result is binary 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.



In the above example:

'V' in six bit code is	111000
'T' in six bit code is	<u>110110</u>
Logical difference is	001110

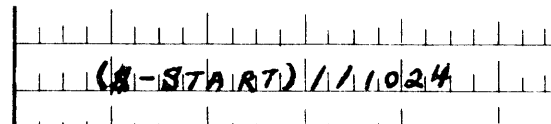
- 7) **** Logical Product (AND):** The logical product operator may be used to AND (Logically multiply) the integer value of one item by another. The assembler will logically multiply the two values and the resulting logical product will be utilized in the expression.



In the above example:

'V' in six bit code is	111000
'T' in six bit code is	<u>110110</u>
Logical product is	110000

- 8) **// Covered Quotient ($a/b = \frac{a+b-1}{b}$):** 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.



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 value of the resultant field is a binary 1. If the two integer values are not equal, the value of the resultant field is a binary 0.

\$ = 7083

In the above example:

If \$ = 7083, the value of the expression is binary 1.

If \$ \neq 7083, the value of the expression is binary 0.

- 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 value of the resultant field is a binary 1. If the first value is less than or equal to the second, the value of the resultant field is a binary 0.

AMOUNT > 2

In the above example:

If the value of AMOUNT is greater than 2, the expression value is binary 1, otherwise it is a binary 0.

- 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 resultant field value is binary 1. If the first value is greater than or equal to the integer value of the second, the value is binary 0.

COUNT < 5

REVISION: 1	SECTION: II
DATE: Feb. 1, 1963	PAGE: 17

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

- $$: 10.0x + : 15$$

```
:10.0*+:15 will produce 671000000000
```

- 15.01-13

: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 -.

UNIVAC III UTMOST

REVISION:

3

SECTION:

II

DATE:

Feb. 1, 1963

PAGE:

17a

- c. The mode of each item within an expression can be different. As the assembler evaluates each item a determination of the mode of the result is made. The determination is contingent upon the operator used, as well as the mode of each item within the expression.

The operators are listed below grouped by function:

GROUP	OPERATOR	DESCRIPTION
A	\approx	Comparison
	$>$	Comparison
	$<$	Comparison
B	$++$	Logical Sum
	$--$	Logical Difference
	$**$	Logical Product
C	$+$	Arithmetic Sum
	$-$	Arithmetic Difference
	$*$	Arithmetic Product
	$/$	Arithmetic Quotient
	$//$	Covered Quotient
D	$*+$	Positive Exponent
	$*-$	Negative Exponent

The following chart depicts the resulting UTMOST mode obtained by combining items of like or unlike mode using any given operator.

UNIVAC III UTMOST

REVISION:

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

17b

MODE OF FIRST ITEM	OPERATOR GROUP	MODE OF SECOND ITEM	MODE OF RESULT
Any	A	Any	Binary
Any	B	Any	Binary
Binary	C	Binary	Binary
Binary	C	Decimal	Binary
Decimal	C	Binary	Binary
Decimal	C	Decimal	Decimal
Any	C	Floating	Floating
Floating	C	Any	Floating
Any	D	Any	Floating

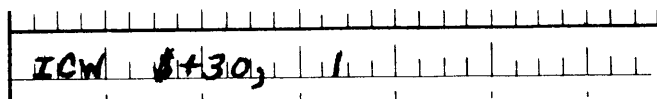
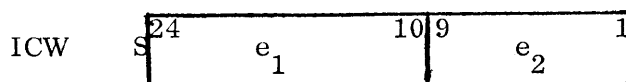
5. Data Word Generation

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 constants, indirect address words and field select words with or without index registers.

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:



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 sign of both words is identical. The left half of the two word constant

UNIVAC III UTMOST

REVISION: 2

SECTION: II

DATE: Feb. 1, 1963

PAGE: 19

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. + or - Operation Field: 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) One word data constants: 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.

UNIVAC III UTMOST

REVISION:

2

SECTION:

II

DATE:

Feb. 1, 1963

PAGE:

20

```

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.

```

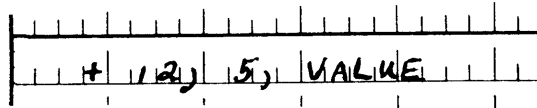
+DATA+10, 9
    
```

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 + 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.

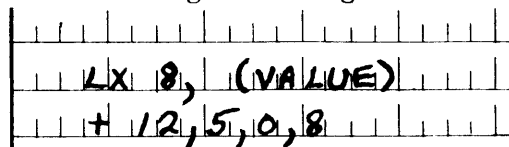
The third expression will generate a ten bit binary address for the word(s) from which the field is to be selected. The sign of the generated word must be positive.



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 + 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.



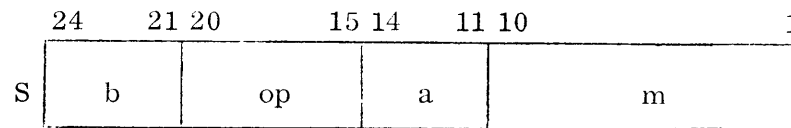
In the above example:

The first expression (the second line) 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. Mnemonic Instructions

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. (See Use Directive, Section II, 8c)

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

UNIVAC III UTMOST

REVISION:

2

SECTION:

II

DATE:

Feb. 1, 1963

PAGE:

23

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')
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
    
```

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, (T 0)
LA 2, (MASK 0,0,0,7,0,0,0,7)
LA 4, ('DON')
    
```

REVISION:	2	SECTION:	II
DATE:	Feb. 1, 1963	PAGE:	24

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 twelve 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.
11.	GO	Means of transfer within a PROC.
12.	NACL	Replace an UTMOST mnemonic.

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.

26

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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE:

Feb. 1, 1963

PAGE:

27

```
PRNTAREA 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 3, PRNTAREA+15
SA 3, PRNTAREA+31
```

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

c. USE

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.

While it is possible to use more than one USE directive in a program, 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 once in a USE directive, or series of USE directives.

UNIVAC III UTMOST

REVISION:

SECTION:

II

DATE:

July 1, 1962

PAGE:

28

USE 5, 4, 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. FORM

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)

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE:

August 24, 1962

PAGE:

29

INST FORM									
1	4	6	4	10					
INST									
0	5	0	2	3	0	1	4	0	0

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	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.


```

WS.1 RES 32
PRINT FORM 1,6,2,1,15
PRINT 0,5,2,1,01004
PRINT 0,2,0,1,0
PRINT 0,6,2,1,MS.1
    
```

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 octal location 1004 through octal 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 the location specified by the RES directive. In all cases interrupt is specified.

e. 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.

```

SET 15, ($)
SL 1
LX 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 following 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 1, 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 Δ , Δ line

The expression, e_1 , must be followed by a space comma. The comma should be followed by a space unless the line to be executed is labelled.

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.

REVISION: 3	SECTION: II
DATE: Feb. 1, 1963	PAGE: 33

```
DO (START+3072)<3, USE 4, 5, 6
```

If the value assumed by the label STRT plus 3072 (3×1024) is less than the current value of the location counter, the value of the expression is a binary 1. In that case, the USE directive on the DO symbolic line will be executed, and three additional index registers will be set up by the assembler. If the condition is not met, the value is 0, and the USE line will not become effective.

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

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

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE:

August 24, 1962

PAGE:

34

```
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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE:

Oct. 1, 1962

PAGE:

35

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)		
	LA	15,	0,	8	
	SA	15,	0,	9	
	IX	8,	(4)		
	IX	9,	(4)		

j. NAME

A NAME directive, or several NAME assembler directives, may be used to qualify a PROC procedure. A NAME line is written at any point within a Procedure where an entrance is desired. 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 15, 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 comments are to appear on the PROC symbolic line, a blank period blank should be used to terminate the line.

UNIVAC III UTMOST

REVISION:

3

SECTION:

II

DATE:

Feb. 1, 1963

PAGE:

37

```
FDOL 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 may have an infinite 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.

```
LX MOVE(4,1), MOVE(1,1)
```

In the example above which is taken from the MOVE PROC, line 34, page 60:

MOVE (4,1) refers to list #4, 1st 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.

```

|-----|
| 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 the value of "label (0,0)" is the value of the expression in the NAME line by which the procedure was referenced.

UNIVAC III UTMOST

REVISION:	3	SECTION:	II
DATE:	Feb. 1, 1963	PAGE:	39

```

FDOL PROC 0
ADOL NAME 0
NDOL NAME 1
DO FDOL(0,0)=0, SAA 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 value of the expression is a binary 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. The last list must be terminated by a period blank if comments are to be written on the line.

REVISION: 3	SECTION: II
DATE: Feb. 1, 1963	PAGE: 40

In the above example:

571 0,8 0,9 50

In the example above:

CHANGE IT HERE, 0 THERE, 0 50 10, 15

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.

1. GO

The GO assembler directive provides the means of transfer to a label specified in the operand field of the directive. This directive can only be used within a PROC. The operand label of the GO line must be the label of a NAME directive within the PROC.

DIRECTIVE FORMAT

<u>LABEL</u>	<u>OP</u>	<u>OPERAND</u>
	GO	ENTER
ENTER	NAME	0

In the above example:

GO will transfer to the NAME directive ENTER within the PROC.

iii. NACL

The NACL assembler directive will cause the UTMOST mnemonic table to be altered. When a mnemonic has been replaced, the new mnemonic must be used for the entire program, or until another directive replacing that mnemonic is issued. The maximum length of the new mnemonic is four alpha characters.

DIRECTIVE FORMAT

<u>LABEL</u>	<u>OP</u>	<u>OPERAND</u>
Mnemonic to be inserted	NACL	Mnemonic to be replaced
TUN	NACL	J
SKIP	NACL	TUN

In the above example:

The NACL directive has been used to replace the mnemonic in the operand field with the code in the label field. The J mnemonic of the UTMOST table is replaced by TUN. TUN now becomes the new mnemonic. The next directive states the method by which TUN may be replaced by SKIP.

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.)

UNIVAC III UTMOST

REVISION: 2

SECTION: II

DATE: Oct. 1, 1962

PAGE:

42

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

LABEL	Δ	OPERATION	Δ	OPERAND	Δ	COMMENTS	7273	80
USE 11, 12, 13.						SET UP INDEX REGISTERS FOR AREA ADDRESSING		
RES 32						PRINTER WORKING STORAGE		
START WOP.								
PRNT EQU 11-33						PROVIDE BASE ADDRESS FOR PRINTER EDIT AREA.		
LIX 8, INM						LOAD IR 10, 7, 8, 9 WITH BASE ADDRESSES OF INPUT AND OUTPUT AREAS		
LIX 9, OUTM						FOR MASTER AND CHANGE FILES AND PRINTER OUTPUT FILE AREA.		
LIX 7, CHNG								
LIX 10, PRNT								
ONE LA 15, 3, 8						SET UP COMPARISON FOR MATCH OR MERGE PROCEDURE		
CM 15, 3, 7								
IF CHNGE.						IF MASTER REC. HAS HIGHER SEQUENCE NUMBER WRITE CHANGE RECORD		
IF EDIT.						IF SEQUENCE NO.'S AGREE, MAKE CHANGE AND PERFORM PRINTER EDIT.		
ST 0, 8 0, 9 50.						INSERT STRAIGHT LINE TRANSFER MASTER INPUT TO OUTPUT PROCEDURE		
SLJ MOUT						EXECUTE OUTPUT RECORD ADVANCE (WRITE)		
SLJ MIN						EXECUTE MASTER RECORD INPUT ADVANCE (READ)		
J ONE						RETURN TO PROCESS NEXT RECORD		
CHNGE IT 0, 7 0, 9 50.						INSERT ITERATIVE TRANSFER PROCEDURE AT THIS POINT		
SLJ MOUT						EXECUTE MASTER RECORD WRITE SUBROUTINE		
SLJ CHNGIN						EXECUTE RECORD READ SUBROUTINE		
J ONE						PROCESS NEXT RECORD		
EDIT SLJ ARRANGE.						EXECUTE EDITING ROUTINE		
SLJ PRINTOUT.						EXECUTE PRINTER TAPE WRITE SUBROUTINE		
IT 0, 8 0, 9 50.						GENERATE MOVE CODING		
SLJ MOUT						EXECUTE WRITE SUBROUTINE		
SLJ MIN						EXECUTE READ SUBROUTINE		

I

43

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, a floating dollar sign to the character position immediately preceding the first significant digit in the field, and a minus sign following the right most digit of the field if the value is negative.

