Chapter 2
Defining and Moving Character Data

Objectives

Upon completion of this chapter you will be able to:
- Define alphanumeric fields and records using the DS and DC instructions,
- Move alphanumeric fields using the MVC instruction,
- Define alphanumeric literals and equated values,
- Move alphanumeric literals using the MVI and MVC instructions,
- Produce a formatted list of the records in a file, and
- Produce report headings.

Introduction

In the previous chapter we produced an 80/80, or card-image, list of the records in the TEACHER file. In this chapter we will continue with that example. Specifically, we would like to produce formatted report with report and column headings. We will begin by producing a quick-and-dirty listing of the records in this file. By quick-and-dirty we mean that, while each field will be in its own column, there will be no headings, no page numbers, etc. Our listing will appear as follows:

```
         1         2         3         4         5         6
1234567890123456789012345678901234567890123456789012345678901234567890
XXX   XXXXXXXXXXXXXXX   XXXX   X   XXXX
XXX   XXXXXXXXXXXXXXX   XXXX   X   XXXX
XXX   XXXXXXXXXXXXXXX   XXXX   X   XXXX
:     :                  :      :   :
:     :                  :      :   :... Phone
:     :                  :      :....... Tenured?
:     :                  :.............. Degree
:     :................................. Name
:..................................... ID number
```

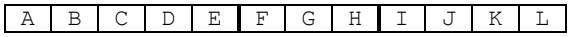
In order to do so, we must be able to
- describe the input and output records, and
- move the input fields to their respective output fields.

Defining and Moving Character Data

The first instruction we will look at is the MVC instruction. MVC stands for Move Character. It is used to copy the contents of one field into another. The instruction is quite different from the move instruction (or its equivalent) in other languages. Specifically,

> **When using the MVC instruction, the length of the move is determined strictly by the length of the receiving field (unless overridden), and a maximum of 256 characters can be moved.**

To illustrate, we will define three fields: FLDA is five byte long and contains 'ABCDE', FLDB is three bytes long and contains 'FGH', and FLDC is four bytes long and contains 'IJKL'. FLDA, FLDB, and FLDC then occupy (5 + 3 + 4) = 12 contiguous bytes of memory. This can be shown as follows:



We would define these fields in BAL as follows:

```
FLDA     DC    CL5'ABCDE'
FLDB     DC    CL3'FGH'
FLDC     DC    CL4'IJKL'
```

DC stands for Define Constant. Its purpose is (1) to allocate space, (2) to assign a name to that space, (3) to indicate the type of data the space is intended to hold, and (4) to give that space some initial value.

```
FLDA     DC    CL5'ABCDE'
:        :     ::  :
:        :     ::  :... Initial value is ABCDE
:        :     ::       Value is always enclosed in apostrophes
:        :     ::
:        :     ::..... Length of field is five (L5)
:        :     :
:        :     :....... Field is intended to hold character data
:        :
:        :............. Define Constant
:
:...................... Field name is FLDA
```

For the benefit of the reader who knows COBOL, the COBOL equivalent would be as follows:

```
05  FLDA     PIC X(5)    VALUE 'ABCDE'.
05  FLDB     PIC X(3)    VALUE 'FGH'.
05  FLDC     PIC X(4)    VALUE 'IJKL'.
```

In COBOL, if the receiving field is shorter than the sending field, the move is truncated. For example, MOVE FLDA TO FLDB results in a value of 'ABC' in FLDB, with 'DE' being truncated. On the other hand, if the receiving field is longer than the sending field, the receiving field is padded with trailing blanks. For example, MOVE FLDB TO FLDA results in a value of 'FGH~~bb~~' in FLDA (~~b~~ will be used throughout to indicate a blank. The strike-through is used to distinguish a blank from a lowercase b).

Now, let's attempt the BAL equivalent of the previous COBOL moves. To move FLDA to FLDB, we code: MVC FLDB,FLDA

In BAL, the instruction, or **operation**, must begin some place after column one. Recall that column one is for comments and labels. The operation usually begins in column ten by convention. At least one blank must separate the instruction from its parameters, or **operands**. The operands usually begin in column sixteen by convention. Multiple operands are separated by commas, without blanks. Optionally, comments may be put on the remainder of the line: at least one blank must separate the last operand from the comments.

