June 18, 1957

REVISION NO. 1 to

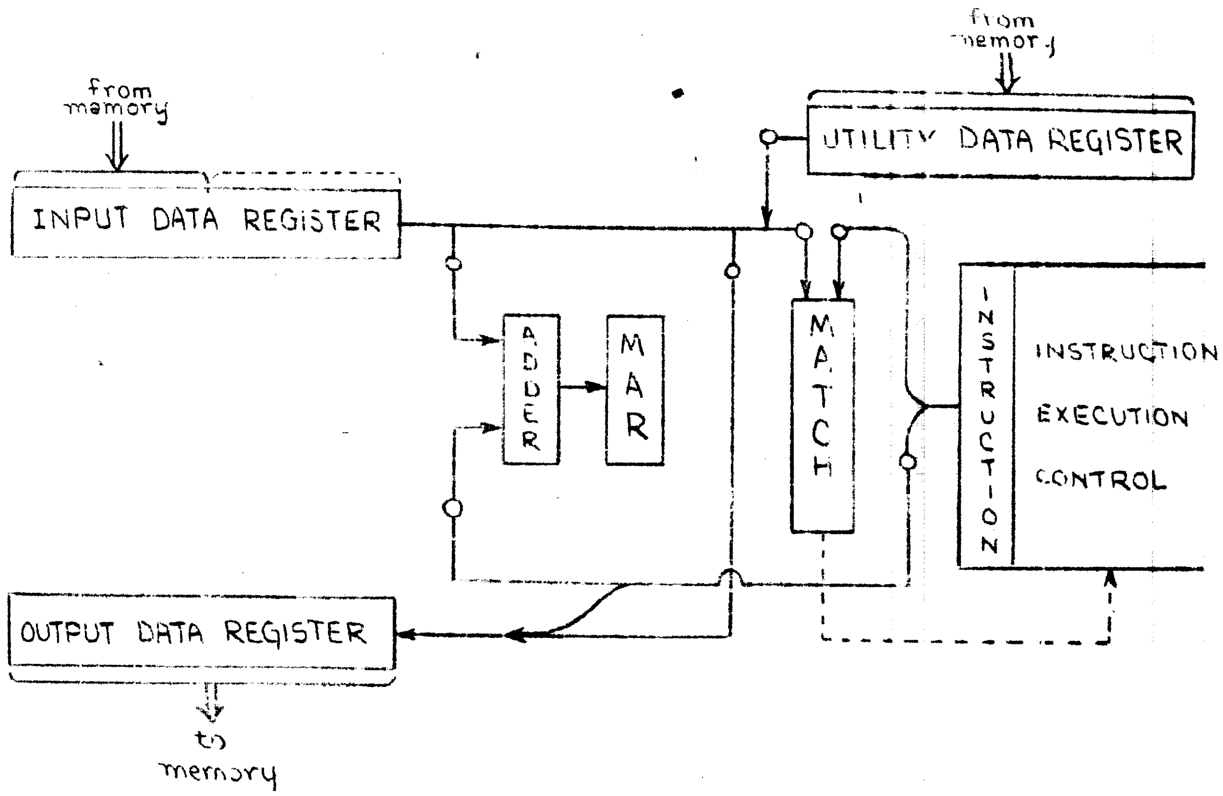
File Memo, dated June 11, 1957

Subject: The Primitive Instruction Approach to Editing

By: H. C. Montgomery

As the result of a preliminary examination of the editing system described in the subject memorandum, the following changes and additions have been made:

A. It has been pointed out that it is at times desirable to be able to perform matching operations on the output data resulting from table look-up operations. The system shown in Figure 1 of the memorandum makes no provision for doing this. Hence, the diagram should be changed as shown below.



Moreover, the table look-up instruction format will require an additional field to provide a match character, and a modifier bit to control when it is to be used. Thus, the format for the table look-up instruction will appear as:

WORD ADDRESS	MATCH CHAR.	FIELD LENGTH INPUT	FIELD LENGTH OUTPUT	REPETIT.	OPERAT CODE	BY SZ OUT	BY SZ IN	I T

B. An editing problem has been suggested that has made it clear that it will be necessary to be able to determine whether an input character is identical to any one of several characters, and to further be able to take a different course of action for each of the possible alternatives. To facilitate this, the following instruction is defined to supplement the eight given in the memorandum:

> BOM (Branch On Match).
> The next input byte and the match character are compared as in the TMT instruction. If they are identical, a transfer is effected to the instruction whose address is given by the word address field of the present instruction; if they are not identical, the next instruction in the present sequence is taken. The bit address controls of both the IDR and ODR remain unchanged by this instruction.