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

Usage:

The procedure expects the value to be edited to be present in AR's 4, 2 and 1 if in alphanumeric format, or in AR's 2 and 1 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 as the operation field of the referencing line.

Reference to Coding:

The floating dollar procedure is found from lines 0006 to 0083 inclusive. It should be noted that the assembler does not produce object code in conjunction with the source code of a procedure. Object code is generated at the point at which the procedure is called. The remaining coding provides an illustration of calls on both versions of the procedure, and was used to test the validity of the routine.

Line 0002 RES places the program in location 023333 and following.

0003 EQUates label DON to location 037777. The value edited by the ADOL call is stored in this location by line 0087.

0004 EQUates label GENE to location 037770. The value edited by the NDOL call is stored in this location by line 0090.

0084 This is the start of the routine to test the procedure. This line and the following line place the value 004565930589 in AR's 4, 2 and 1 in alphanumeric format.

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

45

- 0086 At this point the ADOL version of the procedure is called. Following this line is the object code generated by the call. This coding may be related to source code of the procedure using the mnemonic operation codes on the left of the print out.
- 0087 Stores the four word edited value. The lable DON has been equated to a location which is not within the range covered by the IR's specified by the USE directive. For this reason, the reference to DON is by indirect address in the object code.
- 0088 Loads the value 01234567 in numeric format in AR's 2 and 1.
- 0089 Calls the NDOL version of the procedure.
- 0090 Stores the edited value.
- 0091 SLJ 0250 terminates the run by entering the EOJ (end of job) location of the typewriter control routine.
- 0092 END directive for the program, with label of starting location of the run.

Following line 0092 are the literals produced by the assembler.

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

46

0001		• FLOATING \$ PROC 0762+ 9 19 62 D E W BUCHER
0002	00000	RES 023333
0003	17 0000 23333	USE 15,14,13
0004	00037777	DON EQU 037777
0005	00037770	GENE EQU 037770
0006		FDOL PROC 0
0007		ADOL NAME 0
0008		NDOL NAME 1
0009		DO FDOL(0,0)=0 , SA 7,TEMP+2
0010		DO FDOL(0,0)=1 , SAA 3,TEMP+2
0011		LA 14,TEMP+2
0012		ASR 3,3
0013		AND 1,(0777700)
0014		JP 3,3+2
0015		OR 1,(1-1)
0016		OR 1,(022000000)
0017		LA 2,TEMP+2
0018		ASR 6,2
0019		AND 2,(0777777)
0020		OR 2,(062000000)
0021		LA 4,TEMP+1
0022		ASR 4,1
0023		OR 4,(062000000)
0024		SA 15,TEMP+3
0025		LAE 15,TEMP
0026		SA 15,TEMP+3
0027		LA 12,(0420000000070000)
0028		CPZ 4,TEMP
0029		JE \$+3
0030		LF 15,*(18,1,TEMP+3)

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

47

0031	J	END
0032	LA	12,(042000000007700)
0033	CPZ	4, TEMP
0034	JE	\$+3
0035	LF	15,*(12,1,TEMP+3)
0036	J	END
0037	LA	12,(0420000000077)
0038	CPZ	4, TEMP
0039	JE	\$+3
0040	LF	15,*(6,1,TEMP+3)
0041	J	END
0042	LA	8,(0)
0043	LA	6,(04200000000770000)
0044	CPZ	2,TEMP+1
0045	JE	\$+3
0046	LF	7,*(18,1,TEMP+3)
0047	J	END
0048	LA	6,(042000000007700)
0049	CPZ	2,TEMP+1
0050	JE	\$+3
0051	LF	7,*(12,1,TEMP+3)
0052	J	END
0053	LA	6,(0420000000077)
0054	CPZ	2,TEMP+1
0055	JE	\$+3
0056	LF	7,*(6,1,TEMP+3)
0057	J	END
0058	LA	4,(0)

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE: 48

0059				LA	3,(04200000000770000)
0060				CPZ	1,TEMP+2
0061				JE	\$+3
0062				LF	3,*(18,1,TEMP+3)
0063				J	END
0064				LA	3,(042000000007700)
0065				CPZ	1,TEMP+2
0066				JE	\$+3
0067				LF	3,*(12,1,TEMP+3)
0068				J	END
0069				LA	3,(0420000000077)
0070				CPZ	1,TEMP+2
0071				JE	\$+3
0072				LF	3,*(6,1,TEMP+3)
0073				J	END
0074				LA	1,(030300)
0075				C	1,*(18,7,TEMP+3)
0076				JE	\$+4
0077				LA	2,(1 301)
0078				LA	1,TEMP+3
0079				J	END
0080				LA	3,(0000000000)
0081				J	END
0082				TEMP	RES 4
0083				END	END
0084	17	0000	23333	17	12 03 0237
0085	17	0001	23334	17	12 04 0240
0086	17	0002	23335	17	10 07 0114
				START	LA 3,(1659305891)
					LA 4,(100451)
					ADOL

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

49

LA	17	0003	23336	17	12	16	0114
ASR	17	0004	23337	00	42	03	0003
AND	17	0005	23340	17	16	01	0241
JP	17	0006	23341	17	60	03	0010
OR	17	0007	23342	17	15	01	0242
OR	17	0010	23343	17	15	01	0243
LA	17	0011	23344	17	12	02	0114
ASR	17	0012	23345	00	42	06	0002
AND	17	0013	23346	17	16	02	0244
OR	17	0014	23347	17	15	02	0245
LA	17	0015	23350	17	12	04	0113
ASR	17	0016	23351	00	42	04	0001
OR	17	0017	23352	17	15	04	0245
SA	17	0020	23353	17	10	17	0115
LAE	17	0021	23354	17	73	17	0112
SA	17	0022	23355	17	10	17	0115
LA	17	0023	23356	17	12	14	0247
CPZ	17	0024	23357	17	56	04	0112
JE	17	0025	23360	17	60	06	0030
LF	17	0026	23361	-17	14	17	0250
J	17	0027	23362	17	06	00	0116
LA	17	0030	23363	17	12	14	0252
CPZ	17	0031	23364	17	56	04	0112
JE	17	0032	23365	17	60	06	0035
LF	17	0033	23366	-17	14	17	0253
J	17	0034	23367	17	06	00	0116
LA	17	0035	23370	17	12	14	0255
CPZ	17	0036	23371	17	56	04	0112

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE: 50

JE	17 0037	23372	17 60 06 0042
LF	17 0040	23373	-17 14 17 0256
J	17 0041	23374	17 06 00 0116
LA	17 0042	23375	17 12 10 0257
LA	17 0043	23376	17 12 06 0247
CPZ	17 0044	23377	17 56 02 0113
JE	17 0045	23400	17 60 06 0050
LF	17 0046	23401	-17 14 07 0250
J	17 0047	23402	17 06 00 0116
LA	17 0050	23403	17 12 06 0252
CPZ	17 0051	23404	17 56 02 0113
JE	17 0052	23405	17 60 06 0055
LF	17 0053	23406	-17 14 07 0253
J	17 0054	23407	17 06 00 0116
LA	17 0055	23410	17 12 06 0255
CPZ	17 0056	23411	17 56 02 0113
JE	17 0057	23412	17 60 06 0062
LF	17 0060	23413	-17 14 07 0256
J	17 0061	23414	17 06 00 0116
LA	17 0062	23415	17 12 04 0257
LA	17 0063	23416	17 12 03 0247
CPZ	17 0064	23417	17 56 01 0114
JE	17 0065	23420	17 60 06 0070
LF	17 0066	23421	-17 14 03 0250
J	17 0067	23422	17 06 00 0116
LA	17 0070	23423	17 12 03 0252
CPZ	17 0071	23424	17 56 01 0114
JE	17 0072	23425	17 60 06 0075

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

51

LF	17	0073	23426	-17	14	03	0253
J	17	0074	23427	17	06	00	0116
LA	17	0075	23430	17	12	03	0255
CPZ	17	0076	23431	17	56	01	0114
JE	17	0077	23432	17	60	06	0102
LF	17	0100	23433	-17	14	03	0256
J	17	0101	23434	17	06	00	0116
LA	17	0102	23435	17	12	01	0260
C	17	0103	23436	-17	54	01	0261
JE	17	0104	23437	17	60	06	0110
LA	17	0105	23440	17	12	02	0262
LA	17	0106	23441	17	12	01	0115
J	17	0107	23442	17	06	00	0116
LA	17	0110	23443	17	12	03	0257
J	17	0111	23444	17	06	00	0116
0087	17	0116	23451	-17	10	17	0263
0088	17	0117	23452	17	12	03	0265
0089	17	0120	23453	17	71	03	0232
LA	17	0121	23454	17	12	16	0232
ASR	17	0122	23455	00	42	03	0003
AND	17	0123	23456	17	16	01	0241
JP	17	0124	23457	17	60	03	0126
OR	17	0125	23460	17	15	01	0242
OR	17	0126	23461	17	15	01	0243
LA	17	0127	23462	17	12	02	0232
ASR	17	0130	23463	00	42	06	0002
AND	17	0131	23464	17	16	02	0244
OR	17	0132	23465	17	15	02	0245

SA 15, DON

LA 3, (:01234567)

NDOL

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE:

52

LA	17 0133	23466	17 12 04 0231
ASR	17 0134	23467	00 42 04 0001
OR	17 0135	23470	17 15 04 0245
SA	17 0136	23471	17 10 17 0233
LAE	17 0137	23472	17 13 17 0230
SA	17 0140	23473	17 10 17 0233
LA	17 0141	23474	17 12 14 0247
CPZ	17 0142	23475	17 56 04 0230
JE	17 0143	23476	17 60 06 0146
LF	17 0144	23477	-17 14 17 0266
J	17 0145	23500	17 06 00 0234
LA	17 0146	23501	17 12 14 0252
CPZ	17 0147	23502	17 56 04 0230
JE	17 0150	23503	17 60 06 0153
LF	17 0151	23504	-17 14 17 0267
J	17 0152	23505	17 06 00 0234
LA	17 0153	23506	17 12 14 0255
CPZ	17 0154	23507	17 56 04 0230
JE	17 0155	23510	17 60 06 0160
LF	17 0156	23511	-17 14 17 0270
J	17 0157	23512	17 06 00 0234
LA	17 0160	23513	17 12 10 0257
LA	17 0161	23514	17 12 06 0247
CPZ	17 0162	23515	17 56 02 0231
JE	17 0163	23516	17 60 06 0166
LF	17 0164	23517	-17 14 07 0266
J	17 0165	23520	17 06 00 0234
LA	17 0166	23521	17 12 06 0252