Note that with the MVC instruction (as with most, but not all, BAL instructions), the first operand is the receiving field. As mentioned above, the length of the move is determined by the length of the receiving field. In this case, FLDB is three bytes long (CL3 in the DC instruction), so three bytes will be moved *regardless of the length of the second operand, or sending field* .

The result is that the first three bytes of 'ABCDE' will be moved to FLDB, the result being FLDB contains 'ABC'. The same twelve bytes in memory would now contain:

A	B	C	D	E	**A**	**B**	**C**	I	J	K	L

We see that when the length of the receiving field is less than or equal to the length of the sending field, BAL's MVC works the same as COBOL's MOVE.

Let's try the second example. To avoid confusion, we start with "fresh" data:

A	B	C	D	E	F	G	H	I	J	K	L

To move FLDB to FLDA, we code `MVC FLDA,FLDB`

In this case, FLDA is five bytes long (CL5 in the DC instruction), so five bytes will be moved *even though FLDB is only three bytes long!* The result will depend upon what is in the first two bytes of memory immediately following FLDB. In this case it is 'IJ', the first two bytes of FLDC. Therefore FLDA will contain 'FGHIJ'. The same twelve bytes in memory would now contain:

F	**G**	**H**	**I**	**J**	F	G	H	I	J	K	L

The results are probably not what you would have expected! This concept of the length of an operation being determined by the length of one operand (only), regardless of the length of the other, is one of the more difficult concepts to get used to when learning BAL for the first time.

* *

You Try It...

Show the result of each of the following MVCs. Start with fresh data each time.

		A	B	C	D	E	F	G	H	I	J	K	L
	Before												
1.	MVC FLDC,FLDA												
2.	MVC FLDC,FLDB												
3.	MVC FLDB,FLDC												

* *

We return now to the problem at hand: to produce a quick-and-dirty listing of the records in the TEACHER file according to specifications. First, we need to define the input record. Recall the record layout for the TEACHER file:

Field Nbr	Field Name	Description	Begins	Ends	Len	Format
1	TID	Teacher ID	1	3	3	ZD
2	TNAME	Teacher name	4	18	15	CH
3	TDEG	Highest degree	19	22	4	CH
4	TTEN	Tenured?	23	23	1	Y/N
5	TPHONE	Teacher phone	24	27	4	ZD
6	TCRLF	PC/370 Only	28	29	2	CR/LF

Note that I have chosen to limit the field names in all of my record layouts to seven characters. BAL allows eight-character field names. Field names may contain national (A-Z, @, #, and $) or numeric (0-9) characters, and must begin with a national character. I have limited my field names to seven characters, so that I can add one more character in front, such as I (for input fields), O (for output fields), or W (for work fields).

As this is the description of the input record, I will use an I as the first character of each field name (not because I *have* to, just because it makes sense.) The complete record layout follows:

```
IREC     DS    0CL29             Teacher record
ITID     DS    CL3               Teacher ID nbr
ITNAME   DS    CL15              Teacher name
ITDEG    DS    CL4               Highest degree
ITTEN    DS    CL1               Tenured?
ITPHONE  DS    CL4               Phone nbr
ITCRLF   DS    CL2               PC/370 only - CR/LF
```

Here we use the DS instruction rather than the DC. DS stands for Define Storage. It serves the same purpose as DC, except that it does not assign an initial value. (If you are familiar with COBOL, a DS is like a field definition without a VALUE clause.) There is no need to assign a value to these fields. They will not be referenced prior to the first file read (GET), after which they will all have values.

Comments are used to describe each field. Nevertheless, field names should be as meaningful as possible (with the eight character limitation). I often include the beginning and ending positions of the the field in the comments. For example:

```
IREC     DS    0CL29        1-29  Teacher record
ITID     DS    CL3          1- 3  Teacher ID nbr
ITNAME   DS    CL15         4-18  Teacher name
ITDEG    DS    CL4         19-22  Highest degree
ITTEN    DS    CL1         23-23  Tenured?
ITPHONE  DS    CL4         24-27  Phone nbr
ITCRLF   DS    CL2         28-29  PC/370 only - CR/LF
```

Let's look again at the `0CL29`. In an earlier chapter we said that the zero meant that this field (`IREC`) would be subdivided. The zero here is actually a **multiplier**, meaning there are zero occurances of 29 bytes, meaning `IREC` is defined as 29 bytes in length, but does not get any storage of its own. Rather, it simply overlaps those fields following it which occupy the next 29 bytes.

