ED D1(L1,B1),D2(B2)		**Edit**			**SS₁**

Op Code	LL₁	B₁D₁	D₁D₁	B₂D₂	D₂D₂

Packed decimal fields can be converted to a character format using the **ED** instruction. Additionally, editing symbols and features like commas, decimal points, and leading zero suppression can be included in the character version of the packed decimal field that is being edited. The first step in editing a packed decimal field is to create an "edit word" which is a pattern of what the character output of the edit process should look like. Typically, the edit word is moved to a field in the output buffer which is being built, prior to printing. Then the packed decimal field is "edited" into the output field, destroying the copy of the edit word.

First, we consider how to construct an appropriate edit word for a given packed decimal field. This can be accomplished by defining a string of hexadecimal bytes that represents the edit word. Each byte in the edit word corresponds to a byte in the edited character representation. In creating the edit word there are a collection of standard symbols which are used to describe each byte:

X'40'   This symbol, which represents a space, is usually coded as the first byte of the edit word where it acts as a "fill character". The fill character is used to replace leading zeroes which are not "significant".

X'20'   Called a "digit selector", this byte represents a position in which a significant digit from the packed field should be placed.

X'21'   This hexadecimal byte represents a digit selector and a significance starter. Significance starts when the first non-zero digit is selected. Alternatively, we can force significance to start by coding a single x'21' in the edit word. In this case, significance starts in the byte **following** the x'21'. Significance is important because every significant digit is printed, even if it is a leading zero.

X'6B'   This is the EBCDIC code for a comma.

X'4B'   This is the EBCDIC code for a decimal point.

X'60'   This is the EBCDIC code for a minus sign. This symbol is sometimes coded on the end of an edit word when editing signed fields. The x'60' byte will be replaced with the fill character if the number being edited is positive. If the number is in fact negative, the x'60' will not be "filled", and the negative sign will appear in the edited output.

X'C4C2' The EBCDIC version of "DB" (Debit). This functions like the x'60'. Coding these symbols at the end of an edit word causes "DB" to appear in the output if the field being edited is negative, otherwise the "DB" is "filled".

X'C3D9' The EBCDIC version of "CR" (Credit). This functions like the Debit symbol above. When the number being edited is negative, the "CR" symbol will

appear in the edited output, otherwise it will be "filled".

X'5C'   The EBCDIC symbol for an asterisk. This character is sometimes used as a fill character when editing dollar amounts on checks.

We now consider a sample edit word and the output it would create for several packed fields.

```
EDWD     DC    X'4020206B2021204B202060'
AOUT     DC    CL11
APK      DC    PL4'45387'
```

Assume we execute the instructions below,

```
MVC      AOUT,EDWD
ED       AOUT,APK
```

First, the edit word is moved to a field in the output buffer. Then the packed field is "edited" into the output field.   The results are illustrated in the diagram below.

APK				00	45	38	7C

EDWD	40	20	20	6B	20	21	20	4B	20	20	60

AOUT	40	40	40	40	F4	F5	F3	4B	F8	F7	40
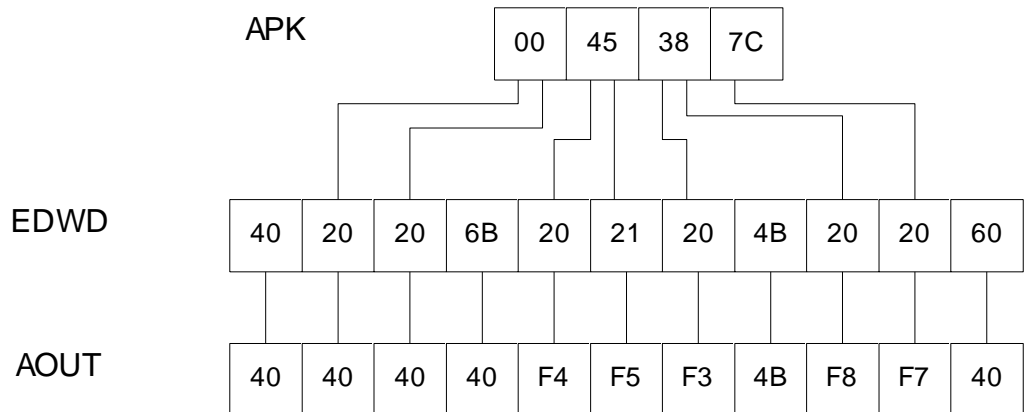
The diagram indicates the results of the edit process: The fill character (x'40') is unaffected, and is left in its position. The first decimal digit, 0, is "selected" by the first x'20', and since leading 0's are not significant, the x'20' is replaced by the fill character. The second digit, 0, is also selected, and it too, is filled with a x'40'. Since significance has not started, the x'6B' is filled with x'40'. The first non-zero digit, 4, is selected and this signals that significance has started. (Any non-zero digit which is selected turns on the significance indicator.) Each digit after the 4 will appear in the edited result. The "4" is replaced with its character equivalent - x'F4'. The "5" is selected and its x'20' is replaced with x'F5'. The "3" is selected and is represented as x'F3'. The x'4B', a decimal point, remains unaffected. The "8" is selected and is represented as x'F8'. The "7" is selected and is represented as x'F7'. Since the number being edited is positive, the x'60' is filled with x'40'.   The final result would print as "   453.87 ".
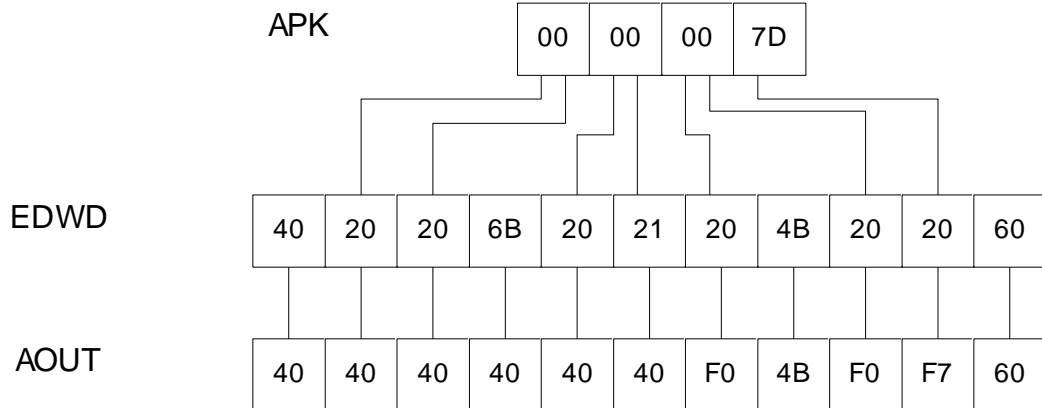
Consider a second edit which uses the same edit word as in the previous example, but with a different value for APK.

```
EDWD     DC    X'4020206B2021204B202060'
AOUT     DC    CL11
APK      DC    PL4'-7'
```