UNIVAC III UTMOST

REVISION: 1

SECTION: II

DATE: Oct. 1, 1962

PAGE: 53

CPZ	17	0167	23522	17	56	02	0231
JE	17	0170	23523	17	60	06	0173
LF	17	0171	23524	-17	14	07	0267
J	17	0172	23525	17	06	00	0234
LA	17	0173	23526	17	12	06	0255
CPZ	17	0174	23527	17	56	02	0231
JE	17	0175	23530	17	60	06	0200
LF	17	0176	23531	-17	14	07	0270
J	17	0177	23532	17	06	00	0234
LA	17	0200	23533	17	12	04	0257
LA	17	0201	23534	17	12	03	0247
CPZ	17	0202	23535	17	56	01	0232
JE	17	0203	23536	17	60	06	0206
LF	17	0204	23537	-17	14	03	0266
J	17	0205	23540	17	06	00	0234
LA	17	0206	23541	17	12	03	0252
CPZ	17	0207	23542	17	56	01	0232
JE	17	0210	23543	17	60	06	0213
LF	17	0211	23544	-17	14	03	0267
J	17	0212	23545	17	06	00	0234
LA	17	0213	23546	17	12	03	0255
CPZ	17	0214	23547	17	56	01	0232
JE	17	0215	23550	17	60	06	0220
LF	17	0216	23551	-17	14	03	0270
J	17	0217	23552	17	06	00	0234
LA	17	0220	23553	17	12	01	0260
C	17	0221	23554	-17	54	01	0271
JE	17	0222	23555	17	60	06	0226

UNIVAC III UTMOST

REVISION:

2

SECTION:

II

DATE:

Feb. 1, 1963

PAGE:

54

LA	17 0223	23556	17 12 02 0262	
LA	17 0224	23557	17 12 01 0233	
J	17 0225	23560	17 06 00 0234	
LA	17 0226	23561	17 12 03 0257	
J	17 0227	23562	17 06 00 0234	
0090	17 0234	23567	-17 10 17 0272	SA 15 GENE
0091	17 0235	23570	00 07 01 0250	SLJ 0250
0092	17 0236	23571		END START
	17 0236	23571	1110140603101314	
	17 0240	23573	03030710	
	17 0241	23574	00777700	
	17 0242	23575	00000002	
	17 0243	23576	22000000	
	17 0244	23577	00777777	
	17 0245	23600	62000000	
	17 0246	23601	4200000000770000	
	17 0250	23603	76510115	
	17 0251	23604	0042000000007700	
	17 0253	23606	75710115	
	17 0254	23607	0000420000000077	
	17 0256	23611	75110115	
	17 0257	23612	00000000	
	17 0260	23613	00030300	
	17 0261	23614	76524115	
	17 0262	23615	00004203	
	17 0263	23616	00037777	
	17 0264	23617	01234567	
	17 0266	23621	76510233	

UNIVAC III UTMOST

REVISION:

1

SECTION:

II

DATE: Oct. 1, 1962

PAGE: 55

17 0267 23622 75710233

17 0270 23623 75110233

17 0271 23624 76524233

17 0272 23625 00037770

15 27333

16 25333

17 23333

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, IR	(4 lists)
ST1	Label	Label	# of words		(3 lists)
IT	0,IR	0,IR	# of words		(3 lists)
ST1	0,IR	0,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.

Line 6 - 8

The opening lines are the definition of the MOVE PROC.

Lines 9 - 12 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".

Lines 13 - 16 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 17 - 20 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 21 - 24 PROC G

This PROC is used 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 25 - 28 PROC J1

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 area addresses were given as labels.

Line 27 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 29 - 32 PROC H

These lines would be generated if straight line coding would be desired and the index register contains the area addresses. The first DO determines if there are any non-multiples of four words and generates a PROC to move them.

Lines 33 - 42 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 43 - 56 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 44

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 45 - 50

are used to manipulate the beginning address and create the proper increment and compare control word for use in iteration.

Lines 51 - 55

comprise the entire coding needed to move four words iterating on

index register given as containing the beginning area address.

Line 56

is the conclusion of a PROC, an END line.

Lines 57 - 60 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 61 - 64 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 65 - 68 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.

Lines 69 - 72 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 73 - 75

The DO statement on lines 73 - 74 test 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.

UNIVAC III UTMOST

REVISION:

2

SECTION:

II

DATE: Oct. 1, 1962

PAGE: 59

0001	00000	RES	011000
0002	17 0000 11000	USE	15
0003	00020000	IN EQU	020000
0004	00030000	OUT EQU	030000
0005	00037777	TEMP EQU	037777
0006		MOVE	PROC
0007		IT	NAME 0
0008		ST1	NAME 1
0009		M	PROC
0010		LA	15, 4*AND-1, MOVE(1,2)
0011		SA	15, 4*AND-1, MOVE(2,2)
0012		END	
0013		L	PROC
0014		LA	L(1,1), MOVE(3,1)-1, MOVE(1,2)
0015		SA	L(1,1), MOVE(3,1)-1, MOVE(2,2)
0016		END	
0017		K	PROC
0018		LA	15, MOVE(1,1)+(4*COUNT-1)
0019		SA	15, MOVE(2,1)+(4*COUNT-1)
0020		END	
0021		G	PROC
0022		LA	G(1,1), MOVE(1,1)+MOVE(3,1)-1
0023		SA	G(1,1), MOVE(2,1)+MOVE(3,1)-1
0024		END	
0025		J1	PROC
0026		DO	MOVE(3,1)**3>0 , G (3**MOVE(3,1)*3-3)/2+4
0027		COUNT	DO MOVE(3,1)/4 , K
0028		END	
0029		H	PROC
0030		DO	MOVE(3,1)**3>0 , I (3**MOVE(3,1)*3-3)/2+4

UNIVAC III UTMOST

REVISION: 2

SECTION:

II

DATE: Oct. 1, 1962

PAGE: 60

0031	ADD DO MOVE(3,1)/4 , M
0032	END
0033	F PROC
0034	LX MOVE(4,1), MOVE(1,1)
0035	LX MOVE(4,2), MOVE(2,1)
0036	LA 15, 3, MOVE(4,1)
0037	SA 15, 3, MOVE(4,2)
0038	IXC MOVE(4,1), ((4*(MOVE(3,1)/4))+MOVE(1,1),4)
0039	IX MOVE(4,2), (4)
0040	JL \$-4
0041	DO MOVE(3,1)**3>0 , 6 (3**MOVE(3,1)*3-3)/2+4
0042	END
0043	E PROC
0044	DO MOVE(3,1)**3>0 , 1 (3**MOVE(3,1)*3-3)/2+4
0045	SX MOVE(1,2), TEMP
0046	LA R, TEMP
0047	BA R,(4*(MOVE(3,1)/4))
0048	BRR R, 16
0049	OR R, (4)
0050	SA R, TEMP
0051	LA 15, 3, MOVE(1,2)
0052	SA 15, 3, MOVE(2,2)
0053	IXC MOVE(1,2), TEMP
0054	IX MOVE(2,2), (4)
0055	JL \$-4
0056	END
0057	D PROC
0058	DO MOVE(3,1)>>0 , F

UNIVAC III UTMOST

REVISION: 1

SECTION:

II

DATE: Oct. 1, 1962

PAGE: 61

0059

0060

0061

0062

0063

0064

0065

0066

0067

0068

0069

0070

0071

0072

0073

0074

0075

0076 17 0000 11000 -17 51 16 0044

LX 17 0001 11001 -17 51 15 0045

LA 17 0002 11002 16 12 17 0003

SA 17 0003 11003 15 10 17 0003

1XC 17 0004 11004 17 53 16 0046

1X 17 0005 11005 17 52 15 0047

JL 17 0006 11006 17 60 05 0002

LA 17 0007 11007 -17 12 05 0050

SA 17 0010 11010 -17 10 05 0051

0077 17 0011 11011 -17 50 16 0052

LA 17 0012 11012 -17 12 10 0052

DO MOVE(3,1)<21 , J1

END

C PROC

DO MOVE(3,1)>20 , E

DO MOVE(3,1)<21 , H

END

B PROC

DO MOVE(1,1)=0 , H

DO MOVE(1,1)>0 , J1

END

A PROC

DO MOVE(1,1)=0 , C

DO MOVE(1,1)>0 , D

END

DO MOVE(0,0)=0 , A

DO MOVE(0,0)=1 , H

END

IT TN OUT 50 14,13

IT 0,14 0,13 400

UNIVAC III UTMOST

REVISION: 1

SECTION: II

DATE: Oct. 1, 1962

PAGE: 62

BA	17 0013	11013	17 24 10 0053
BRR	17 0014	11014	00 44 10 0020
OR	17 0015	11015	17 15 10 0047
SA	17 0016	11016	-17 10 10 0052
LA	17 0017	11017	16 12 17 0003
SA	17 0020	11020	15 10 17 0003
IXC	17 0021	11021	-17 53 16 0052
IX	17 0022	11022	17 52 15 0047
JL	17 0023	11023	17 60 05 0017
0078	17 0024	11024	-17 12 17 0054
SA	17 0025	11025	-17 10 17 0055
LA	17 0026	11026	-17 12 17 0056
SA	17 0027	11027	-17 10 17 0057
LA	17 0030	11030	-17 12 17 0060
SA	17 0031	11031	-17 10 17 0061
LA	17 0032	11032	-17 12 17 0062
SA	17 0033	11033	-17 10 17 0063
LA	17 0034	11034	-17 12 17 0064
SA	17 0035	11035	-17 10 17 0065
0079	17 0036	11036	16 12 05 0011
SA	17 0037	11037	15 10 05 0011
LA	17 0040	11040	16 12 17 0003
SA	17 0041	11041	15 10 17 0003
LA	17 0042	11042	16 12 17 0007
SA	17 0043	11043	15 10 17 0007
0080	17 0044	11044	
	17 0044	11044	00020000
	17 0045	11045	00030000

ST1 IN OUT 20

ST1 0.14 0.13 10

END

UNIVAC III UTMOST

REVISION: 1

SECTION: II

DATE: Oct. 1, 1962

PAGE: 63

17 0046 11046 20060004

17 0047 11047 00000004

17 0050 11050 00020061

17 0051 11051 00030061

17 0052 11052 00037777

17 0053 11053 00000620

17 0054 11054 00020003

17 0055 11055 00030003

17 0056 11056 00020007

17 0057 11057 00030007

17 0060 11060 00020013

17 0061 11061 00030013

17 0062 11062 00020017

17 0063 11063 00030017

17 0064 11064 00020023

17 0065 11065 00030023

17 11000

UNIVAC III UTMOST

REVISION:

SECTION:

Notes

DATE:

July 1, 1962

PAGE:

REVISION:	1	SECTION:	III
DATE:	Feb 1, 1963	PAGE:	1

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 sixteen 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 sixteen 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.

PRIORITY	OPERATOR	DESCRIPTION
1	$a * \uparrow b = a * 10^b$	
1	$a * - b = a * 10^{-b}$	
2	*	Arithmetic Product
2	/	Arithmetic Quotient
2	//	Covered Quotient
3	+	Arithmetic Sum
3	-	Arithmetic Difference
4	**	Logical Product (AND)
5	$\uparrow \uparrow$	Logical Sum (OR)
5	--	Logical Difference (exclusive OR)
6	$=$	Equal $a = b$ is 1 if $a = b$ $a = b$ is 0 if $a \neq b$
6	$>$	Greater Than $a > b$ is 1 if $a > b$ $a > b$ is 0 if $a \leq b$
6	$<$	Less Than $a < b$ is 1 if $a < b$ $a < b$ is 0 if $a \geq b$

UNIVAC III UTMOST

REVISION:

3

SECTION:

III

DATE:

Feb. 1, 1963

PAGE:

3

In the absence of parentheses, rules of priority determine the sequence of operations performed within an expression. When two or more operators of the same priority are used, the sequence of interpretation is from left to right. The levels of priority are shown in the chart on the preceding page. These levels are illustrated in following examples.