C. The introduction of the BOM instruction permits the deletion of the TMN instruction from the list. The motivation for the TMN instruction is illustrated by Example 1 of the memorandum in which a numerical field is edited to remove leading zeros and punctuate it in the usual dollar field format. The TMN instruction is used after the decimal point to take care of numerical fields which are zeros.

Recall that in the original memorandum, Example 1 appeared as:

Model Field	PPS	APS	Input	Output	Input	Output
≀	≀	≀	≀	≀	≀	≀
∘	ICH(.)	ICH(.)		.		. b
x	TMN	PBY	8	8	0	0 b
x	TMN	PBY	1	1	0	0 b
	BSP(3)					
	ICH(3 repet.)					

The same results can be obtained by using the BOM instruction as follows:

```
  {         {
  {         {
  .        ICH(. )                                 insert decimal point
  x        BOM(0) ──► PBY                          pass the tenths digit
  x        PBY          BOM(0) ──► BSP(3)          if both digits are 0,
           PBY          PBY          ICH(b)        back up and fill in
                                   (3 repet. )     blanks;  if the last
                                                   digit is non-zero,
                                                   pass it too.
```

This cascade effect of the BOM instruction could be written in more conventional form by writing the PPS as:

```
   {
   ICH (. )
┌--BOM (0)
│  PBY
│  PBY
│  TRA (out)
└─►PBY
┌--BOM (0)
│  PBY
│  TRA (out)
└─►BSP (3)
   ICH (b) (3 repet. )
   .
   .
   .
```

D. It is also possible to use the BOM instruction to effect the same editing operations for which the TMS instruction was used in Example 3 of the memorandum. This could be done by writing the PPS in the following way:

```
┌--BOM (b)
│  TRA  Z
└─►SBY
┌--BOM (b)
│  TRA (Z + 1)
└─►SBY
┌--BOM (b)
│  TRA (Z + 2)
└─►SBY
   .
   .
   .
   etc.
```

Z is the address of the instruction in the APS which corresponds to the first BOM instruction in the PPS.

By comparing the above PPS with that using the TMS instruction in Example 3 of the memorandum, it is clear that the PPS using the BOM instruction is approximately three times as long as the one using the TMS instruction. Hence, the price of eliminating the TMS instruction for this example is an increase in the number of instructions required.

E. In order to preserve the simplicity of the primitive instructions, it has been decided to delete the unconditional transfer associated with the SBY (Skip Byte) instruction. This deletion has no effect on the flexibility of the system since the same operations can be done by using an ordinary transfer instruction immediately following the SBY instruction in those cases where it is needed.

F. A serious problem which can arise in the use of the BSP (Backspace) instruction is that which occurs when the range of the backspacing extends into a word which has already been sent to memory by the automatic storing feature of the ODR. As the instruction is defined in the memorandum, this situation would necessitate the bringing back of the previously stored word and replacing the present contents of the ODR with the word.

This problem can be avoided without sacrificing the backspacing feature by providing another instruction which resembles a variable field length LOAD instruction. Adopting this compromise means that the programmer must know the word and bit address of the operand, and therefore that he cannot merely use the instruction on the "next byte" basis as is possible with the other primitive instructions. It is believed, however, that the problem applications for such an instruction are of a type such that the information concerning operand addresses will be easily available to the programmer, and that the considerable saving in hardware achieved by adopting this approach justifies the additional programming difficulty.

The BSP instruction will, therefore, be removed and an Adjust Output Field (AOF) instruction substituted for it. The AOF instruction is defined as:

The word whose address is given by the instruction is loaded into the ODR. The bit address control of the ODR is set to the place specified by the bit address field of the AOF instruction. The contents and controls of the other machine registers are unchanged.

The bit address control of the ODR may be changed without altering its contents by using the AOF instruction with the new bit address and giving the address of the ODR in the word address field of the instruction.

The format for the AOF instruction is:

not needed

WORD ADDRESS	BIT ADDR.	NOT USED	REPETITIONS	NOT USED	OPN. CODE	BY SZ.	I T

G. Although it was stated in the original memorandum that the byte size of input data was specified by the primitive instructions, it should also be pointed out that the specifications are exercised by a byte size control associated with the IDR. A similar control is associated with the ODR, but it must be set by a special instruction before the primitive instructions are used, if it is not already set to the correct value. An exception, of course, is the table look-up instruction which sets the byte size controls of both the IDR and ODR.