* *

The use of non-zero multipliers is quite common as well. For example, if I want a ten byte field containing all asterisks, I could code any of the following:

```
STARS    DC    CL10'**********'
STARS    DC    C'**********'
STARS    DC    10CL1'*'
STARS    DC    10C'*'
STARS    DC    5CL2'**'
STARS    DC    2CL5'*****'
```

You *cannot* assign a value (using `DC`) to a field defined with a zero multiplier. Also, if you attempt to assign a value with a `DS` rather than `DC`, you will not get an error message; but the value is ignored. That can be a very difficult error to find if you're not aware of it!

You Try It...

Which of the following will define a twenty-four byte field containing blanks?

4. `BLANKS  DC   CL24' '`
5. `BLANKS  DS   4CL6' '`
6. `BLANKS  DC   2CL12' '`
7. `BLANKS  DC   0CL24' '`
8. `BLANKS  DC   24C' '`

The desired report format is as follows:

```
          1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345678890
XXX   XXXXXXXXXXXXXXX   XXXX   X   XXXX
XXX   XXXXXXXXXXXXXXX   XXXX   X   XXXX
XXX   XXXXXXXXXXXXXXX   XXXX   X   XXXX
:     :                  :      :   :
:     :                  :      :   :... Phone
:     :                  :      :....... Tenured?
:     :                  :.............. Degree
:     :................................. Name
:...................................... ID number
```

Note the report is 60 characters wide. But recall from an earlier discussion that, when using PC/370, we must account for the carriage return/line feed as well. Consequently, our report will be 62 characters wide. Thus, the `LRECL` parameter of the `DCB` for `REPORT` will be `LRECL=62`.

```
OREC     DS    0CL62
OTID     DS    CL3               Teacher ID nbr
         DC    CL3' '
OTNAME   DS    CL15              Teacher name
         DC    CL3' '
OTDEG    DS    CL4               Highest degree
         DC    CL3' '
OTTEN    DS    CL1               Tenured?
         DC    CL3' '
OTPHONE  DS    CL4               Phone nbr
         DC    CL21' '
OCRLF    DS    CL2               PC/370 only - CR/LF
```

Note that the gaps between the fields have been given initial values of blanks. I could have initialized all of the fields to blanks (if I change the `DS`s to `DC`s), but it isn't necessary since those fields will not be used until after the input fields have been moved in. Even though each field is more than one byte long, a single blank is sufficient. When defining character data (`DC`, with type `C`), the field will be padded with blanks. This should not be confused with the `MVC` instruction, which does not pad!

I will add another section to the program: Miscellaneous field definitions. I like to group any work fields together. I often start the field names with `W`. I know that my use of PC/370 necessitates that I put a `CR/LF` at the end of each print line, so I have added a work field to the program for this purpose. This field is defined as:

```
*
*        Miscellaneous field definitions
*
WCRLF     DC    X'0D25'           PC/370 ONLY - EBCDIC CR/LF
```

The `X` indicates that this field is intended to hold hexadecimal data. The hexadecimal `0D` corresponds to an EBCDIC carriage return, and a hexadecimal `25` corresponds to an EBCDIC line feed. This same field will be used in most of our programs.

Having defined the fields, we can now move them. The MVCs to move the input or work fields to the corresponding output fields are:

```
MVC   OTID,ITID          Move teacher ID Nbr to output
MVC   OTNAME,ITNAME      Move teacher Name to output
MVC   OTDEG,ITDEG        Move highest degree to output
MVC   OTTEN,ITTEN        Move tenure to output
MVC   OTPHONE,ITPHONE    Move phone nbr to output
MVC   OCRLF,WCRLF        PC/370 ONLY - end line w/ CR/LF
```

Note that I used WCRLF rather than ITCRLF to move a carriage return/line feed to OCRLF. I could have used either one. I will always use WCRLF. Here's one reason: perhaps I am reading a file which does not have a CR/LF at the end of each record. For example, perhaps when I keyed in the data, I did not press the Enter key after each 29 bytes of data. It's okay to do so, and it does save space (two bytes per record). But I usually press Enter so I can view the data more easily using DOS' TYPE command, or something similar. Nevertheless, if I did not press Enter, there would be no ITCRLF field, and to produce the report I would have to make use of another field such as WCRLF.

