6345
Engineering Note E-516

Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts

SUBJECT: COMPREHENSIVE SYSTEM OF SERVICE ROUTINES

To: S. & E. C. Group and Group 61

From: H. Uchiyamada

Date: December 17, 1952

Abstract: The Comprehensive System of Service Routines provides for conversion by WWI to binary form from Flexowriter-coded perforated tapes prepared according to conventions which have been chosen to facilitate the task of coding programs for WWI. In addition to straightforward conversion of function letters and decimal addresses, the Comprehensive System (CS) provides for (1) use of floating addresses for which assignment of final storage locations is made by the computer (this has the important advantage of permitting insertions and deletions of instructions without extensive renumbering in the program) (2) automatic selection of Input/Output and Programmed Arithmetic (PA) interpretive subroutines which eliminates to a considerable degree the time wasted in handling tapes and the possible errors involved (3) automatic cycle control (patterned after the Manchester B-tube) available within the PA routines which will reduce the need for using uninterpreted WWI instructions within an interpreted program and which will generally facilitate programming (4) the handling of generalized decimal numbers (gdn) of the form $\pm z \times 2^{\gamma i} \times 10^{\delta i}$ which enables the programmer to express numerical data in whatever form is best suited to the particular calculation.

Comment: The author has acted, in the main, as editor of this E-note. Sections have been written by Eric Mutch, John Frankovich, Frank Helwig and Edwin Kopley. The CS as a whole represents the work of many people in the Scientific and Engineering Computation Group. This note is intended as a reference manual, not as an introductory presentation of programming techniques and conventions, which will be available later.

Index for Engineering Note E-516

Table of Contents

I. Introduction

A program is an ordered sequence of words, written with the intention of having it typed on paper tape in the (new) Flexo-code and inserted in WWI by the intermediary of the Comprehensive Conversion Program (CCP).

A word is a finite ordered sequence of syllables. Normally all the syllables forming a word must be separated by a plus or minus sign, but plus signs may be omitted wherever there is no danger of ambiguity. Details of this and other rules governing the assembly of syllables will be given in Section II. Any word made up of one or more syllables must be followed by a terminating character. There are four possible terminating characters giving four possible ways in which the conversion program will treat the word. These terminating characters and their functions will be described in Section II. A given word is meaningful, from the conversion program's viewpoint only if the words, or syllables, respectively, are chosen in a manner not contrary to any of the rules. Any combination of words or syllables not forbidden by the rules will be accepted by the conversion program. Special words will be described later in this section. A single length word is represented in WWI by a 16 binary digit array.

Syllables may be divided into two classes, namely, constant syllables and parametric syllables. The class of constant syllables includes operations, integers and octal numbers. The class of parametric syllables includes preset parameters, relative address, temporary storage, and floating address.

Constant Syllables

Operations are of two kinds, namely, WWI orders and interpreted orders. The WWI operations or orders (ca, cs,---slh, slr, srh, etc.) are described in detail in M-1624. The interpreted orders (ica, ics, etc.) will be found listed with their functions under Section III on Programmed Arithmetic (PA).

Integers may be positive or negative decimal integers or the literal integers, b or c. The decimal integers used are 0, 1, 2,---, 32767 with an implicit factor of 2^{-15} and no decimal point. The literal integers serve a specific purpose which will be described under Section III on PA.

Octal numbers are of the form $d_0 \cdot d_1 d_2 d_3 d_4 d_5$ where d_0, the sign digit, is either 0 or 1 and where $d_1 \ldots d_5$ are the octal digits having one of the values 0, 1, 2, 3, 4, 5, 6, 7. A 1. indicates the start of a negative octal number, the remaining digits being the sevens complement of the absolute magnitude of the number. If an octal number occurs as a syllable in a word, it must always be the first syllable, i.e. only one octal number syllable can occur in any word. An example of a positive octal number is 0.04573. In order to obtain the negative of this number one must change the 0. to 1. and also get the sevens complement of the five octal digits following the sign digit; thus the negative becomes 1.73204.

Parametric Syllables

Preset parameters are of the form $\alpha_1 \alpha_2 \#$, where α_1 is u, p or z depending on whether the parameter is of the type universal (assigned particular meaning and never used for anything else), personal (can be used by anyone to mean anything desired) or subroutine (for parameter in subroutines) respectively; α_2 is any letter of the alphabet except o and 1; and # is any decimal number of the form 1, 2, 3, ..., 255, with initial zero suppressed.

A relative address is one which is used for writing instructions within a subroutine or within any block of instructions with addresses relative to the start of the block (that is, as if the block started in register zero). Such relative addresses are obtained by including an "r" in the address of the instruction, e.g. ca 35r (which consists of the three syllables ca + 35 + r).

The single lower case letter "t" indicates the zero-th register of a block of temporary storage. Its value must be assigned in the same way as for a preset parameter. See Section II on preset parameters.

A floating address is one which enables a programmer to write his instructions so that they refer to the words of his program and not to the locations of those words in storage.

Special Words[+]

The following are different groups of special words:

Program title words: TAPE, MOD, PARAMETER, suffixed by additional information
Output words:* TOA, FOR etc. (See Section IV under Input/Output) perhaps suffixed by additional information and perhaps preceded by an i
Number system indicators: MULTIPLE, SINGLE, (m,n) where m and n are integral numbers. For details see Section III under P.A.
Entry to and exit from PA routines: IN* OUT*
Word called a fence: |||...||| (i.e. 25 vertical bars)
Words: DITTO, START AT, i START AT, the last two of which are suffixed by the starting address
Denial of need for a PA interpretive routine: NOTPA
Special words which are ignored: LSR (library of subroutines), END OF SUBROUTINE

The number system (m,n) indicates a number, m-binary digits long with n the number of binary digits in the exponent of 2, and the number is of the form $Z = x \cdot 2^y$ where x is a m binary digit number and y is a n binary digit number. A single length number with a fixed point would be a (15,0) number. An example of a single length floating point number would be (15,15). An example of a multiple-length number with a fixed point would be (30,0). An example of a multiple-length number with a floating point is (30-j,j) where $1 \leq j \leq 14$. For a detailed description of multiple length numbers see Section III on PA. Single length numbers with fixed point are adequately handled by the WWI operations. Multiple-length and single length floating point numbers are handled by the interpretive operations for which see Section III on PA.