$9 - 2 * 3 + 12 ** 6$ the result is 7
 $((9 - (2 * 3)) + 12) ** 6$ the result is 6

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 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.

UNIVAC III UTMOST

REVISION:

SECTION:

III

DATE:

July 1, 1962

PAGE:

4

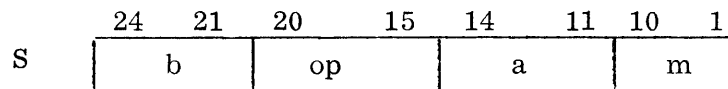
<u>TYPE</u>	<u>FORM</u>	<u>VALUE</u>	<u>EXAMPLE</u>
Label	any label	value assigned to label	L
Location	\$	value of location counter	\$
Octal	the digit 0 followed by octal (0-7) digits	value interpreted as base 8 (binary representation)	017
Decimal	non-zero digit followed by decimal (0-9) digits	value 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:



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).

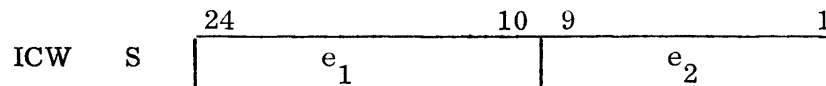
G. DATA WORD GENERATION

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 comparing 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.

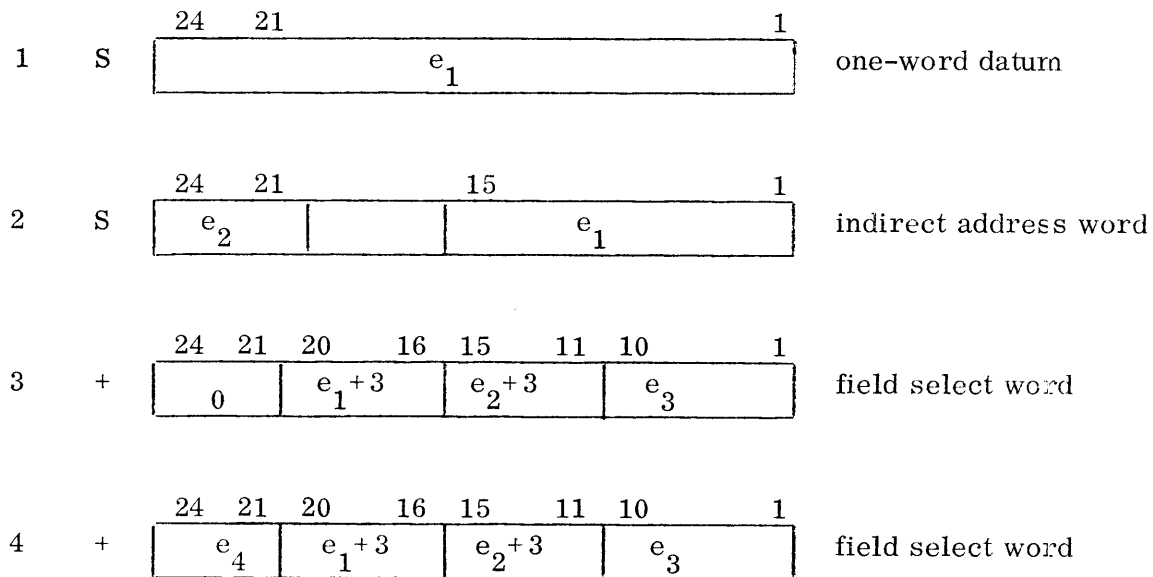
The format of the generated word is illustrated below:



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

2. + or - Operation Field

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:



3. Two Word Constant, TWC

A TWC data generating word will actually generate two words. 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.

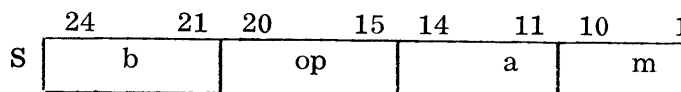
A literal will be double precision if the line was a "TWC" line or if it was a data line with one expression and the mode of the expression was floating.

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).

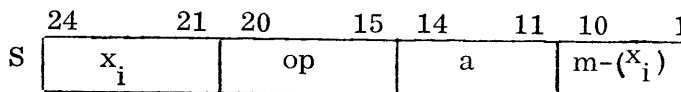
1. $m < 2^{10}$



2. $b = 0$, and
 $m \geq 2^{10}$ and

for some i

$0 \leq m - (x_i) < 2^{10}$



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 $y \pmod{2^{10}}$.

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).

UNIVAC III UTMOST

REVISION:

SECTION:

III

DATE:

July 1, 1962

PAGE:

8

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 $e_1 \dots e_n$;

REFERENCE: label $e_1, e_2, \dots e_n$

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 $n + 1$ 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 Δ , Δ line of coding

9. USE

This directive is followed by not more than 15 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.

UNIVAC III UTMOST

REVISION:

SECTION:

Notes

DATE:

July 1, 1962

PAGE:

UNIVAC III UTMOST

REVISION:

1

SECTION:

IV

DATE:

August 24, 1962

PAGE:

1

UTMOST INTERIM OPERATING PROCEDURES

1. Mount blank tape on logical servo #2.
2. Mount tape to be corrected on logical servo #1. (If there is not a tape to be corrected, a blank should be mounted on servo #1.)
3. Set the printer at absolute line #1 and place blank cards in the punch unit.
4. Place the UTMOST binary card deck in the high speed reader followed by one or more symbolic decks for assembly. Each symbolic deck must be separated from its successor by a blank card.
5. Depress the CLEAR button.
6. Feed one card.
7. Depress the RUN button.
8. After loading UTMOST deck and after completing an assembly, computer stops.
9. Depress RUN button for each assembly.

UNIVAC III UTMOST

REVISION:

SECTION:

Notes

DATE:

July 1, 1962

PAGE:

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

INTRO.

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.

UNIVAC III UTMOST

REVISION:	SECTION: NOTES
DATE: July 1, 1962	PAGE:

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 productivity 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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

August 24, 1962

PAGE:

2

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 arithmetic, 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 $\frac{1}{2}$ 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

UNIVAC III UTMOST

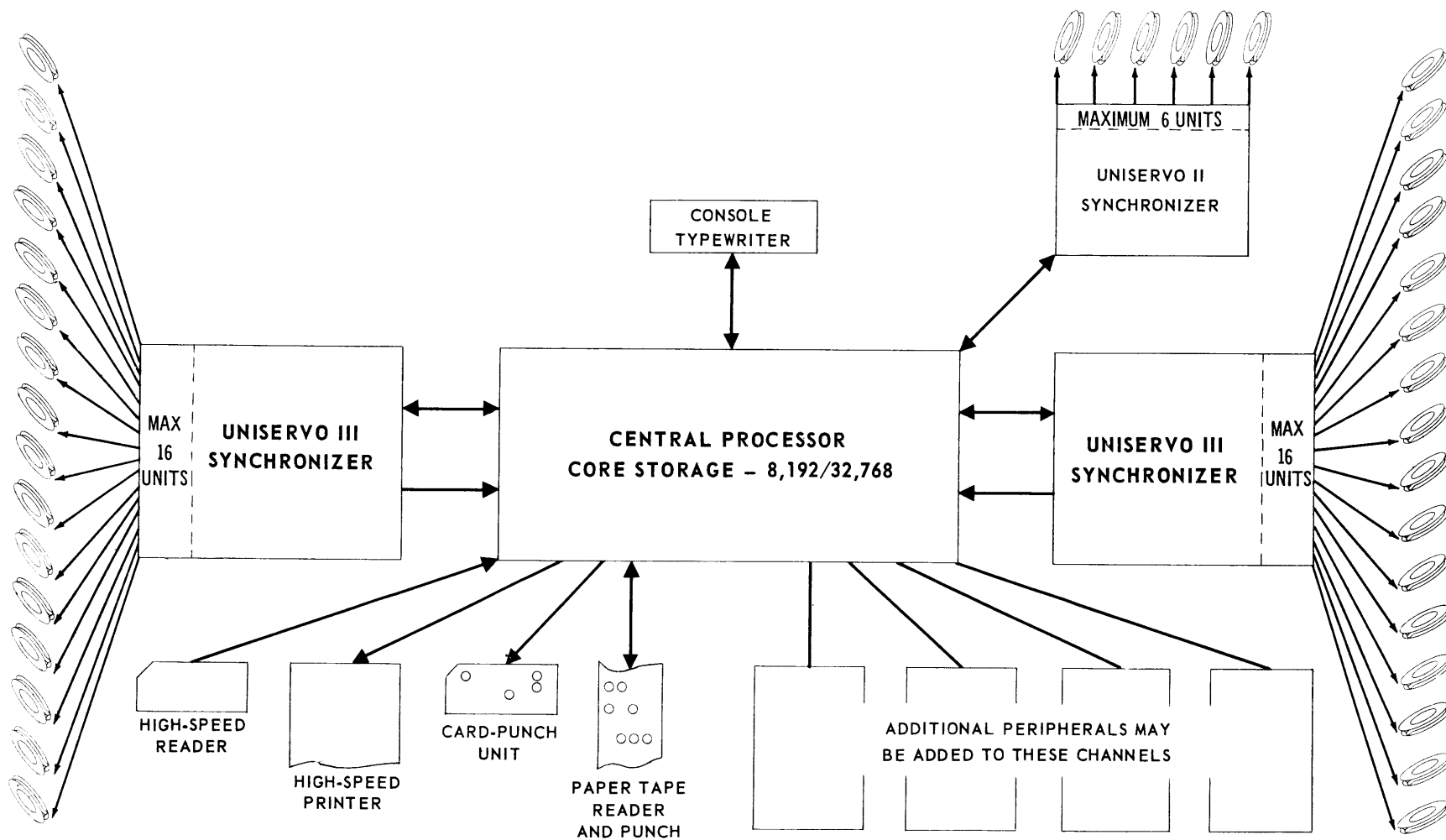


Figure 1-1. Maximum Configuration of the UNIVAC III System

REVISION:	SECTION:
1	V
DATE: August 24, 1962	PAGE: 3

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

August 24, 1962

PAGE:

4

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.

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

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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

5

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	

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

6

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 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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

7

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.

MODULO 3 CHECK BITS (00-01-10)

SIGN			
27	26	25	24
1			

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			it
	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.

S I G N	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.

S I G N	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.

S I G N	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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

10

	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	X0			^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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

11

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.

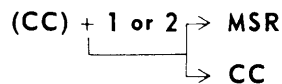
However, if position 25 is a 1-bit, positions 1-10 reflect the unindexed address of either an Indirect Address Control Word or a Field Select Control Word. The original function of positions 1-10 of the basic instruction will in these cases be relegated to the control words.

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.

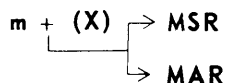
CONTROL COUNTER (CC)

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.



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.

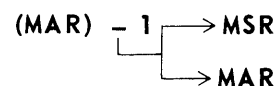


MEMORY SWITCH REGISTER (MSR)

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.