You can see from the above discussion that I never reference ITCRLF in this program. I need to have the two bytes of storage allocated to hold it. Afterall, it is there, even if I don't use it. But if I choose to do so, I can leave off the name of the field since it is never used. For example, I could code:

```
IREC     DS    0CL29        1-29  Teacher record
ITID     DS    CL3           1- 3  Teacher ID nbr
ITNAME   DS    CL15          4-18  Teacher name
ITDEG    DS    CL4          19-22  Highest degree
ITTEN    DS    CL1          23-23  Tenured?
ITPHONE  DS    CL4          24-27  Phone nbr
         DS    CL2          28-29  PC/370 only - CR/LF
```

You will see many uses of this (omitting a field name) when we discuss creating report headings in the next chapter. (If you are familiar with COBOL, this is the BAL equivalent to a FILLER.) In this program, however, I would prefer to continue to use the label ITCRLF for the sake of completeness. It doesn't hurt anything, and has no effect on program execution time.

The complete program, TEACH2A.MLC, and its output follow.

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  TEACH2A.MLC                                 *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  A quick-and-dirty list of teachers.         *
*****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'TEACH2A ... Begin execution'
         OI    TEACHERS+10,X'08'  PC/370 ONLY - Convert all
*                                 input from ASCII to EBCDIC
         OI    REPORT+10,X'08'    PC/370 ONLY - Convert all
*                                 output from EBCDIC to ASCII
         OPEN  TEACHERS
         OPEN  REPORT
LOOP     GET   TEACHERS,IREC      Read a single teacher record
         MVC   OTID,ITID          Move teacher ID Nbr to output
         MVC   OTNAME,ITNAME      Move teacher Name to output
         MVC   OTDEG,ITDEG        Move highest degree to output
         MVC   OTTEN,ITTEN        Move tenure to output
         MVC   OTPHONE,ITPHONE    Move phone nbr to output
         MVC   OCRLF,WCRLF        PC/370 ONLY - end line w/ CR/LF
         PUT   REPORT,OREC        Write report line
         B     LOOP
*
*        EOJ processing
*
ATEND    CLOSE TEACHERS
         CLOSE REPORT
         WTO   'TEACH2A ... Teacher list on REPORT.TXT'
         WTO   'TEACH2A ... Normal end of program'
         RETURN
*
*        Literals, if any, will go here
*
         LTORG
*
*        File definitions
*
TEACHERS DCB   LRECL=29,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='TEACHER.DAT'
REPORT   DCB   LRECL=62,RECFM=F,MACRF=P,
               DDNAME='REPORT.TXT'
*
*        Miscellaneous field definitions
*
WCRLF     DC    X'0D25'            PC/370 ONLY - EBCDIC CR/LF
*
*        Input record definition
*
IREC     DS    0CL29              Teacher record
ITID     DS    CL3                Teacher ID nbr
ITNAME   DS    CL15               Teacher name
ITDEG    DS    CL4                Highest degree
ITTEN    DS    CL1                Tenured?
ITPHONE  DS    CL4                Phone nbr
ITCRLF   DS    CL2                PC/370 only - CR/LF
```

(continued)

```
*
*        Output (line) definition
*
OREC     DS    0CL62
OTID     DS    CL3                Teacher ID nbr
         DC    CL3' '
OTNAME   DS    CL15               Teacher name
         DC    CL3' '
OTDEG    DS    CL4                Highest degree
         DC    CL3' '
OTTEN    DS    CL1                Tenured?
         DC    CL3' '
OTPHONE  DS    CL4                Phone nbr
         DC    CL21' '
OCRLF    DS    CL2                PC/370 only - CR/LF
         END   BEGIN
```

```
A:\MIN>teach2a
TEACH2A ... Begin execution
TEACH2A ... Teacher list on REPORT.TXT
TEACH2A ... Normal end of program