Again we execute the same sequence of instructions.

```
MVC      AOUT,EDWD
```

```
                    ED        AOUT,APK
```

APK	00	00	00	7D

EDWD	40	20	20	6B	20	21	20	4B	20	20	60

AOUT	40	40	40	40	40	40	F0	4B	F0	F7	60

As in every edit, the x'40' fill character is unaffected by the edit process. The first and second digits, both 0, are selected, and since they are leading 0's and significance has not started, they are filled with x'40'. The x'6B' is also filled with x'40' since significance has not started. The next two digits, both 0, are selected and filled. Since the x'21' selected a leading 0, the significance indicator is turned on - significance starts with the **next** digit. This means that all other digits will appear in a character representation, even if they are leading 0's. All other editing symbols will be printed as well. The fifth digit, 0, is selected and represented as x'F0'. The x'4B' is preserved. The next two digits, 0 and 7, are selected and represented as x'F0' and x'F7'. Finally, since the APK contains a negative number, the x'60' is preserved.   The final result would print as " 0.07-".

Examples

Some Unrelated ED's:

```
APK       DC    PL2'123'   X'123C'
AOUT      DS    CL4
AEDWD     DC    X'40202020'
          ...                 Result:
          MVC   AOUT,AEDWD
          ED    AOUT,APK        AOUT = X'40F1F2F3' - ' 123'

BPK       DC    PL2'0'   X'000C'
BOUT      DS    CL4
BEDWD     DC    X'40202020'
          ...                 Result:
          MVC   BOUT,BEDWD
          ED    BOUT,BPK        BOUT = X'40404040' - '    '

CPK       DC    PL2'0'   X'000C'
COUT      DS    CL4
CEDWD     DC    X'40202120'
          ...                 Result:
          MVC   COUT,CEDWD
          ED    COUT,CPK        COUT = X'404040F0' - '   0'
```

```
      DPK         DC    PL2'0'    X'000C'
      DOUT        DS    CL4
      DEDWD       DC    X'5C202120'    ASTERISK IS USED AS A FILL CHARACTER


                  ...                Result:
            MVC   DOUT,DEDWD
            ED    DOUT,DPK         DOUT = X'5C5C5CF0' - '***0'


      EPK         DC    PL2'-30'   X'030D'   NEGATIVE NUMBER
      EOUT        DS    CL4
      EEDWD       DC    X'40202120'
                  ...                Result:
            MVC   EOUT,EEDWD
            ED    EOUT,EPK         EOUT = X'4040F3F0' - '  30'


      FPK         DC    PL2'-30'   X'030D'    NEGATIVE NUMBER
      FOUT        DS    CL5    OUTPUT FIELD IS LARGER TO ACCOMODATE EDIT
WORD
      FEDWD       DC    X'4020212060'
                  ...                Result:
            MVC   FOUT,FEDWD
            ED    FOUT,FPK         FOUT = X'4040F3F060' - '  30-'
```

☞ **Tips**

1. There are two errors that beginners make when using **ED** :

   1) The number of x'20's and x'21's does not match the number of decimal digits in the field
      being edited. This is a critical error. If the packed field has length "n", the number of
      x'20's and x'21's is 2n - 1. For example, if you are editing a packed field of length 6, the
      edit word must contain exactly 11 x'20's and x'21's. A bad edit word will produce
      unpredictable output.

   2) The output field size does not match the edit word size. For example, suppose you coded
      the following,

```
               AEDWD       DC    X'402020202120'
               AOUT        DS    CL5
```
   When the edit word is moved to AOUT, the last byte of the edit word is not moved since
   the edit word is 6 bytes and the target field is 5 bytes. The effect is that we are using an
   incorrect edit word, even though the definition of the edit word was correct.

2) When editing, start with the packed field and design an edit word that matches it. Then define
the output field to match the edit word. For example, if we start with a packed field of length 3 (5
decimal digits), we could design x'402021204B2020' as an appropriate edit word (5 x'20's and
x'21's). Since the edit word is 7 bytes long, we would design a 7 byte output field to hold the edit
word.

```
               XPK         DS    PL3
               XEDWD       DC    X'402021204B2020'
               XOUT        DS    CL7
```

```
...
MVC     XOUT,XEDWD
ED      XOUT,XPK
```