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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

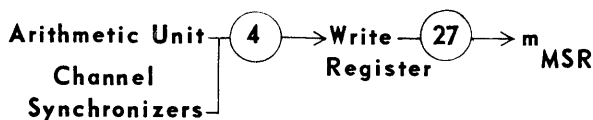
13

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.

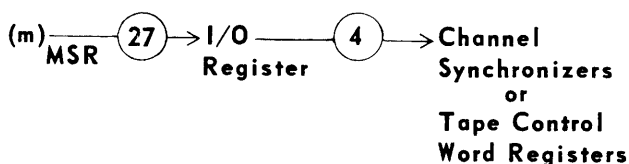
WRITE REGISTER

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

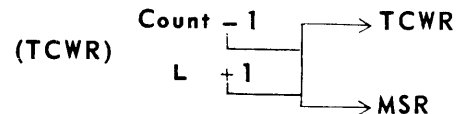
When read from memory, all output, including tape control words and input-output function specifications, pass through this register. Its function is to convert the 27-bit parallel transmission from memory to a 4-bit parallel transmission to the channel synchronizers.



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.

UNIVAC III UTMOST

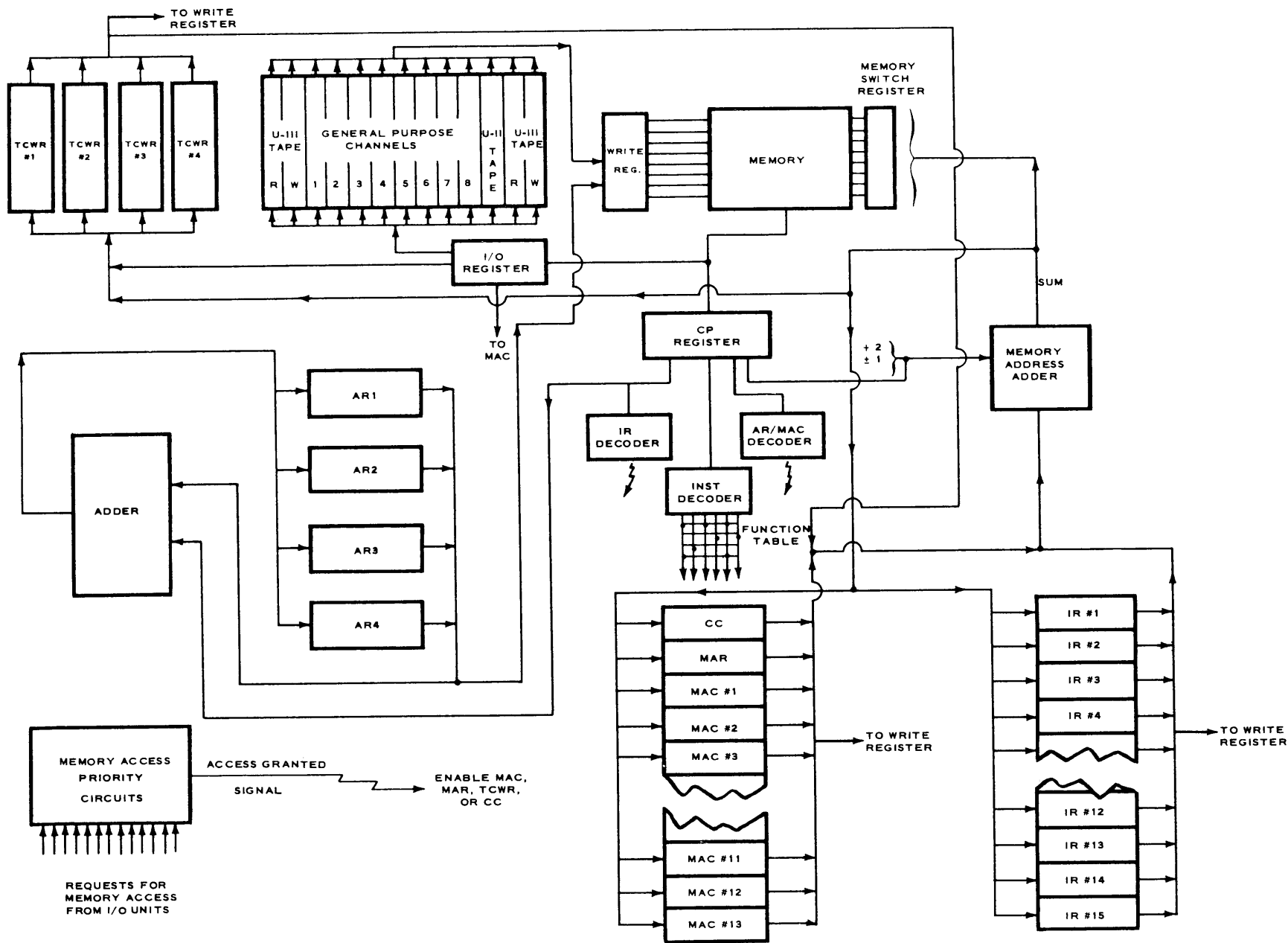


Figure 3-1. UNIVAC III Processor Block Diagram

REVISION:

SECTION:

DATE:

July 1, 1962

PAGE:

14

V

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.

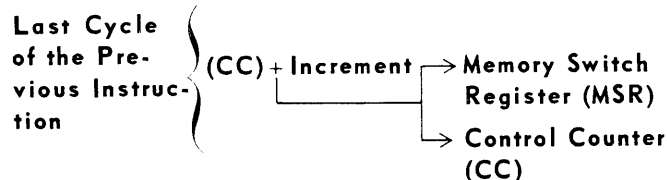
The sequencing of instructions is a function of the Control Counter (CC). The CC contains the memory address of the instruction being executed in a 15-bit binary format.

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.

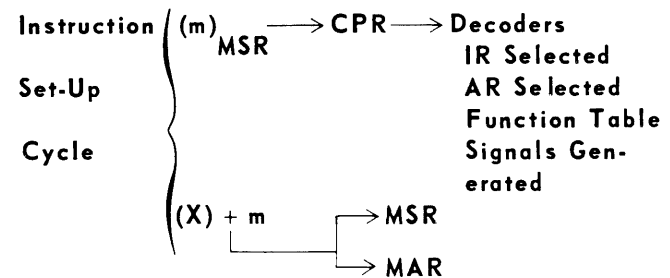


Instruction Set-Up Cycle

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.

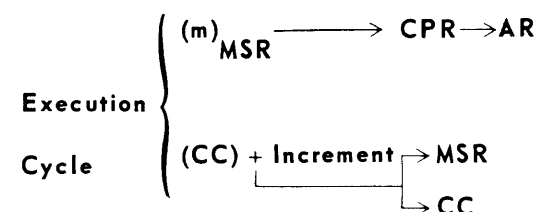
During the latter part of the Instruction Set-Up Cycle, the contents of the index register specified by the instruction, and the memory address (from the CPR) are combined in the Memory Address Adder, and the result is sent to the Memory Switch Register and the Memory Address Register. The MSR, which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



UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

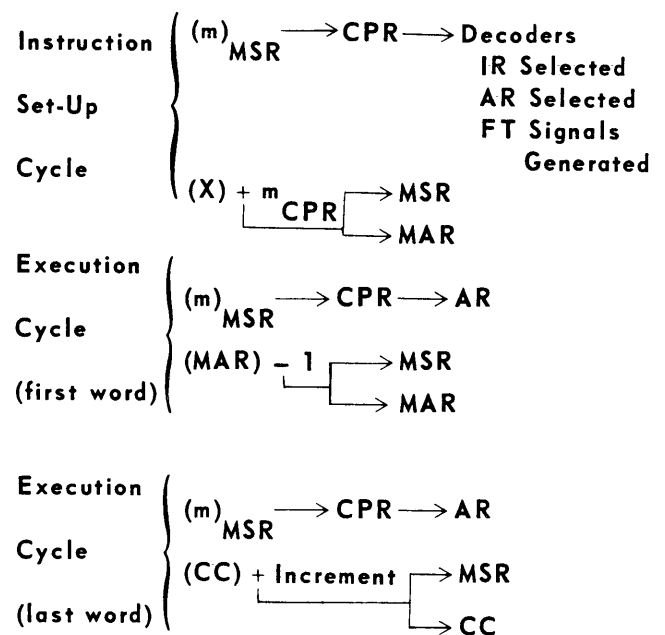
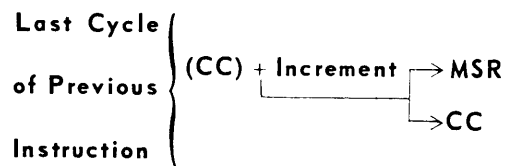
July 1, 1962

PAGE:

16

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.



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	AR 8
0	1	0	0	AR 4
0	0	1	0	AR 2
0	0	0	1	AR 1

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.	L-Addr.
25	24	21	20 18 17 16	15

I/A Indirect address/field selection option

X Binary address of index register, 1 to 15

Bits 18–20 Must be 0's

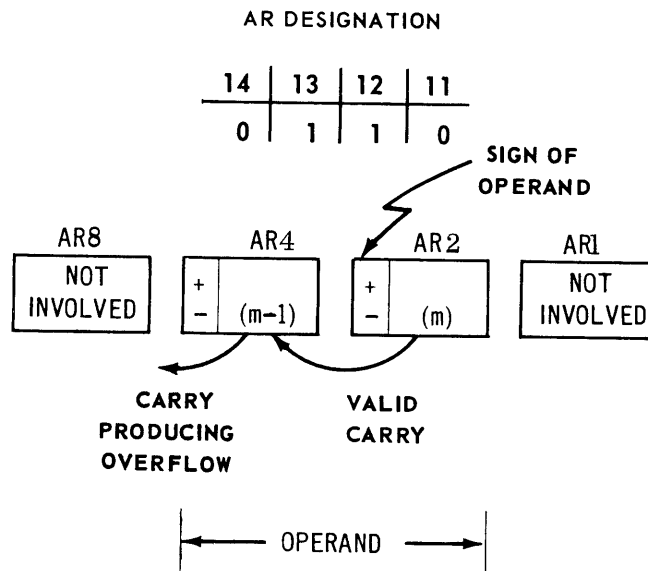
Bits 16–17 Unassigned

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

Adjacent Registers Used



Non-Adjacent Registers Used

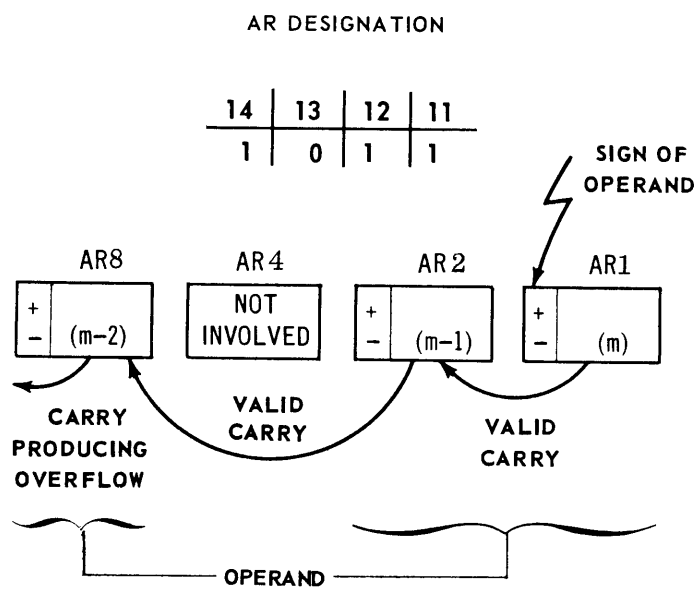


Figure 4-1. Examples of Multi-Word Operands

REVISION:	SECTION: V
DATE: July 1, 1962	PAGE: 20

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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

21

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).