A:\MIN>type report.txt
732   BENSON, E.T.      PHD    N   5156
218   HINCKLEY, G.B.    MBA    N   5509
854   KIMBALL, S.W.     PHD    Y   5594
626   YOUNG, B.         MBA    Y   5664
574   SMITH, J.         MS     Y   5320
```

Defining and Moving Alphanumeric Literals

In the previous section we have saw how to define and move character data. Specifically, we moved the fields of an input record to the corresponding fields of an output record. Here we will look at how to move character constants, or literals. The following examples will help to illustrate the need for doing so:

- A phone number stored as `XXXXXXX` is to be printed as `XXX-XXXX`. The hyphen (`-`) is the constant.
- A social security number stored as `XXXXXXXXX` is to be printed as `XXX-XX-XXXX`. Again, the hyphens are constant.
- A date stored as `YYMMDD` is to be printed as `MM/DD/19YY`. Here the slashes (`/`) and the `19` are constant.

We will show several ways to move literals. We will concentrate on the first and third examples. (The second example is very similar to the first and will be left as an exercise.)

* *

Example #1 - A phone number stored as `XXXXXXX` is to be printed as `XXX-XXXX`.

Let's first define the input and output fields as follows:

```
IPHONE   DS    0CL7
IPFX     DS    CL3
ILINE    DS    CL4

OPHONE   DS    0CL8
OPFX     DS    CL3
OHYPHEN  DS    CL1
OLINE    DS    CL4
```

Moving the prefix and line are no problem: we simply use the `MVC` instruction as we have already discussed it:

```
MVC   OPFX,IPFX
MVC   OLINE,ILINE
```

But what about moving the hyphen? One solution would be to define a new work field:

```
WHYPHEN  DC    CL1'-'
```

We can then code `MVC OHYPHEN,WHYPHEN`

There is nothing wrong with this method; it works. But there are better ways, both in terms of simplicity in coding, execution time, and memory. We can save ourselves some time in coding by using a constant, or literal, rather than defining a work field with a value of '-'. For example, we could code:

```
MVC   OHYPHEN,=CL1'-'
```

or

```
MVC   OHYPHEN,=C'-'
```

Note the equal sign *is* required. These two methods (using a defined field vs. using a literal) are illustrated in the next two programs, `MOVE2A.MLC` and `MOVE2B.MLC`:

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  MOVE2A.MLC                                  *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  Demonstrate character moves.                *
*****************************************************************
         START 0
BEGIN    BEGIN
         WTO   IPHONE
         MVC   OPFX,IPFX
         MVC   OHYPHEN,WHYPHEN
         MVC   OLINE,ILINE
         WTO   OPHONE
         RETURN
*
*        Literals if any will go here
*
         LTORG
```

(continued)

```
*
*        Other field definitions
*
WHYPHEN  DC    CL1'-'
*
IPHONE   DS    0CL7
IPFX     DC    CL3'555'
ILINE    DC    CL4'1212'
*
OPHONE   DS    0CL8
OPFX     DS    CL3
OHYPHEN  DS    CL1
OLINE    DS    CL4
         END   BEGIN
```

```
A:\MIN>move2a
5551212
555-1212
```

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  MOVE2B.MLC                                  *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  Demonstrate character moves.                *
*****************************************************************
         START 0
BEGIN    BEGIN
         WTO   IPHONE
         MVC   OPFX,IPFX
         MVC   OHYPHEN,=CL1'-'
         MVC   OLINE,ILINE
         WTO   OPHONE
         RETURN
*
*        Literals if any will go here
*
         LTORG
*
*        Other field definitions
*
IPHONE   DS    0CL7                    Note that the field
IPFX     DC    CL3'555'                definition for WHYPHEN
ILINE    DC    CL4'1212'               has been removed.
*
OPHONE   DS    0CL8
OPFX     DS    CL3
OHYPHEN  DS    CL1
OLINE    DS    CL4
         END   BEGIN
```