+ All special Words must be terminated by a tab or carriage return.

* Only these Special Words occupy storage registers.

II. Programming

Terminating Characters and their Functions

Any word made up of one or more syllables must be followed by a terminating character. There are four possible terminating characters giving four possible ways in which the conversion program will treat the word.

Tab (--→|) or Carriage Return (↲)	=	"word to be stored" indicator. This causes the word to be stored in the register determined by the current address indicator, unless the word is <u>preceded</u> by an equals sign, for which see below.
Vertical bar (|)	=	address assignment indicator. This causes the current address indicator to be set to the value corresponding to the preceding word. Thus the following word to be stored will be placed in the register thus indicated regardless of consecutivity.
Comma (,)	=	floating and/or relative address assignment indicator (see paragraphs on relative and floating addresses below)
Equals sign (=)	=	parameter assignment indicator. This causes the parameter immediately preceding the equals sign to be set to the value following it (which will be terminated by <u>a</u> tab or carriage return). If no word follows the equals sign (i.e. if the next symbol is a tab or carriage return) the parameter will be assigned the value zero.

Absolute Addresses

At the start of a program and at any point thereafter a decimal integer followed by a vertical bar (e.g. 96|) indicates the location into which the next word is to be placed. In the absence of any further indication words will be stored consecutively; in the absence of an initial indication words will be stored consecutively starting in register 32 (decimal). Note that this conversion program does <u>not</u> permit the use of octal <u>addresses</u>.

Relative Addresses

Instructions within a subroutine or within any block of instructions may be written with addresses relative to the start of the block. Such relative addresses are obtained by including an "r" in the address. This causes the content of a special register known as the relative address indicator (r.a.i) to be added to the instruction during conversion. The r.a.i. may be set at the beginning of the block by the symbols 0r, which cause it to be set to the value of the current address--i.e. the address into which the next word will be put. If an integer n precedes the letter r instead of the zero the r.a.i. will be set to a value equal to the current address minus the integer n; e.g. if the current address is 90 the symbols 5r, will set the r.a.i. to 85. Note that a comma following a floating address will also set the r.a.i. (For details see the following paragraph on floating addresses.) The current address indicator may be set to a desired relative value at any point in a program by punching that value followed by the letter r and a vertical bar; e.g. 35r| will cause the next word to be stored in 35r regardless of consecutivity.

Floating Addresses

As already stated a floating address system is designed to enable a programmer to write his instructions so that they refer to the words of his program and not to the locations of those words in storage.

For example, consider the following set of instructions with fixed addresses:

```
32| ca 41
33| ad 100
34| ts 41
35| ca 42
36| ad 100
37| ts 42
38| ca 43
39| ad 100
40| ts 43
41| ca 101
42| mr 102
43| ts 103
44| cp 32
```

Seven of these instructions refer to the locations of other instructions within the group. If any instructions (or words) were to be added to or deleted from this set, a considerable amount of renumbering would be necessary, in general. A floating address system removes the need for this, by labelling each word to which reference is made by a floating address label. The floating address is of the form $\alpha\#$, where α is any lower-case letter of the alphabet except o and l, and where # is any integer of the form 1, 2, 3,..., 255 with initial zeros suppressed. This floating address, without the comma, is then used as the address section of any instruction which is to refer to the word so labelled. Thus the above program might become:

```
f3, ca m9
    ad 100
    ts m9
    ca h5
    ad 100
    ts h5
    ca b2
    ad 100
    ts b2
m9, ca 101
h5, mr 102
b2, ts 103
    cp f3
```

Note that floating addresses may be used in any order and that words referring to a floating address may occur either earlier or later than the word labelled by the corresponding floating address. Thus insertions into or deletions from such a program may be made without any renumbering or any alterations to the existing words.

The current address is the address of the register into which the next word will go. If the next word occupies several registers, then this is the address of the first register of the word. When a floating address is read, the conversion program records it, together with the current address, as an entry in a special table. The word following is then stored away in the normal manner--i.e. in the location specified by the current address. At a later stage in the conversion--when all the information to be converted has been read--all words referring to floating addresses have added to them the relevant entries from this table. The letter and number(s) forming a floating address may be chosen at will (within the limits already set on floating address labels) but care must be taken that the sum over all letters of the maximum numbers used for each letter does not exceed 255 (e.g. if only floating addresses a3, a17, d9, x31, x100 and z5 were used in a given program, the condition would be satisfied because 17 + 9 + 100 + 5 = 131 < 256). The comma following a floating address serves also as a reference for relative addresses which follow, by setting the relative address indicator to the value of the current address indicator (c.a.i.).