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

22

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

SYMBOLOLOGY AND ABBREVIATIONS USED

()	The contents of
() _x	The contents of, as specified by x
$a \rightarrow b$	a is transferred to b
m	A 10-bit unindexed address
m'	A 15-bit indexed address
AR _i	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	L A
------	-----

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

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

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

August 24, 1962

PAGE:

23

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: $(m) \rightarrow AR_i$
 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	CP 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, *FIELD B* (1002), to AR8 reversing the sign(s) of the operand.

LAN 8, FIELD B				
I/A	X	OP Code	AR	m
0	0000	13	1000	1002

LOAD FIELD INTO REGISTER

LF

Operation: $(m') \xrightarrow{FSEL} AR_i$

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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb, 1, 1963

PAGE:

24

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

SA

Operation: $(ARi) \rightarrow m'$

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 *FIELD B* (0551-0552).

SA 6, *FIELD B*

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

STORE A NEGATIVELY

SAN

Operation: $(ARi) \rightarrow m'$

OP Code: 11

Cycles: 2

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

25

Description: *Transfer the contents of the arithmetic 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 FIELD C (0482) reversing the sign.

SAN 4, FIELD C

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: $(ARi) + (m') \rightarrow AR$

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 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 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.*

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

26

4. See Arithmetic Modes for a discussion of recombination 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: $(ARi) + (m') \rightarrow ARi$ 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

- The addend will be undisturbed.
- 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 recombination and determination of signs.

Illustration

Add *FIELD D* (0585) to *AR8* and place the sum in *AR2*.

DAH		10,		FIELD D	
I/A	X	OP Code	AR	m	
0	0000	22	1010	0585	

DECIMAL SUBTRACT

DS

Operation: $(ARi) - (m') \rightarrow ARi$
 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	

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

27

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 re-complementation and determination of signs.

Illustration

Subtract *FIELDA* (0565) from *AR1*.

DS 1, FIELDA				
I/A	X	OP Code	AR	m
0	0000	21	0001	0565

DECIMAL SUBTRACT HIGHER

DSH

Operation: $(ARi) - (m') \rightarrow ARi'$, 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 re-complementation and determination of signs.

Illustration

Subtract *FIELD5* (0782) from *AR4* placing the difference in *AR2*.

DSH 6, FIELD5				
I/A	X	OP Code	AR	m
0	0000	23	0110	0782

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

28

DECIMAL MULTIPLY

DM

Operation: $(m') \times (AR8) \rightarrow AR4 \text{ and } AR2$
 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			FIELD B	
I / A	X	OP Code	AR	m
0	0000	30	1110	0538

DECIMAL DIVIDE

DD

Operation: $(AR12) \div (m') \rightarrow AR4(\text{quotient})$
 $\rightarrow AR8(\text{remainder})$
 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 FIELD D (0685).

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

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

29

BINARY ADD

BA

Operation: $(ARi) + (m') \rightarrow ARi$
 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 re-complementation and determination of signs.

Illustration

Add in binary, FIELD A (0789) to AR2.

BA		2,	FIELD A	
I / A	X	OP Code	AR	m
0	0000	24	0010	0789

BINARY ADD HIGHER

BAH

Operation: $(ARi) + (m') \rightarrow ARi'$ 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 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 re-complementation and determination of signs.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

30

Illustration:

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

BAH 5, FIELD

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

BINARY SUBTRACT

BS

Operation: $(ARi) - (m') \rightarrow ARi$
 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 re-complementation and determination of signs.

Illustration

Subtract in binary FIELD (0823) from AR2.

BS 2, FIELD

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

BINARY SUBTRACT HIGHER

BSH

Operation: $(ARi) - (m') \rightarrow ARi'$ 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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

31

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 re-complementation 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.

UNIVAC III UTMOST

REVISION: 1

SECTION:

V

DATE: Oct. 1, 1962

PAGE:

32

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 lower 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 higher 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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

33

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 result in 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 two 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

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

34

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 sixteen 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 multi-word operands may be employed.

Illustration

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

CM 2, FIELD A ,				
I/A	X	OP Code	AR	m
0	0000	55	0010	0732

COMPARE		C
Operation:	$(ARi) : (m')$	
OP Code:	54	
Cycles:	2	

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

35

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 $(AR_i) > (m')$, set Greater Comparison Indicator
If $(AR_i) < (m')$, set Less Comparison Indicator
If $(AR_i) = (m')$, 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 AR1 with FIELD A (0835).

C 1, FIELD A				
I/A	X	OP Code	AR	m
0	0000	54	0001	0835

COMPARE PRODUCT WITH A REGISTER

CPA

Operation: (AR_i) 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 AR4 with the 1-bits of FIELD D (0823).

CPA 4, FIELD D				
I/A	X	OP Code	AR	m
0	0000	57	0100	0823

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

36

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 FIELD D (0834).

CPZ		2,		FIELD D
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, $m' \rightarrow CC$.
If reset, $(CC) + 1 \rightarrow CC$
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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

37

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

JUMP IF HIGH				JG
--------------	--	--	--	----

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 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.

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 LESS				JL
--------------	--	--	--	----

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 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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

38

Illustration

Transfer control to LOCB (0938) if the LessComparison 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, $m' \rightarrow CC$
If reset, $(CC) + 1 \rightarrow CC$
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:

AR ⁸	0001	1
AR ⁴	0010	2
AR ²	0011	3
AR ¹	0100	4

3. Indirect addressing may be employed.

Illustration

Transfer control to LOCD (0659) if the sign of AR 2 is positive.

TPOS		3,	LOCD	
I/A	X	OP Code	Indicator	m
0	0000	60	0011	0659

JUMP	J
------	---

Operation: $m' \rightarrow CC$
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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

39

STORE LOCATION AND JUMP
STORE CHANNEL AND JUMP

SLJ
SCJ

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

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

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

40

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 multi-word 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 multi-word 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, $m' \rightarrow CC$
If reset, $(CC) + 1 \rightarrow CC$
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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

41

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 *FIELD B* (0830-0832) from alpha-numeric format to decimal format and locate the result in AR12.

LAD 12, *FIELD B* + 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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

42

- The signs of the result in memory will be that of the least significant word of the operand in the arithmetic registers.
- The contents of the arithmetic registers are not altered.
- 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	

LOAD 'A' EDITED

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

- The operand in memory is unaltered.

- The original sign(s) are retained.

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

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

- Indirect addressing, but not field selection may be employed.

Illustration

Edit *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:

AR _i	
m	(AR _i) before execution
(m')	(AR _i) after execution

OR

OR

Operation: (m') → 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	

UNIVAC III UTMOST

REVISION:	1	SECTION:	V
DATE:	Feb. 1, 1963	PAGE:	43

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	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: $(m') \rightarrow AR_i$
0-bits

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	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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

44

LOAD INDEX REGISTER

LX

Operation: $(m') \rightarrow XO_i$
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: $(XO_i) \rightarrow m'$
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: $(XO_i) + (m') \rightarrow XO_i$
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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

45

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: $(XO_i) + (m') \xrightarrow{\text{bits 1-9}} XO_i$
 $|XO_i| : |(m')| \xrightarrow{\text{bits 10-24}}$

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 $|XO_i| > |(m')|$ bits 10-24, set Greater Comparison Indicator.
- if $|XO_i| < |(m')|$ bits 10-24, set Less Comparison Indicator.
- if $|XO_i| = |(m')|$ bits 10-24, set Equal Comparison Indicator

I / A	X			OP Code		X0		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 ICW

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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

46

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	X0	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

- 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.
- The condition of the indicator(s) will not be affected by the test.
- 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

- The location immediately following the instruction will normally be an unconditional transfer.
- 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		RC
------------------------------	--	----

OP Code: 65

Cycles: 2

Description: Reset the Contingency Indicator(s) specified in bit positions 1–10 of the instruction.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

47

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 re-setting 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

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

48

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

- Any number of indicators may be reset. The inclusion of several 1-bits will result in the resetting of all the indicators designated.
- 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 → CC
If reset, (CC) + 2 → 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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

49

Channel Designator (See below.)

Bit
Positions

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 instruction 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 ReadWrite	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	

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE: Oct. 1, 1962

PAGE:

50

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 Read Channel.

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		

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.

UNIVAC III UTMOST

REVISION: 1

SECTION:

V

DATE: Oct. 1, 1962

PAGE:

51

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

JIP

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				LOCE	
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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

52

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	L-Addr.
25 24 21 20	17 16 15		1

HIGH-SPEED PRINTER FUNCTIONS

0	Number of Lines Paper Advance	FUNCTION CODE	L-Addr.
25 24	19 18 17 16	15	1

HIGH-SPEED CARD READER AND CARD-PUNCH FUNCTIONS

00000	Function Code	L-Addr.
25 21 20	17 16 15	1

PAPER-TAPE READER AND PUNCH FUNCTIONS

0	No. of Words	FUNCTION CODE	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, 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. Resetting 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 synchronizer remains reset. The instruction in its stand-by location will therefore not be transferred for execution.

UNIVAC III UTMOST

REVISION:	1	SECTION:	V
DATE:	Feb. 1, 1963	PAGE:	53

- The address of the memory locations associated with the channel is the binary value of the channel designator.
- Indirect addressing but not field selection may be employed.
- See Note 2 of SC for channel addresses.

Illustration

Initiate a tape operation for the Basic Read Channel. The function specification is located in LOCB (0839).

LC		4,		LOCB	
I/A	X	OP Code	Channel	m	
0	0000	70	0100	0839	

MISCELLANEOUS INSTRUCTIONS

NO OPERATION	NOP
--------------	-----

Operation: $(CC) + 1 \rightarrow CC$
 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

- Memory, arithmetic registers and indicators are not affected.

STORE LOCATION
STORE CHANNEL

SL
SC

Operation: $(MACi) \rightarrow m'$
 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

- Bit positions 16-25 of the indexed location will be binary 0's.
- 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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

54

3. The Memory Address Counter for the channel designated at the time of transfer will contain:

UNISERVO III Unit – 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 LOCB (0839).

SC		4,		LOCB	
I/A	X	OP Code	Channel	m	
0	0000	04	0100	0839	

STORE TAPE CONTROL REGISTER ST

Operation: (TCR_i) → 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, FIELD^D

I/A	X	OP Code	Channel	m
0	0000	05	1000	0832

REVISION:	SECTION: V
DATE: July 1, 1962	PAGE: 55

Description: Replace the contents of the Control Counter with the indexed address of the instruction and stop the arithmetic and control unit.

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

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.*

Stop the arithmetic and control unit. Then resume the program with the instruction located in LOCB (0839).

READ CLOCK	RCK
Operation: (Clock) \rightarrow AR_i	
OP Code: 76	
Cycles: 2	

Description: *Transfer the reading of the clock to the arithmetic register designated.*

I/A	Should be 0
X	Should be 0
AR	Positional designation of arithmetic register
m	Should be 0's

1. If the clock is cycling, one-half second every six seconds, an invalid time is transferred to AR_i and the next instruction in sequence is accessed; $(CC) + 1 \rightarrow CC$.
2. If the clock is not cycling, a valid time is transferred to bit positions 1–20 of AR_i with 21–25 binary 0's and the next instruction in sequence is skipped; $(CC) + 2 \rightarrow CC$.
3. The valid time is expressed in five 4-bit excess-three digits in the following format:

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.*

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

56

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 multi-word 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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

58

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	Λ	&		
	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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

59

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 Contingency Interrupt.