It should be observed that the setting of the various controls for the IDR and ODR, as well as loading the proper input data into the input register, belongs to a set-up process which must be completed before the primitive instructions are used.

H. Since the instruction list has undergone several changes and deletions, it is given here in its present form.

1. SOM (Substitute On Match).
The next byte of input data is sent to the match register where it is compared with the match character sent there from the instruction. If the two bytes are identical, the utility byte from the instruction is sent to the ODR and the bit address controls of the IDR and ODR advanced. If the bytes are not identical, the bit address controls of the IDR and ODR are left unchanged and control transferred to the instruction whose address is given by the word address field of the present instruction.

2. BOM (Branch on Match).
The next input byte and the match character are compared as in the SOM instruction. If they are identical, a transfer is effected to the instruction whose address is given by the word address field of the present instruction; if they are not identical, the next instruction in the present sequence is taken. The bit address controls of both the IDR and ODR remain unchanged by this instruction.

3. ICH (Insert Character).
The utility character is sent to the ODR a number of times equal to the number in the repetition field of the instruction. The bit address of the ODR is advanced by one byte for each insertion, while the bit address control of the IDR is left unchanged.

4. PBY (Pass Byte).
The next input byte is sent unchanged to the ODR. The bit address controls of both the IDR and ODR are advanced by one byte for each repetition of this instruction.

5. AOF (Adjust Output Field).
The word whose address is given by the instruction is loaded into the ODR. The bit address control of the ODR is set to the place specified by the bit address field of the AOF instruction. The contents and controls of the other machine registers are unchanged.

6. SBY (Skip Byte).
The bit address control of the IDR is advanced by one byte length.

7. TLU (Table Look-Up).
The input field in the IDR, which is identified by the bit address control of the IDR and the input field length given by the instruction , is added to the word address given by the instruction. The word containing the field identified by this sum is brought to the UDR. If the matching modifier bit is 0, the output field, defined by the output field length given by the instruction and the bit address control of the UDR, is sent to the ODR. If the matching modifier bit is 1, the first byte of the table entry is compared with the match character. If the two bytes are identical, the next instruction of the sequence is taken. If the two bytes are not identical, the table entry byte is sent to the ODR and the next instruction of the sequence is skipped.

how about repetitions on match?

The bit address of the IDR is advanced by the input field length amount, while the bit address of the ODR is advanced by the output field length, output byte size, or not at all, depending upon the status of the matching modifier bit and the nature of the table entry.

EXAMPLES

1. Suppose it is desired to edit a ten digit numerical field to put it in a dollar field format.

Model Field	PPS	APS	Input	Output	Input	Output	Input	Output	
x	TMT	PBY							
x	TMT	PBY							
x	TMT	PBY							
,	ICH(b)	ICH(,)							
x	TMT	PBY							
x	TMT	PBY	5	5	0	b	0	b	
x	TMT	PBY	4	4	0	b	0	b	
,	ICH(b)	ICH(,)		,		b		b	
x	TMT	PBY	9	9	0	b	0	b	
x	TMT	PBY	7	7	7	7	0	b	
x	TMT	PBY	3	3	0	0	0	b	
,	ICH(b)	ICH(,)		,		,		b	
x	TMT	PBY	6	6	0	0	0	b	
x	TMT	PBY	4	4	3	3	0	b	
x	TMT	PBY	2	2	5	5	0	b	
.	ICH(.)	ICH(.)		.		.		.	b
x	BOM(0)	PBY	8	8	1	1	0	0	b
x	PBY	PBY	1	1	2	2	0	0	b
	PBY								
	TRA(out)								
	PBY								
	BOM(0)								
	PBY								
	TRA(out)								
	AOF(3 bytes)								
	ICH(b) (3 repet.)								

2. A related problem is that of reducing fields of the form bbbx, xxx. xx to xxxxxx by deleting leading blanks and punctuation. The unknown quantity in this case is the number of leading blank characters. The overall length of the field is known.