Examples

```
(Absolute address)--->  34|  ca g7 <--- (floating)
(Floating address)--->  g7,  sp 73 <--- (absolute)
                             ts 2r <--- (relative)
(Relative address)--->  4r|  si 7a2<--- (floating)
(Floating address)--->  a2,  + 3
                             -.0055
```

The words in this example would be converted to:

```
34| ca 35
35| sp 73
36| ts 37
37|  --        (contains +0)
38|  --        (contains +0)
39| si 47
40| +3
41| -.0055
```

A word not itself labelled by a floating address may be referred to in floating address fashion relative to a preceding or following word which has a floating address. Thus the word "si 7a2" in the above example refers to the seventh word after the word with the floating address "a2". The same word could be referred to by the floating address 12g7. It is of interest to note in this respect that a2 = 5g7 and g7 = -5a2 (+ is implicit between -5 and a2). Note that no additions or deletions may be made between a word referred to by such means and the word carrying the floating address without a certain amount of renumbering. Corrections may be made to words already labelled by floating addresses, and to the words following them, by preceding each corrected word by the relevant floating address terminated by a vertical bar instead of by a comma, e.g.

```
 g7| sp 72
1a2| -.0065
```

would amend the second and last orders of the above example.

Preset Parameters

The three classes of preset parameters, (universal, personal, and subroutine) have already been mentioned in the introduction. A preset parameter consists of two lower case letters followed by a decimal integer less than 256 but greater than zero. The second letter may be any letter other than o or 1. The first letter is used to distinguish the three classes of parameters; i.e. u for universal parameter, p for personal parameter and z for subroutine parameter. (Note that the letter s could not be used to indicate a subroutine parameter owing to the fact that the conversion program would confuse an sa parameter with an sa WWI operation, etc.) Care must be taken that the sum over all parameter letter pairs of the maximum numbers used for each letter pair must not exceed 40 (e.g. if only parameters pa2, za5, za7, pd7, zg4, ug6, ug8 and zz11 were used in a given program, the condition would be satisfied because $2+7+7+4+8+11 = 39<41$. If the single lower case letter "t" were used anywhere in the program one more would have to be added to the sum which must not exceed 40. In the example given above if a "t" was used anywhere in the program the condition would still be satisfied since $39+1 = 40<41$). A value may be assigned to a preset parameter by a word consisting of the parameter followed by an equals sign and the value to be assigned terminated by a tab or carriage return. After assignment any number of parameters may be added to or subtracted from any word. Preset parameters may be assigned values which depend on other preset parameters. They may also be assigned values which depend on floating addresses. Subroutine library tapes will begin with the symbols ↑LSR↓ followed by the catalog number and the title of the subroutine. After the title the various parameters needed by the subroutine will be listed, each followed by an equals sign, a stop character and a tab or carriage return. Thus, when copying a library tape onto a program tape, parameter values may be inserted by hand each time the Flexowriter stops. If the value of any parameter is zero, nothing need be inserted and it is only necessary to restart the Flexowriter.

Examples, illustrating point made above on preset parameters, follow:

um3=+3	universal parameter
ca71+um3	word becomes ca74
pm3=0.00020	personal parameter
sℓr+pm3	word becomes sℓr16
zm3=rs0	subroutine parameter
zs2=pm3+um3	subroutine parameter
cs7-zm3	word becomes ca7
sℓrzs2	word becomes sℓr19

The value of a temporary storage parameter is assigned in the same way as for a preset parameter; e.g. t=190 or t=pn3 or t=f3. To refer to a temporary storage register in an instruction, the fourth for example, the symbols 3t are used; e.g. ca3t.

<u>Rules for forming words out of syllables</u>

1) No other syllable may occur in a generalized decimal number but the generalized decimal number and the terminating character. A <u>generalized decimal number</u> is of the form $\pm d_1 d_2 --- d_k . d_{k+1} --- d_m x2^{\gamma_i} x10^{\delta_i}$ where $0 \leq k \leq m \leq 18$ and γ_i and δ_i are integers, signed if negative, otherwise not signed, and such that the final number is restricted by the number system indicated by the programmer.

2) Only one octal number syllable can occur in any word, i.e. the octal number syllable must always be the first syllable.

3) A word, address assignment, parameter assignment, or floating address assignment can be found by the sum formed by "special add" of successive syllables contained in them. (Note that the value thus obtained depends upon the sequence in which the syllables are written.)

4) A plus or minus sign preceding a syllable, indicates that the value of the syllable is to be multiplied by +1 or -1 respectively before being added into the word value.

5) A plus or minus sign should always be used when there is an ambiguity in the meaning of a syllable or pair of syllables. (For examples of ambiguities see list of ambiguities.)

6) Rules concerning the use of single letters:

   i) t is considered exactly as a preset parameter, and is usually used to indicate the zero-th register of a block of temporary storage registers.

   ii) b has the value of the address of the buffer storage register in PA routines.

   iii) c adds a value to the word sufficient to change an interpreted instruction into a cycle count interpreted instruction and should be used only with its, iex, ica, ics, iad, isa, imr, idv, isp.

   iv) r is the relative address and is given a value each time a comma occurs.

   r = current address - stem.

   A word containing the terminal character "," and at most one floating address syllable and one integer syllable, is called a <u>floating address assignment</u>, e.g. "7g9,". The stem of a floating address assignment or parameter assignment is the integer (if it exists) which precedes the lower case letter in the floating address assignment or parameter assignment. In the example above (7g9) 7 is the stem.

7) Whenever a "," occurs, the floating address in the floating address assignment is set equal to the current address less the stem.

8) Whenever an "=" occurs, the parameter in the parameter assignment is set equal to the following word less the stem.

9) A starting address word consists of a START AT or i START AT, suffixed by any word, i.e. the starting address including a tab or carriage return.

Rules for forming a program out of words

1) A fence (at least 25 vertical bars) must occur initially and terminally in a program.

2) The initial word of a program will go into the initial register of storage (i.e. register 32) and successive words will go into successive registers unless an address assignment is made. An address assignment consists of constant (except octal numbers) and/or parametric syllable(s) followed by a vertical bar. A definite address is one where the value is explicitly known. An indefinite address is one which depends upon floating addresses or parameters, i.e. only implicitly known. The current address is said to be indefinite, following an indefinite address assignment, and is said to be definite as soon as a definite address assignment is made; but is called indefinite again if another indefinite address assignment is made. If an address assignment (definite or indefinite) is made, the word following such an address will go into the register indicated by the address assignment. (Note that in the case of a definite address assignment the current address is given directly, whereas in the case of an indefinite address assignment the current address may be found indirectly). If no initial address assignment is made, the current address is considered to be definite.