```
A:\MIN>move2b
5551212
555-1212
```

You Try It...

A phone number with area code stored as `XXXXXXXXXX` is to be printed as `(XXX)XXX-XXXX`

9. Define the input field.
10. Define the output field.
11. Write the instructions necessary to move the input field to the output field. Include the hyphen and parentheses.

* *

Many beginning programmers are under the (mistaken) impression that they can save memory by using literals instead of defining variables with the desired values. This is simply not true. These two examples are functionally equivalent. In fact, whenever a literal is coded in such a way, *the assembler generates a field definition*, just as if you had defined one and given it a name. These fields which are generated by the assembler are placed after the `LTORG` instruction (which we have already used but put off discussing.) We can see this by examining the `.PRN` files produced by the `A370` step. Check the `.PRN` files on the next page and notice:

- Both `MVC` instructions to move the hyphen occupy six bytes. This can be seen by looking at the object code, which is shown in hexadecimal and to the left of the instruction in the `.PRN` listing. In both cases it is equal to `D200D0ABD0A0`. These twelve hex digits occupy six bytes. This is the amount of memory required by *all* `MVC` instructions. (That's just the instruction, not the data.)

- When a literal was used (in `MOVE2B.MLC`), the assembler generated a field definition and place it after the `LTORG`. The result is that the same amount of memory was used for the hyphen: even though we reduced the memory requirements by one byte when we took out the definition for `WHYPHEN`, the assembler put back one byte for the literal.

- Finally, the total program length is the same: the address of the first byte of the last field in each program is `0000BC`.

Taken from MOVE2A.PRN

```
00007C D202D0A8D0A1   00B8   00B1   29          MVC      OPFX,IPFX
000082 D200D0ABD0A0   00BB   00B0   30          MVC      OHYPHEN,WHYPHEN
000088 D203D0ACD0A4   00BC   00B4   31          MVC      OLINE,ILINE


0000AA                             42 *
0000AA                             43 *        Literals if any will go here
0000AA                             44 *
0000B0                             45          LTORG
0000B0                             46 *
0000B0                             47 *        Other field definitions
0000B0                             48 *
0000B0 60                          49 WHYPHEN  DC       CL1'-'
0000B1                             50 *
0000B1                             51 IPHONE   DS       0CL7
0000B1 F5F5F5                      52 IPFX     DC       CL3'555'
0000B4 F1F2F1F2                    53 ILINE    DC       CL4'1212'
0000B8                             54 *
0000B8                             55 OPHONE   DS       0CL8
0000B8                             56 OPFX     DS       CL3
0000BB                             57 OHYPHEN  DS       CL1
0000BC                             58 OLINE    DS       CL4
000000                             59          END      BEGIN
```

Taken from MOVE2B.PRN

```
00007C D202D0A8D0A1   00B8   00B1   29          MVC      OPFX,IPFX
000082 D200D0ABD0A0   00BB   00B0   30          MVC      OHYPHEN,=CL1'-'
000088 D203D0ACD0A4   00BC   00B4   31          MVC      OLINE,ILINE


0000AA                             42 *
0000AA                             43 *        Literals if any will go here
0000AA                             44 *
0000B0                             45          LTORG
0000B0 60                          45                   CL1'-'
0000B1                             46 *
0000B1                             47 *        Other field definitions
0000B1                             48 *
0000B1                             49 IPHONE   DS       0CL7
0000B1 F5F5F5                      50 IPFX     DC       CL3'555'
0000B4 F1F2F1F2                    51 ILINE    DC       CL4'1212'
0000B8                             52 *
0000B8                             53 OPHONE   DS       0CL8
0000B8                             54 OPFX     DS       CL3
0000BB                             55 OHYPHEN  DS       CL1
0000BC                             56 OLINE    DS       CL4
000000                             57          END      BEGIN
```

The MVI Instruction

Whenever a character literal of length one is moved, you should use the MVI instruction instead of the MVC instruction. The MVI, or Move Immediate, instruction differs from the MVC instruction in several ways:

- With MVI, the move is always for a length of one, regardless of the length of the receiving field. (Recall that with MVC, the move is always determined by the length of the receiving field.)
- With MVI, you can move a literal only. (For example, I could not use the MVI to move the teacher's tenure status to the output area, even though that field is defined as one byte in length.)

The equivalent MVI instruction is `MVI OHYPHEN,C'-'`

Note that an equal sign is not used with MVI! Whereas with the MVC instruction, the equal sign was required, with the MVI instruction the equal sign is not allowed.

The use of the MVI instruction is illustrated in the next program, MOVE2C.MLC.

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  MOVE2C.MLC                                  *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  Demonstrate character moves.                *
*****************************************************************
         START 0
BEGIN    BEGIN
         WTO   IPHONE
         MVC   OPFX,IPFX
         MVI   OHYPHEN,C'-'
         MVC   OLINE,ILINE
         WTO   OPHONE
         RETURN
*
*        Literals if any will go here
*
         LTORG
*
*        Other field definitions
*
IPHONE   DS    0CL7
IPFX     DC    CL3'555'
ILINE    DC    CL4'1212'
*
OPHONE   DS    0CL8
OPFX     DS    CL3
OHYPHEN  DS    CL1
OLINE    DS    CL4
         END   BEGIN
```

```
A:\MIN>move2c
5551212
555-1212
```