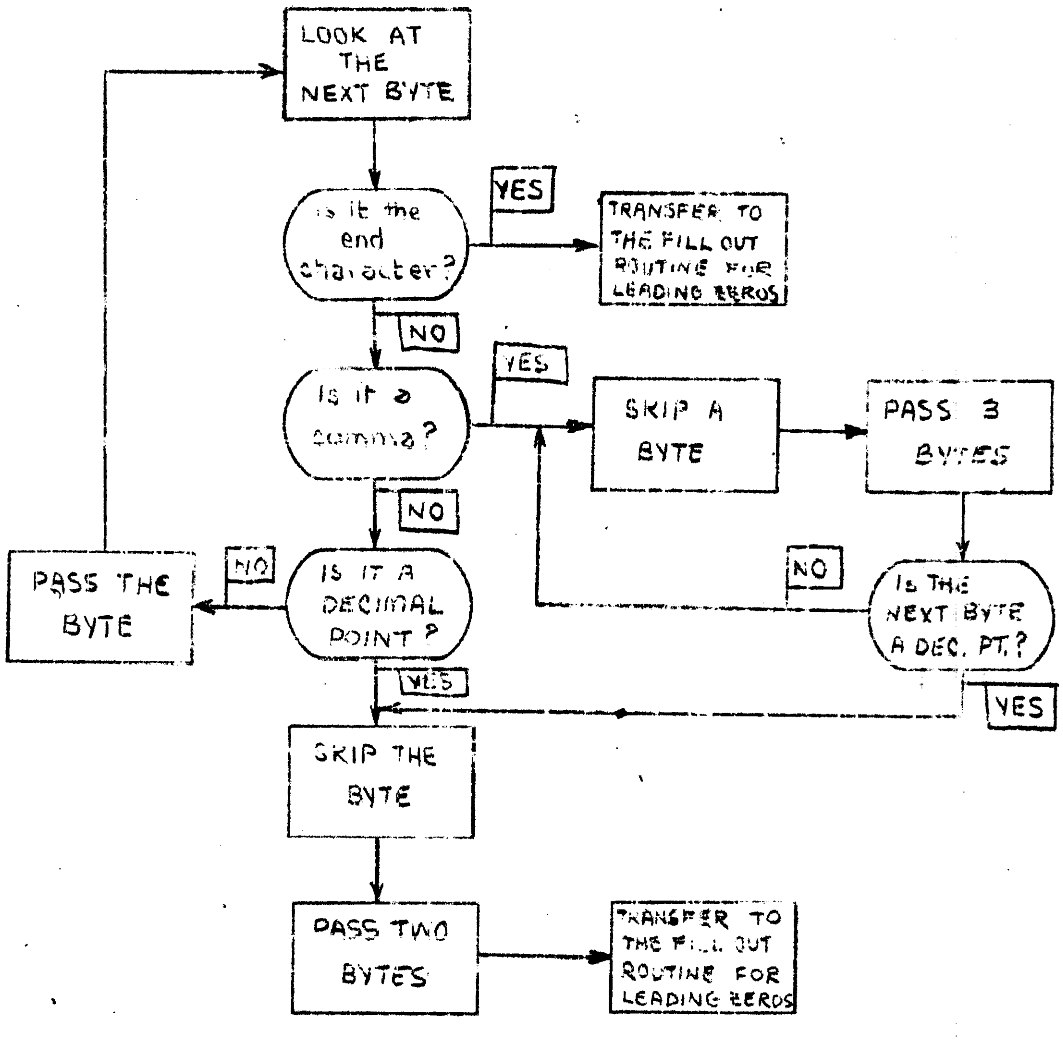
Possible Input	PPS	APS	Input	Output	Input	Output	Input	Output
(b) x	BOM(b)							
	TRA →	PBY						
	SBY							
(b) x	BOM(b)		2	2	b		b	
	TRA →	PBY						
	SBY							
(b) x	BOM(b)		8	8	b		b	
	TRA →	PBY						
	SBY							
,	SBY	SBY	,		b		b	
(b) x	BOM(b)		5	5	b		b	
	TRA →	PBY						
	SBY							
(b) x	BOM(b)		4	4	7	7	b	
	TRA →	PBY						
	SBY							
(b) x	BOM(b)		6	6	4	4	b	
	TRA →	PBY						
	SBY							
.	SBY	SBY	.		.		b	
x	BOM(b)		7	7	5	5	b	
	TRA →	PBY						
	SBY							
x	BOM(b)		9	9	2	2	b	
	TRA →	PBY						
	SBY							

3. Yet another problem of this same class is that of having as input a dollar amount field of unknown length and producing from it a purely numerical (unpunctuated) field of standard length. The input field always has an unpunctuated length less than or equal to the standard length and would be filled out to the standard length for the output by placing zeros before the leading digit. The starting point address of the input field is assumed known, while the other end of the field is marked by a special character E.