3) No floating address assignments may be made while the current address is indefinite.

4) The special word "i START AT" must occur just before the last word.

5) Titular special words usually occur immediately after the initial fence, but may occur anywhere.

6) A fence must occur both before and after any output or titular special word.

7) A word containing the terminal character "=" and at most one parameter syllable and one integer syllable, is called a parameter assignment, e.g. "5pc10=". The word following a parameter assignment, less the stem of the parameter assignment, is the value given to the indicated parameter. For example, if the word following the above parameter assignment is 7 (i.e. 5pc10=7), then pc10=7-5, which says in effect that the parameter is assigned the value 2.

8) A generalized decimal number will be converted to the number system indicated by the last preceding number system indicator, i.e. SINGLE means that the number will be converted to the (15,0) system, and MULTIPLE to the system determined by the preceding (m,n), otherwise to (15,0).

9) Words occupy one register of storage, generalized decimal numbers (m+n)/15 registers, output special words and IN and OUT one register each. No other kinds of words occupy any register of storage.

10) The special word DITTO, followed by a tab (--→|) or carriage return (⤶) must be preceded by a word or generalized decimal number and followed by an address assignment. The word or generalized decimal number preceding the DITTO will be ditto'd up to but not including the address

indicated; e.g. 131| DITTO↲would cause the word or generalized decimal number preceding DITTO, to be stored in the registers up to but not including 131.

11) The special words LSR--, END OF SUBROUTINE--, OCTAL--, and DECIMAL--, including all words that follow these special words up to tab or carriage return are ignored by the conversion program. (One result is that octal addresses are not permitted).

12) A parameter must be assigned before it is used.

List of common ambiguities

cℓc	vs	clc,	write cl+c	if the floating address cl is meant
sℓc	vs	slc,	write sl+c	" " " " sl " "
aol	vs	a01,	write al	i.e. initial zeros must be suppressed
bol	vs	b01,	write bl	
sol	vs	s01,	write sl	

Some ambiguities of the conversion program are not obvious to the programmer. In particular, single letters may not be written without preceding and following it by a plus or minus sign; e.g. not tca, but ±t±ca
not imrtc, but imr±t±c
To avoid difficulties always use a + between two syllables. The + may always be omitted between function letters and the next syllable.

III. Programmed Arithmetic

Number Systems and Preliminary Definitions

In the following discussion we shall frequently refer to (m,n) numbers where (m,n) = (30,0) or (15,15) or (30-j,j), j = 1, 2,..., 14. We now define these numbers.

(i) A (30,0) number is a 30 digit binary number with the binary point at the left-hand end of the number. Such numbers are stored in two consecutive registers, say q and q+1, with the most significant part of the number being contained in register q. We shall refer to this number as "the (30,0) number contained in "location" q."

(ii) A (15,15) number is a number which has been expressed in the form

$$z = x \cdot 2^y$$

where x is a 15 binary digit number such that $1/2 \leq x < 1$ or $x=0$ and y is a 15 binary digit integer. Such numbers are stored in two consecutive registers, say q and q+1. The number x is stored in register q and the number y is stored in register q+1. We shall refer to this number as the (15,15) number contained in "location" q.

(iii) A (30-j,j), j = 1, 2,..., 14, number is a number which has been expressed in the form

$$z = x \cdot 2^y$$

where x is a 30-j binary digit number such that $1/2 \leq x < 1$ or $x=0$ and y is a j-digit binary integer. Such numbers are stored in two consecutive registers, say q and q+1. The 15 most significant digits of x are stored in register q and the 15-j least significant digits of x are stored in the right-hand 15-j digits of register q+1. The integer y is stored in the left-hand j digits of register q+1. The sign digit of register q+1 refers to the sign of y. We shall refer to this number as the (30-j,j) number contained in "location" q.

On the basis of the above definitions it should be noted that ordinary calculations on WWI are in the (15,0) number system. (30,0) and (15,0) numbers shall be referred to as fixed (binary) point numbers. (15,15) and (30-j,j) numbers shall be referred to as floating (binary) point numbers.

Interpretive Subroutines

(m,n) interpretive subroutines shall mean a particular group of coded programs whose purpose is to facilitate computation using (m,n) numbers. These enable the programmer to write coded programs using (m,n) numbers which are in many ways analogous to ordinary WWI coded (15,0) programs. Such programs, when called into action, take "interpreted instructions" (more strictly, program parameters written as instructions) one at a time from consecuitve storage registers and perform the designated single address operations as defined by the interpreted instruction code. (For a complete list of interpretive operations and their functions see end of Section III.)

A multiple register accumulator (MRA) is used in place of the AC in many interpreted instructions. The MRA is not a special register as is the AC, but rather is a group of 3 ordinary storage registers contained within the interpretive subroutine.

Entry to and Exit from Interpretive Subroutines

Entry to the interpretive subroutine is accomplished by means of the (15,0) word IN. This word is changed into a (15,0) sp instruction by the CS which transfers control from the program to the proper register in the interpretive subroutine. The instructions following the word IN are then performed as interpreted instructions, e.g.

```
32| IN
33| ica50 } This program forms the sum of the (m,n) numbers in
34| iad52 } locations 50 and 52
..  ...
```

Exit from the interpretive subroutine is accomplished by means of an interpreted instruction sp. In this particular case the interpreted instruction, sp, and the (15,0) WWI instruction, sp, have the same binary value. As an example we have

```
32| IN
33| ica50
34| iad52
35| sp 60  }
..  ...    }
60| ca 100 } (15,0) WWI operation is resumed at register 60
..  ...
```

Since it is frequently desired to resume (15,0) WWI operation at the register following the interpreted sp the special word OUT has been included in the conversion vocabulary. If p is the register containing the word OUT, then the special word is converted to an sp(p+1). The previous example can now be written as

```
32| IN
33| ica50
34| iad60
35| OUT
36| ca 100  (15,0) WWI operation is resumed at register 36.
```

Generalized Decimal Numbers

Several words are included in the CS to facilitate the insertion of (m,n) numbers into the computer.