In order to use the keyboard for input, the Activate 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. Depressing 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:	<i>If Typewriter on-line: (m') \rightarrow TBR one character Then print and (CC) + 2 \rightarrow CC If Typewriter off-line: (CC) + 1 \rightarrow CC</i>	
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.*

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

60

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
AR	12 11	12 11	12 11	12 11
BITS	1 1	1 0	0 1	0 0
SALT	3	2	1	0

2. When the character is printed or function performed, the Typewriter Interrupt Indicator is set causing Contingency 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 FIELD B (0683).

WT 3, FIELD B

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) → 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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

61

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
10's complement	7	8	5	6	4

	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

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

63

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 +003212
the sign
of the input
with the greater
absolute value.

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.

UNIVAC III UTMOST

REVISION:

1

SECTION:

V

DATE: Oct. 1, 1962

PAGE:

64

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. Re complementation 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 is thus effected to one of three locations where JUMP to a program may be initiated to determine the exact nature of the interrupt. This

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

66

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**.)

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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

67

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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

68

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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

Feb. 1, 1963

PAGE:

69

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	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 .

$n'_7 = 0$ if $n'_6 < 5$

$n'_7 = 1$ 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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

71

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'_i	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
$\sum T_i =$			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:

$$\begin{array}{r} 2762 \\ 11 \overline{) 2762} \quad 1 = \text{check value} \end{array} \quad \begin{array}{r} 3438 \\ 11 \overline{) 3438} \quad 6 = \text{check value} \end{array}$$

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.

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

72

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.

$$\begin{array}{r} 20 \\ 3 \overline{) 61} \\ \underline{60} \\ 1 \end{array} \quad \begin{array}{l} 1 = \text{check value} \\ 01 = \text{binary check value} \end{array}$$

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	

UNIVAC III UTMOST

REVISION:

SECTION:

V

DATE:

July 1, 1962

PAGE:

73

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-numeric (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.

UNIVAC III UTMOST

REVISION:	SECTION:
DATE: July 1, 1962	Notes PAGE:

UNIVAC III UTMOST

REVISION:

SECTION:

VI

DATE:

PAGE:

July 1, 1962

1

Communications with the executive system (BOSS III) will be specified later.

UNIVAC III UTMOST

REVISION:	SECTION:
DATE: July 1, 1962	Notes PAGE:

UNIVAC III UTMOST

REVISION:

SECTION:

VII

DATE:

July 1, 1962

PAGE:

1

MNEMONIC INSTRUCTIONS

Instruction is type 0 unless an A value is listed

<u>Octal</u> <u>OP</u> <u>Code</u>	<u>A</u> <u>Field</u>	<u>Instructions' Function</u>		<u>Timing</u>
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

UNIVAC III UTMOST

REVISION:

SECTION:

VII

DATE:

July 1, 1962

PAGE:

2

<u>Octal</u> <u>OP</u> <u>Code</u>	<u>A</u> <u>Field</u>	<u>Instructions' Function</u>		<u>Timing</u>
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

UNIVAC III UTMOST

REVISION:

SECTION:

VII

DATE:

July 1, 1962

PAGE:

3

<u>Octal</u> <u>OP</u> <u>Code</u>	<u>A</u> <u>Field</u>	<u>Instructions' Function</u>		<u>Timing</u>
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 indexX	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

UNIVAC III UTMOST

REVISION:

SECTION:

Notes

DATE:

July 1, 1962

PAGE:

UNIVAC III UTMOST

REVISION:

SECTION:

Appendix 1

DATE: Oct. 1, 1962

PAGE: 1

ERROR CODES

FLAG

E
D
U
A
I
L
R
T

✱

DESCRIPTION

Bad expression - *not known*
Duplicate *code*
Undefined
Address - *not known*
Instruction
Too many levels *64 for 128*
Relocation
Truncation - *2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536, 131072, 262144, 524288, 1048576, 2097152, 4194304, 8388608, 16777216, 33554432, 67108864, 134217728, 268435456, 536870912, 1073741824, 2147483648, 4294967296, 8589934592, 17179869184, 34359738368, 68719476736, 137438953472, 274877906944, 549755813888, 1099511627776, 2199023255552, 4398046511104, 8796093022208, 17592186044416, 35184372088832, 70368744177664, 140737488355328, 281474976710656, 562949953421312, 1125899906842624, 2251799813685248, 4503599627370496, 9007199254740992, 18014398509481984, 36028797018963968, 72057594037927936, 144115188075855872, 288230376151711744, 576460752303423488, 1152921504606846976, 2305843009213693952, 4611686018427387904, 9223372036854775808, 18446744073709551616, 36893488147419103232, 73786976294838206464, 147573952589676412928, 295147905179352825856, 590295810358705651712, 1180591620717411303424, 2361183241434822606848, 4722366482869645213696, 9444732965739290427392, 18889465931478580854784, 37778931862957161709568, 75557863725914323419136, 151115727451828646838272, 302231454903657293676544, 604462909807314587353088, 1208925819614629174706176, 2417851639229258349412352, 4835703278458516698824704, 9671406556917033397649408, 19342813113834066795298816, 38685626227668133590597632, 77371252455336267181195264, 154742504910672534362390528, 309485009821345068724781056, 618970019642690137449562112, 1237940039285380274899124224, 2475880078570760549798248448, 4951760157141521099596496896, 9903520314283042199192993792, 19807040628566084398385987584, 39614081257132168796771975168, 79228162514264337593543950336, 158456325028528675187087900672, 316912650057057350374175801344, 633825300114114700748351602688, 1267650600228229401496703205376, 2535301200456458802993406410752, 5070602400912917605986812821504, 10141204801825835211973625643008, 20282409603651670423947251286016, 40564819207303340847894502572032, 81129638414606681695789005144064, 162259276829213363391578010288128, 324518553658426726783156020576256, 649037107316853453566312041152512, 1298074214633706907132624082305024, 2596148429267413814265248164610048, 5192296858534827628530496329220096, 10384593717069655257060992658440192, 20769187434139310514121985316880384, 41538374868278621028243970633760768, 83076749736557242056487941267521536, 166153499473114484112975882535043072, 332306998946228968225951765070086144, 664613997892457936451903530140172288, 1329227995784915872903807060280344576, 2658455991569831745807614120560689152, 5316911983139663491615228241121378304, 10633823966279326983230456482242756608, 21267647932558653966460912964485513216, 42535295865117307932921825928971026432, 85070591730234615865843651857942052864, 170141183460469231731687303715884105728, 340282366920938463463374607431768211456, 680564733841876926926749214863536422912, 1361129467683753853853498429727072845824, 2722258935367507707706996859454145691648, 5444517870735015415413993718908291383296, 10889035741470030830827987437816582766592, 21778071482940061661655974875633165533184, 43556142965880123323311949751266331066368, 87112285931760246646623899502532662132736, 174224571863520493293247799005065324265472, 348449143727040986586495598010130648530944, 696898287454081973172991196020261297061888, 1393796574908163946345982392040522594123776, 2787593149816327892691964784081045188247552, 5575186299632655785383929568162090376495104, 11150372599265311570767859136324180752990208, 22300745198530623141535718272648361505980416, 44601490397061246283071436545296723011960832, 89202980794122492566142873090593446023921664, 178405961588244985132285746181186892047843328, 356811923176489970264571492362373784095686656, 713623846352979940529142984724747568191373312, 1427247692705959881058285969449495136382746624, 2854495385411919762116571938898990272765493248, 5708990770823839524233143877797980545530986496, 11417981541647679048466287755595961091061972992, 22835963083295358096932575511191922182123945984, 45671926166590716193865151022383844364247891968, 91343852333181432387730302044767688728495783936, 182687704666362864775460604089535377456991567872, 365375409332725729550921208179070754913983135744, 730750818665451459101842416358141509827966271488, 1461501637330902918203684832716283019655932542976, 2923003274661805836407369665432566039311865085952, 5846006549323611672814739330865132078623730171904, 11692013098647223345629478661730264157247460343808, 23384026197294446691258957323460528314494920687616, 46768052394588893382517914646921056628989841375232, 93536104789177786765035829293842113257979682750464, 187072209578355573530071658587684226515959365500928, 374144419156711147060143317175368453031918731001856, 748288838313422294120286634350736906063837462003712, 1496577676626844588240573268701473812127674924007424, 2993155353253689176481146537402947624255349848014848, 5986310706507378352962293074805895248510699696029696, 11972621413014756705924586149611790497021399392059392, 23945242826029513411849172299223580994042798784118784, 47890485652059026823698344598447161988085597568237568, 95780971304118053647396689196894323976171195136475136, 191561942608236107294793378393788647952342390272950272, 383123885216472214589586756787577295904684780545900544, 766247770432944429179173513575154591809369561091801088, 1532495540865888858358347027150309183618739122183602176, 3064991081731777716716694054300618367237478244367204352, 6129982163463555433433388108601236734474956488734408704, 12259964326927110866866776217202473468949912977468817408, 24519928653854221733733552434404946937899825954937634816, 49039857307708443467467104868809893875799651909875269632, 98079714615416886934934209737619787751599303819750539264, 196159429230833773869868419475239575503198607639501078528, 392318858461667547739736838950479151006397215279002157056, 784637716923335095479473677900958302012794430558004314112, 1569275433846670190958947355801916604025588861116008628224, 3138550867693340381917894711603833208051177722232017256448, 6277101735386680763835789423207666416102355444464034512896, 12554203470773361527671578846415332832204710888928069025792, 25108406941546723055343157692830665664409421777856138051584, 50216813883093446110686315385661331328818843555712276103168, 100433627766186892221372630771322662657637687111424552206336, 200867255532373784442745261542645325315275374222849104412672, 401734511064747568885490523085290650630550748445698208825344, 803469022129495137770981046170581301261101496891396417650688, 1606938044258990275541962092341162602522202993782792835301376, 3213876088517980551083924184682325205044405987565585670602752, 6427752177035961102167848369364650410088811975131171341205504, 12855504354071922204335696738729300820177623950262342682411008, 25711008708143844408671393477458601640355247900524685364822016, 51422017416287688817342786954917203280710495801049370729644032, 102844034832575377634685573909834406561420991602098741459288064, 205688069665150755269371147819668813122841983204197482918576128, 411376139330301510538742295639337626245683966408394965837152256, 822752278660603021077484591278675252491367932816789931674304512, 1645504557321206042154969182557350504982735865633579863348609024, 3291009114642412084309938365114701009965471731267159726697218048, 6582018229284824168619876730229402019930943462534319453394436096, 13164036458569648337239753460458804039861886925068638906788872192, 26328072917139296674479506920917608079723773850137277813577744384, 52656145834278593348959013841835216159447547700274555627155488768, 105312291668557186697918027683670432318895095400549111254310977536, 210624583337114373395836055367340864637790190801098222508621955072, 421249166674228746791672110734681729275580381602196445017243910144, 842498333348457493583344221469363458551160763204392890034487820288, 1684996666696914987166688442938726917102321526408785780068975640576, 3369993333393829974333376885877453834204643052817571560137951281152, 6739986666787659948666753771754907668409286105635143120275902562304, 13479973333575319897333507543509815336818572211270286240551805124608, 26959946667150639794667015087019630673637144422540572481103610249216, 53919893334301279589334030174039261347274288845081144962207220498432, 107839786668602559178668060348078522694548577690162289924414440996864, 215679573337205118357336120696157045389097155380324579848828881993728, 431359146674410236714672241392314090778194310760649159697657763987456, 862718293348820473429344482784628181556388621521298319395315527974912, 1725436586697640946858688965569256363112777243042596638790631055949824, 3450873173395281893717377931138512726225554486085193277581262111899648, 6901746346790563787434755862277025452451108972170386555162524223799296, 13803492693581127574869511724554050904902217944340773110325048447598592, 27606985387162255149739023449108101809804435888681546220650096895197184, 55213970774324510299478046898216203619608871777363092441300193790394368, 110427941548649020598956093796432407239217743554726184882600387580788736, 220855883097298041197912187592864814478435487109452369765200775161577472, 441711766194596082395824375185729628956870974218904739530401550323154944, 883423532389192164791648750371459257913741948437809479060803100646309888, 1766847064778384329583297500742918515827483896875618958121606201292619776, 3533694129556768659166595001485837031654967793751237916243212402585239552, 7067388259113537318333190002971674063309935587502475832486424805170479104, 14134776518227074636666380005943348126619871175004951664972849610340958208, 28269553036454149273332760011886696253239742350009903329945699220681916416, 56539106072908298546665520023773392506479484700019806659891398441363832832, 113078212145816597093331040047546785012958969400039613319782796882727665664, 226156424291633194186662080095093570025917938800079226639565593765455331328, 452312848583266388373324160190187140051835877600158453279131187530910662656, 904625697166532776746648320380374280103671755200316906558262375061821325312, 1809251394333065553493296640760748560207343510400633813116524750123642650624, 3618502788666131106986593281521497120414687020801267626233049500247285301248, 7237005577332262213973186563042994240829374041602535252466099000494570602496, 14474011154664524427946373126085988481658748083205070504932198000989141204992, 28948022309329048855892746252171976963317496166410141009864396001978282409984, 57896044618658097711785492504343953926634992332820282019728792003956564819968, 115792089237316195423570985008687907853269984665640564039457584007913129639936, 231584178474632390847141970017375815706539969331281128078915168015826259279872, 463168356949264781694283940034751631413079938662562256157830336031652518559744, 926336713898529563388567880069503262826159877325124512315660672063305037119488, 1852673427797059126777135760139006525652319754650249024631321344126610074238976, 3705346855594118253554271520278013051304639509300498049262642688253220148477952, 7410693711188236507108543040556026102609279018600996098525285376506440296955904, 14821387422376473014217086081112052205218558037201992197050570753012880593911808, 29642774844752946028434172162224104410437116074403984394101141506025761187823616, 59285549689505892056868344324448208820874232148807968788202283012051522375647232, 118571099379011784113736688648896417641748464297615937576404566024103044751294464, 237142198758023568227473377297792835283496928595231875152809132048206089502588928, 474284397516047136454946754595585670566993857190463750305618264096412179005177856, 948568795032094272909893509191171341133987714380927500611236528192824358010355712, 1897137590064188545819787018382342682267975428761855001222473056385648716020711424, 3794275180128377091639574036764685364535950857523710002444946112771297432041422848, 7588550360256754183279148073529370729071901715047420004889892225542594864082845696, 1517*