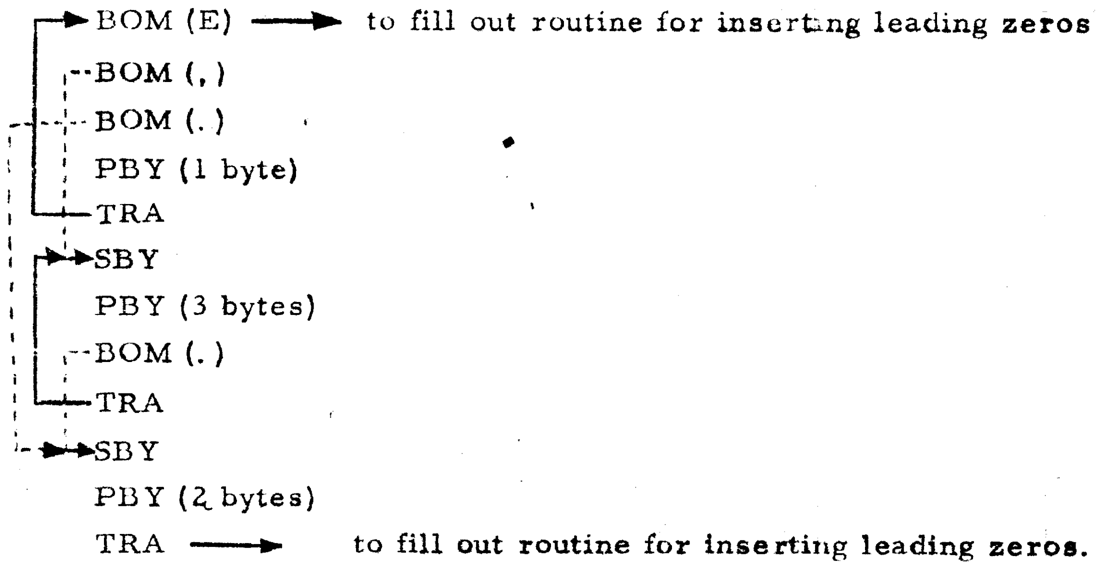
Hence, the problem is

```
INPUT  DATA:   xx, xxx, . . . , xxx, xxE
OUTPUT DATA:   00000xxxxxxxxxxxxxxxx
```

The flow diagram for the solution, in terms of bytes (characters), is:



The symbolic program for this solution, using the primitive instructions, can be written as:



```
BOM (E)  ───►  to fill out routine for inserting leading zeros
BOM (,)
BOM (.)
PBY (1 byte)
TRA
SBY
PBY (3 bytes)
BOM (.)
TRA
SBY
PBY (2 bytes)
TRA  ───►      to fill out routine for inserting leading zeros.
```

The solution to this problem will require a counting mechanism which can maintain a count of output bytes from a given starting point. The mechanism could be set to the number of output bytes required for the output field and reduced each time a byte enters the ODR. When the input data have been exhausted, the contents of the counting mechanism can be used to set up the repetitions field of the ICH instruction used for inserting the leading zeros.

4. The variable byte size and matching features of this system provide considerable flexibility in the manipulations which can be performed on fields of signed numerical data.

Consider the problem in which the sign bit of a numerical quantity is contained in a sign byte of several bits. The location of the sign byte is known and it is desired to determine the sign of the numerical quantity and to take one of two different courses of action, depending upon whether the sign is plus or minus. Assume that the sign byte and numerical quantity have been loaded into the input data register. The input data are stepped through to the left-most bit of the sign byte. Using a byte size of 1 bit, the bits of the sign byte are stepped through up to the sign bit itself. Then the BOM instruction is given, again using a 1 bit byte size. The word address in the BOM instruction gives the address of the first instruction in the subroutine which is used if the sign is plus. The BOM instruction is followed by a transfer instruction which transfers to the first instruction in the subroutine used when the sign is minus.

Symbolically, the program for treating the sign in this manner would be:

```
        .
        .
        .
   PBY (digits)     to step through the numerical quantity to the sign byte
   PBY (bits)       to step through the sign byte to the sign bit
   BOM (+) (Z)      transfer to location  Z  if the sign is plus
   TRA  X           transfer to location  X  if the sign is minus
        .
        .
        .
```