The most general decimal number which can be converted and stored by the CS has the form

$$\pm d_1d_2\ldots d_k.d_{k+1}\ldots d_m \times 2^{\gamma} \times 10^{\delta}$$

Such numbers are first converted by the CCP into the integer

$$\pm d_1d_2\ldots d_k\ldots d_m.$$

The associated exponent of 2 is γ and the associated exponent of 10 is $\delta - m + k$. This result is then further processed in accordance with the last special word (m,n) which appears in the program. This special word causes the conversion program to convert all subsequent generalized decimal numbers into (m,n) numbers unless it is superseded by another special word (m_2,n_2). In the case of (30,0), (15,15) and (30-j,j) numbers the components of the number are stored in consecutive registers. The special word (15,0) gives us of course a single register number.

As an example, to store the (24,6) numbers 2 and 5 in consecutive locations write

```
         (24,6)
      32| +2.
      34| +5.    but 34| is not necessary.
```

It should be emphasized here that all generalized decimal numbers must contain at least a sign and a decimal point.

Two applications of the special word (m,n) are handled by the use of further special words.

The first of these is the special word SINGLE. All generalized decimal numbers, converted after this word appears in a program, are converted to (15,0) numbers.

The second of these is the special word MULTIPLE. All generalized decimal numbers, converted after this word appears in a program, are converted to (m_1,n_1) numbers, where (m_1,n_1) is the last special word (m,n) which appears on the tape. It should be noted that the word MULTIPLE in a tape will be redundant unless the special word SINGLE occurs between it and the last (m,n) on tape.

An example of the use of these words is

```
(24,6)
 32| +2.  }
 34| +5.  } Converted as (24,6) numbers
SINGLE
 36| +.2  }
 37| +.5  } Converted as (15,0) numbers
MULTIPLE
 38| +2.  }
 40| +5.  } Converted as (24,6) numbers
```

It is assumed for the most part that a generalized decimal number is of a magnitude commensurate with the number system into which it is being converted. If the number is not commensurate with the number system, an alarm may occur or an incorrect number occurs.

Cycle Control

The cycle control block of an interpretive subroutine is designed to facilitate the writing of cyclic programs and to permit a certain amount of "red tape" to be handled in the interpretive mode.

The heart of the cycle control block is the cycle control register pair. This is actually two storage registers located in the interpretive subroutine. These registers are called the index register, whose contents is "a," and the comparison register, whose contents is "b."

The following interpreted instructions are now defined:

Int. Inst.	Function
icr m (cycle reset)	Set the index register to +0 and the comparison register to +m.
ict y (cycle count)	Increase the index register by one and form the quantity $\big| \, |a+1| - |b| \, \big| - 0$. If the quantity is $>$ 0 interpret next the instruction in register y. If the quantity is = -0, ignore this instruction and reset the index register to +0.

If now one of the interpreted instructions ca, cs, ad, su, mr, dv, ts, ex, sp is written in the form

ixy 100 c or ixy 100 + c

the interpretive subroutine first forms the instruction

ixy (100 + 2a)

and then executes this instruction. The quantity 100 + 2a is formed instead of 100 + a since we deal mainly with arithmetic operations on 2 register numbers.

This procedure is best explained by a simple example. Suppose we wish to transfer the (24,6) numbers in 100, 102, 104, and 106 to registers 200, 202, 204, 206. We could then write

```
32| icr 4      Set up for four cycles
33| ica 100 c  Pick up C(100 + 2a)    a = 0, 1, 2, 3
34| its 200 c  Store in 200 + 2a      a = 0, 1, 2, 3
35| ict 33     Go thru 4 cycles.
```

Since it will not always be desired to operate on (m,n) numbers stored in consecutive locations we now define the following interpretive instructions

Int. Inst.	Function
ici m (cycle increase)	Increase the contents of the index register by +m
icd m (cycle decrease)	Decrease the contents of the index register by +m

As an example of the use of the ici, let us write a program which transfers the (24,6) numbers in registers 100, 104, 108 and 112 into registers 200, 204, 208, and 212. We have

```
32| icr 8      Set up for 4 cycles
33| ica 100 c  Pick up C(100 + 2a)   a = 0, 2, 4, 6
34| its 200 c  Store in 200 + 2a     a = 0, 2, 4, 6
35| ici 1      Increase contents of index register by 1
36| ict 33     Go thru 4 cycles
```

Since most programs usually contain cycles within cycles, the following interpretive instruction, which effectively permits one to use more than one cycle control register pair, is added to our code to enable these more complicated programs to be treated effectively.