UNIVAC III UTMOST

REVISION:

SECTION:

Appendix 2

DATE:

Feb. 1, 1963

PAGE:

1

BINARY CARD FORMATS

The output of an UTMOST assembly is a deck of 80 column punched cards. The cards are punched in column binary for loading into the UNIVAC III.

In an 80 column untranslated card, a word is 4 columns wide and 6 rows long (4 x 6). Word 1 occupies columns 1-4, rows 12-3; Word 2 rows 4-9; etc. In this fashion, the upper half of a card contains the odd numbered words (1, 3, 5, 7, . . . 39); the lower half of a card contains the even numbered words (2, 4, 6, . . . 40).

The following example illustrates the relationship between a column binary word and its position in memory.

	Word 1				Word 3				Word 5				
Rows													
12	19	13	7	1									
11	20	14	8	2									
0	21	15	9	3									
1	22	16	10	4									
2	23	17	11	5									
3	24	18	12	6									
4													
5													
6													
7													
8													
9													
Cols.	1	2	3	4	5	6	7	8	9	10	11	12	
	Word 2				Word 4				Word 6				

WORD 1 IN MEMORY

S	1	1	0	0	0	1	1	1	1	0	1	0	0	1	0	1	1	0	0	0	1	0	S	0
25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

The Sign Bit of the above example is explained in the discussion of Word 3 under the Instruction and Data Card section which follows.

UNIVAC III UTMOST

REVISION:

SECTION:

Appendix 2

DATE:

Feb. 1, 1963

PAGE:

2

UTMOST produces five types of binary cards: Instruction and Data Cards (the data which the assembler produces--instructions, constants, etc.), Relocation Cards, External Symbol Reference Cards, External Symbol Definition Cards, and End Cards.

An Instruction and Data Card may contain up to and including 24 words to be loaded into memory. (Word 5 thru Word 28)

Word 1 contains the address of the area in memory where the contents of the card are to be stored.

Word 2 is of special format to cause the card to have even parity.

Word 3 contains the sign bits for the data words; a blank for +, a punch for -.

Word 4 is always blank.

Words 5-28 contain the data which will be stored in memory.

The remaining four card formats are explained later in the section.

Col. 1	Col. 2	Col. 3	Col. 4
1 0 0 0 1 1	0 1 1 0 0 1	0 0 0 0 1 0	0 0 0 0 1 0
9 8 7 6 5 4	9 8 7 6 5 4	9 8 7 6 5 4	9 8 7 6 5 4

UNIVAC III UTMOST

REVISION:

SECTION:

Appendix 2

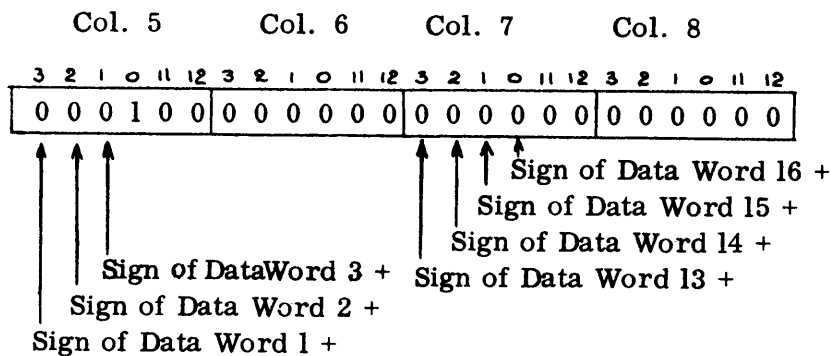
DATE:

Feb. 1, 1963

PAGE:

4

WORD 3



Signs of Data Words:

bit 24 = Sign of Data Word 1

bit 23 = Sign of Data Word 2

.
 .
 .
 .
 etc.

WORD 4 Always Blank

WORDS 5 - 28 The Data Words

	Col. 9				Col. 10				Col. 11				Col. 12												
	3	2	1	0	11	12	3	2	1	0	11	12	3	2	1	0	11	12							
<u>WORD 5</u>	1	0	0	1	0	1	0	1	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	10554

On output listing this is instruction word

Bits

24 - 21 IR $9 = 11_8$

20 - 15 OP $= 24_8 = BA$

14 - 11 AR $= 01_8 = \text{arithmetic register } 1$

10 - 1 10 bit address $= 0476_8$

WORD 6

Col. 9				Col. 10				Col. 11				Col. 12						
9	8	7	6	5	4	9	8	7	6	5	4	9	8	7	6	5	4	
1	0	0	1	0	1	0	1	0	0	0	0	0	1	0	1	1	0	10555

This is an instruction word

UNIVAC III UTMOST

REVISION:

SECTION:

Appendix 2

DATE:

Feb. 1, 1963

PAGE:

5

Bits

24 - 21 IR 9 = 11_8

20 - 15 OP = 24_8 = BA

14 - 11 AR = 01_8 = arithmetic register 1

10 - 1 10 bit address = 0644_8

etc.

UNIVAC III RELOCATABLE BINARY CARD FORMATS

A. INSTRUCTION AND DATA CARD

1. Word 1

Bits 24-23	3
Bits 22-21	0
Bits 20-16	Word count of information
Bits 15-1	Location of information

2. Word 2

Parity word

3. Word 3

Sign bits

4. Word 4

Not used

5. Words 5 thru 28

Binary information to be loaded

B. RELOCATION CARD

1. Word 1

Bits 24-23	3
Bits 22-21	1
Bits 20-16	Number of words to be relocated
Bits 15-1	Irrelevant

2. Word 2

Parity word

3. Word 3

Sign bits	1----Backward adjustment
	0----Forward adjustment

4. Word 4.

Not used

5. Words 5 thru 28

Relocation references

Sign Bits	Direction of adjustment
Bits 24-21	Not used
Bits 20-16	Low order position of 15 bits to be adjusted
Bits 15-1	Location to be relocated (relative to current base)

Each relocation reference will cause location referenced to be adjusted by amount of current base.

Relocation cards must follow corresponding instruction and data card.

C. EXTERNAL SYMBOL REFERENCE CARD

1. Word 1

Bits 24-23	3
Bits 22-21	2
Bits 20-16	Word count of symbol references
Bits 15-1	Irrelevant

2. Word 2

Parity word

3. Word 3

Sign bits

4. Word 4

Not used

5. Words 5 thru 28

External symbols referenced, and corresponding relocation reference.

Each entry consists of two to five words. The last word contains the reference information and is of the same form as a relocation word.

Sign bits	Direction of adjustment
Bits 24-21	Not used
Bits 20-16	Low order bit position of 15 bit reference address
Bits 15-1	Value of adjustment to external value or location

Symbol is given in first words of an entry and may be one to four words long. All but last word of a symbol carry a negative sign.

External symbol references may appear anywhere in a relocatable binary deck, previous to the symbol definition cards. An entry will not overflow from one card to another.

D. EXTERNAL SYMBOL DEFINITION CARDS

1. Word 1

Bits 24-23	3
Bits 22-21	3
Bits 20-16	Word count of symbol definitions
Bits 15-1	Irrelevant

2. Word 2

Parity word

3. Word 3

Sign bits

4. Word 4

Not used

5. Words 5 thru 28

External symbols being defined and corresponding absolute values (25 bits) or locations relative to current base.

Each entry consists of two to five words. The last word specifies the location or value of the external symbol.

Sign bits

Bits 24-17

Not used

Bit 16

Relocation indicator

Bits 15-1

Value or location

If relocation indicator (bit 16) is one then current base should be added to bits 15-1.

Symbol is given in first one to four words of an entry. All but the last word of a symbol carry a negative sign.

External symbol definitions appear at the end of a relocatable binary deck immediately preceding the end card.

E. END CARD

1. Word 1

Bits 24-23

3

Bits 22-16

0

Bits 15-1

End address

2. Word 2

Check sum

3. Word 3

Signs (normally zero)

4. Word 4

Bit 16

If 1 indicates end card has transfer address

Bits 15-1

Location following last word used by program

UNIVAC III UTMOST

REVISION:

SECTION:

Appendix 2

DATE:

Feb. 1, 1963

PAGE:

10

5. Words 5 thru 28

Index Load words

Bits 24-21

Index designation

Bits 15-1

Value to be loaded in index

UTMOST

UNIVAC

DIVISION OF SPERRY RAND CORPORATION