Int. Inst.	Function
icx y (cycle exchange)	Exchange the contents of the index register with C(y) and exchange the contents of the comparison register with C(y+1)

To illustrate the use of this instruction, suppose that it is desired to form four scalar products. There are two sets, each with four four-dimensional vectors. The coefficients of each vector are (24,6) numbers. The coefficients of the first set of four vectors are stored in four blocks whose addresses start with 100, 108, 116 and 124. The coefficients of each vector are stored in one block. The coefficients of the second set of four vectors are stored in four blocks whose addresses start with 200, 208, 216 and 224. Scalar products will be formed with the first vector of the first set and the first vector of the second set; the second vector of the first set and the second vector of the second set; etc. It is desired to store the results of the four scalar products in a block starting with address 300. Register 400 is a register used to store the temporary sum. The instructions are as follows:

```
32| icr 16  }
33| icx 60  }    Set up for 16 cycles
34| icr 4   }
35| icx 70  }    Set up for 4 cycles
36| icr 4        Set up for 4 cycles
37| ica 51  }
38| its 400 }    Clears register 400
39| icx 60
40| ica 100 c }   a = 0, 1, 2, 3
41| imr 200 c }
42| iad 400   }  Forms scalar product
43| its 400   }
44| ici 1        Increase index register by 1
45| icx 60
46| ict 39       Go through 4 cycles
47| icx 70
48| ica 400   }
49| its 300 c }  Stores scalar product
50| ict 35       Go through 16 cycles
51| +.0
```

Finally, the following interpreted instructions are added to facilitate the handling of "red tape" while in the interpretive mode

Int. Inst.	Function
iat y (add and transfer)	Add the contents of the index register to the C(y) and store the result in the index register and register y.
iti y (transfer index digits)	Transfer the right 11 digits of the index register into the right 11 digits of register y.

These instructions primarily serve as a means of transferring the contents of the index register into a given storage register. Since the icr, ici and icd instructions enable one to set and change the contents of the index register, this register can be looked upon as an interpretive analogue of the single length, fixed point AC, with <u>iti</u> analogous to td, etc.

The Buffer Register

Although 2 register are used to store a (30-j,j) number, 3 registers are used for the MRA to avoid the time consuming operation of packing the last 15-j digits of the number and the j digits of the exponent into a single register after each interpreted instruction. A further advantage is gained in that any sequence of arithmetic operations is performed using 30 digits for the number and 15 digits for the exponent. This provides in effect a (30,15) system. The results of computation are combined into (30-j,j) number only when the C(MRA) is stored by the instructions *its* and *iex*.

The buffer register can be used in any of the instructions

icab, icsb, itsb, iexb, iadb, isub, imrb

In all of these cases "b" should be considered to represent a 3 register (30,15) location. Each of the instructions is then carried out as the corresponding instruction in a (30,15) interpretive subroutine would be carried out.

It should be emphasized that the above words represent the complete vocabulary available using the buffer symbol b.

The buffer can be used to store intermediate results in a cyclic program and thus rounding off can be avoided until after the final cycle.

Automatic Assembly of Interpretive Subroutines

Interpretive subroutines for computation in the (30,0), (15,15) and (30-j,j) number systems have been incorporated into the CCP in such a way that the type of subroutine and the features of this subroutine which are called for by the programmer in the process of writing his program are automatically punched out on 5-56 tape.

The kind of interpretive subroutine selected by the CCP will be determined by the value of the last (m,n) appearing on tape, e.g. if this is (30,0), the (30,0) interpretive subroutine will be selected. The corresponding (m,n) subroutine is then punched out onto paper tape if any interpretive instruction, ixy, is used in the program. However, the special word NOTPA (which means NOT Programmed Arithmetic) appearing anywhere on the tape overrides the effect of writing the interpretive instructions and generalized numbers, and no PA subroutine is automatically selected. NOTPA is used if a programmer wishes to convert (m,n) numbers and use an interpretive subroutine which is not part of the CCP or not to use any interpretive subroutine.

Particular interpretive subroutines are further specialized in accordance with the words appearing in a program. If the single letters b or c are used in any of the instructions in the program, then the corresponding buffer and cycle control subblocks in the particular PA selected are punched out. If these letters are not used the corresponding subblocks are not punched out. Similarly, if an idv instruction is used in a program, the division subblock is punched out. These specializations are made so that parts of the subroutine which are not used will not be read into storage.

The interpretive subroutines are automatically stored by the CCP in a block of registers ending in register 1056. The initial address of the block is found by adding up the lengths of the several subblocks punched out and subtracting the result from +1057.

A table of subblocks and their lengths follows:

Subblock			Words necessary on tape for read in	Length
(30-j,j)				
	PA	Buffer	final (30-j,j), b	39
		PA	final (30-j,j), ixy	199
		Cycle Count	final (30-j,j), c	57
		Divide	final (30-j,j), idv	26
(15,15)				
	PA	PA	final (15,15), ixy	113
		Cycle Count	final (15,15), c	57
		Divide	final (15,15), idv	9
(30,0)				
	PA	PA	final (30,0), ixy	135
		Cycle Count	final (30,0), c	57

Interpretive Operations and their functions

Interpreted Instruction	Function
ica*y (* refers to footnote and is not part of the instruction)	Clear the MRA and add into it the (m,n) number in location y.
ics*y	Clear the MRA and subtract from it the (m,n) number in location y.
iad*y	Add the (m,n) number in the MRA to the (m,n) number in location y and leave the sum in the MRA.
isu*y	Subtract from the (m,n) number in the MRA the (m,n) number in location y and leave the difference in the MRA.
imr*y	Multiply the (m,n) number in the MRA by the (m,n) number in location y and leave the product in the MRA.
idv*+y	Divide the (m,n) number in the MRA by the (m,n) number in location y and leave the quotient in the MRA.
its*y	Transfer the (m,n) number in the MRA to location y.
iex*y	Exchange the (m,n) number in the MRA with the (m,n) number in location y.
isp*y	Interpret next the instruction in register y.
sp y	Resume (15,0) WWI operation at register y.
icp*y	If the (m,n) number in the MRA is negative interpret next the instruction in register y; if positive, ignore this instruction.
ita*y	Transfer the address p+1 into the right 11 digits of register y, leaving the left 5 digits unchanged; p being the address of the isp or icp most recently interpreted.
icr m	Cycle Reset--set the index register to +0 and the comparison register to +m.
ict y	Cycle Count--increase the index register by one and form the quantity $\|\,\|a+1\|-\|b\|\,\|-0$. If this quantity is >0, interpret next the instruction in register y. If the quantity is =-0, ignore this instruction and reset the index register to +0.
ici m	Cycle Increase--increase the contents of the index register by +m.

* This interpretive operation is analogous to the (15,0) WWI operation obtained by dropping the initial i from the letter triple which designates it. The binary equivalent of the interpretive operation will not however be equal to the binary equivalent of the corresponding (15,0) WWI operation.

+ Not available on (30,0).

icd m	Cycle Decrease--decrease the contents of the index register by +m.
icx y	Cycle Exchange--exchange the contents of the index register with the contents of register y and exchange the contents of the comparison register with the contents of register y+1.
iat y	Add and transfer--add the contents of the index register to the contents of register y and store the result in the index register and register y.
iti y	Transfer index digits--transfer the right 11 digits of the index register into the right 11 digits of register y.

IV. INPUT/OUTPUT

Introduction

The output media currently available for use with the In/Out routine consist of typewriters, punches, oscilloscopes and magnetic tape units. The latter may be used to record data for subsequent print out on a magnetic typewriter or as auxiliary storage devices. The oscilloscope may be used in any of three ways:

a) as a curve plotting instrument

b) to display information in binary form

c) as a numeroscope displaying alphabetical or digital characters (i.e. "alphanumeric" characters) in any desired layout.

Following are the relative speeds of the several media for recording alphanumeric characters and also their characters/line limits:

a)	Typewriter	8 characters/sec.	160 characters/line max.
b)	Scope	150 characters/sec.	64 characters/line max.
c)	Magnetic Tape - to be used with Magnetic Typewriter	250 characters/sec.	90 characters/line max.

The In/Out routine is called into use by three upper case letters. The first specifies the equipment to be used, the second states whether information is to be fed into or out of the computer and the third specifies the type of information. The letters used are the initial letters of the following words:

Drum	In	Alphameric (alphanumeric)
Magnetic Tape	Out	Binary
Punch		Curve
Scope		
Typewriter		
Reader		

Examples of In/Out Instructions

TOA will print alphameric characters on the typewriter
SOC will display a curve on the scope
MIB will transfer binary information into the computer from magnetic tape.

A typical example of an output instruction is

$$\text{iTOA+p123.1234sx}2^{11}\text{x}10^{-2}$$

When the In/Out routine is called upon, it will handle the word currently in the AC or MRA. When a number expressed in any number system other than (15,0) is to be dealt with, the calling-in letters must be preceded by the lower case letter i so that the number will be interpreted. Thus iTOA will call in the output routine to print the contents of the MRA on the typewriter. At present, the following number systems are available; (30-n,n) with scale factor, (30-n,n) without scale factor, (15,0) with scale factor, (15,0) with binary point at extreme left, (15,0) with binary point at extreme right.

When the In/Out routine is required to print, display, or punch a number, the calling-in letters must be followed by a specimen number of the following general form where the numbers in parentheses refer to paragraphs below:

$$\underset{(1)}{\pm}\underset{(2)}{\alpha}\ \#\#\#\ldots\#\ \underset{(3)}{.}\ \#\#\ldots\#\ \underset{(4)}{\beta_i}\ \underset{(5)}{\times\ 2^{\gamma_i}}\ \underset{(5)}{\times\ 10^{\delta_i}}$$

The components of the number have the following meanings: (Note that in the following description the word "print" is used to mean print, punch or display, depending upon the medium previously selected).

(1) + = print the number preceded by its sign

- = print the number preceded by its sign if the number is negative, otherwise just print the number

sign omitted } = print the number with no sign

<u>note</u>: By "omitted" we mean that nothing at all is written. We do <u>not</u> mean that the word "omitted" is written.

(2) (α is a lower case letter)
(By initial zeros we mean initial zeros at the left of the decimal point.)

If α is i	initial zeros are <u>i</u>gnored in printing and the first significant digit of the number is printed on the extreme left of the column.
If α is p	initial zeros are <u>p</u>rinted as spaces.
If α is omitted	initial zeros are printed.
If α is n	the number is <u>n</u>ormalized before printing, e.g., all numbers are multiplied by such a power of 10 that the first non-zero significant digit will always be in the same relative position with respect to the decimal point. <u>This cannot be used with (15,0) output</u>.

The actual digits of the numerical part of the specimen number are immaterial; they merely serve to indicate the number of digits which the programmer desires to have printed on each side of the decimal point. Thus iTOA + p347.6210s x 2^{-3} x 10^{5} would indicate that the programmer wanted 3 digits to the left and 4 digits to the right of the decimal point and the numbers would be printed in the form ###.####. However, if α is n the number would be printed in the form ###.####x10^{u} which is the normalized case.

(3)	If a decimal point is indicated,	it will be printed in the position indicated.
	If a decimal point is omitted,	none will be printed. This is used in printing integers.
	If a decimal point is replaced by r,	no decimal point will be printed but the r indicates where a decimal point would have been placed had there been one.

The latter facility would be of practical use in the case of decimal fractions in which it is desired to save printing time by omitting decimal points. Unless one indicates a decimal point or replaces the decimal point by an r, the entire number will be treated as though it were an integer.

(4) (β is a lower case letter)

The symbol(s) β_i specify the character(s) with which the printed number is to be terminated:

If β_i is s	we get one space
If β_i is ss	we get two spaces
If β_i is sss	we get three spaces
If β_i is ssss	we get four spaces
If β_i is c	we get a carriage return
If β_i is t	we get a tab
If β_i is omitted	we get no terminating character
If β_i is f	we get format, i.e., the terminating character will be determined by the layout section of the In/Out routine which is in turn controlled by the Format Specification. (See paragraph on Format Specification)

(5) a) If the number is to be printed as a decimal fraction, then $\gamma = 0$, $\delta = 0$.

b) If the number is to be printed as a decimal integer, then $\gamma = 15$, $\delta = 0$.

c) Every factor must be preceded by a lower case x.

d) Any number of factors may be utilized, i.e., $2^{\gamma_1} \times 10^{\delta_1} \times 10^{\delta_2} \times 2^{\gamma_2}$ etc. with the following restriction: $\left|\sum\gamma_i\right|, \left|\sum\delta_i\right| \leq 127$

e) Whenever a factor such as 2^{γ} or 10^{δ} has a zero exponent, that factor may be omitted.

f) If any factor has an exponent of 1, the 1 may be omitted.

g) The exponents γ_i, δ_i are signed if negative, and not signed if positive.

Examples of the use of output instructions in the (30-n,n) system follow:

ex 1: Let the MRA contain the octal number 0.6277574516 with an exponent of 15 (octal).
Thus the number = $0.6277574516 \times 2^{15}$ (octal)
This is equivalent to $+.796812369 \times 2^{13}$ and to $+.652748693 \times 10^4$(decimal).
Let the output order be iTOA+nl.2345678c
Then the typewriter would print out $+6.5274869/+03_{2}$ where the number at the left of the slash is decimal and the number at the right is its exponent of 10. Thus the number is actually $+6.5274869 \times 10^3$.

ex 2: Let the MRA contain the octal number 1.1500203261 with an exponent of 15 (octal).
Thus the number = $1.1500203261 \times 2^{15}$ (octal).

This is equivalent to $-.796812369 \times 2^{13}$ and to $-.652748693 \times 10^{4}$ (decimal).
Let the output order be iSOA - $12.3456s \times 10^{-5}$
Then the 'scope would display -00.0652 sp. (see note below) where sp. means that a space is provided ~~for~~ on the 'scope.
note: At present no provision is made for rounding off to -00.0653.

In/Out Order Repeated

A specimen number need not be designated each time the In/Out routine is called in. If the calling-in letters are not followed by anything, then the In/Out routine will provide exactly the same set up as it furnished for the last In/Out specification. By exactly the same we mean that if one wrote iSOA following an iTOA + $i12.345s \times 2^{-4} \times 10^{6}$, he would automatically get iTOA + $i12.345s \times 2^{-4} \times 10^{6}$. If the programmer wants the same In/Out order as the last one except for the calling-in letters, he must write out the In/Out order in its entirety.

Format Specification

The In/Out routine contains a layout section which may be set by the special word:

FOR $\alpha \times \beta \times \gamma$

a) this word must precede any output order for which it is to be used

b) the entire word FORMAT may be written instead of FOR, if desired

c) α represents the number of words/line. (maximum is 15)

d) numbers α, β, γ should be separated by lower case x

e) β represents the number of spaces between words (maximum is 6)
If a tab is desired between words then set $\beta = 7$

f) γ represents the number of words per block (typewriter)
γ represents the number of words per frame ('scope)

The maximum γ is 511. However, if the programmer has more than 511 words to be printed, the block counter becomes automatically reset after each block is completed.

ex 1: Let us suppose that the programmer wishes to type 2500 words in blocks of 400. If he specifies that $\gamma = 400$, then he will automatically get 6 blocks of 400 words each and a seventh block of 100 words. The blocks will be separated by 2 carriage returns. In order to get the final 100 words as a separate block one must heed the following note.
Note: provision is made for one automatic carriage return at the beginning of the Format routine and two at the end of a block. However, the programmer should provide carriage returns at the end of his print-out if that doesn't coincide with the end of a block. This carriage return order is described in the Special Characters section.

ex 2: Let us consider ex. 1 if the scope were being used instead of the typewriter. The only difference is the restriction on the number of lines per frame which is 36. However, if the programmer requested 8 words/line, 400 lines/block, he would get 288(8x36) words on one frame and 112(400-288) words on the next frame since provision is made for automatic indexing at the end of 36 lines and at the end of a block. Thus the programmer would

get altogether 6 frames of 288 words each, 6 frames of 112 words each and one frame of 100 words. However, the last frame of 100 words will be obtained only if the following note is heeded.
Note: The programmer must provide the order FRAME in order to index the camera at the end of any particular display since it is unlikely that the end of a display will coincide with the filling up of a frame or the end of a block. An automatic index is provided at the beginning of the display routine.

Special Characters

a) One may obtain a -, +, ., s (space), t (tab), c (carriage return) at any time by merely using the call-in letters followed by any one of the above six.

```
exs: TOA +  gives a + on the typewriter
     SOA c  gives a carriage return on the 'scope
```

b) The order COL continues the 'scope display in the next column, at the top of the frame.

The order FRA takes a picture, and sets the camera up for the next frame.

One may use the entire word COLUMN, FRAME instead of COL, FRA respectively but all letters must be upper case.

V. Conclusion

At present the CS is entirely on paper tape. Strides have been made in the direction of replacing some of the paper tapes with magnetic tapes. The latter transition will depend to a considerable extent upon the availability of magnetic tape units. At present only one magnetic tape unit is available whereas it is considered that three tape units is the optimum number for the efficient use of the CS. It is planned to store the CS permanently in the magnetic drum as soon as the drum is available. Post-mortems (PM) and Mistake Diagnosis (MD) routines will be incorporated into the CS in the near future. As soon as new In/Out routines are prepared, they will be incorporated into the CS.

Signed H. C. Uchiyamada
H. C. Uchiyamada

Signed E S Kopley
E. S. Kopley

Approved CWA.
C. W. Adams

Index for Engineering Note E-516