If we examine the .PRN file, can compare it to the .PRN files produced by MOVE2A.MLC and MOVE2B.MLC we can see that:

- The MVI instruction to move the hyphen occupies four bytes only (vs. six bytes for the MVC instruction.) This can be seen by looking at the object code, which is shown in hexadecimal and to the left of the instruction in the .PRN listing. It is equal to 9260D0A2. These eight hex digits occupy four bytes. This is the amount of memory required by *all* MVI instructions.

- The value of the literal (in this case '-') is actually a part of the instruction. If we examine the object code more closely we see that the second byte ('60' in '9260D0A2') is the hexadecimal equivalent to a hyphen.

- When an MVI is used, the assembler does not generate a field to be placed after the LTORG. As we've just seen, there's no need to do so: the value is actually a part of the instruction.

- The program requires less memory. The address of the first byte of the last field in this program is 0000B3**.**

- Finally (and this is an important fact, but not apparent from the listings), the MVI instruction is faster executing than is the MVC instruction. Sure, it doesn't make any difference in these programs, but when you're processing fifty million records, such as in a targeted marketing application, the difference can be significant.

* *

Taken from MOVE2C.PRN

```
00007C D202D09FD098   00AF   00A8   29          MVC      OPFX,IPFX
000082 9260D0A2              00B2   30          MVI      OHYPHEN,C'-'
000086 D203D0A3D09B   00B3   00AB   31          MVC      OLINE,ILINE


0000A8                              42 *
0000A8                              43 *        Literals if any will go here
0000A8                              44 *
0000A8                              45          LTORG
0000A8                              46 *
0000A8                              47 *        Other field definitions
0000A8                              48 *
0000A8                              49 IPHONE   DS       0CL7
0000A8 F5F5F5                       50 IPFX     DC       CL3'555'
0000AB F1F2F1F2                     51 ILINE    DC       CL4'1212'
0000AF                              52 *
0000AF                              53 OPHONE   DS       0CL8
0000AF                              54 OPFX     DS       CL3
0000B2                              55 OHYPHEN  DS       CL1
0000B3                              56 OLINE    DS       CL4
000000                              57          END      BEGIN
```

Equated Values

Consider the following MVI instructions:

```
MVI   ELIGIBLE,C'Y'
MVI   TAXABLE,C'N'
MVI   GENDER,C'F'
MVI   OSLASH,C'/'
```

When you are moving a Y or N to a field, the meaning is pretty obvious. In those cases where you do not move a Y or N, the field name (such as GENDER) will often make the meaning of the immediate value (M or F) obvious. But what about the following:

```
MVI   RACE,C'1'
MVI   STATUS,C'D'
```

What does a RACE = '1' mean? What is STATUS, and what does a value of 'D' indicate? The following are much more meaningful:

```
MVC   RACE,HISPANIC        where...        HISPANIC DC    CL1'1'
MVC   STATUS,DIVORCED                      DIVORCED DC    CL1'D'
```

But we just said that we should use an MVI instead of an MVC to move constants of length one. So how can we use the MVI and still get the advantages of more meaningful code? The answer is by using the EQU instruction.

The EQU, or Equate, verb allows you to assign a name to an immediate value. You can then use that name as the second operand of an MVI (and other instructions as we will see later). To use an EQU with the above examples, we would code:

```
MVI   RACE,HISPANIC        where...        HISPANIC EQU   C'1'
MVI   STATUS,DIVORCED                      DIVORCED EQU   C'D'
```

Note that there can be no length operator on the EQU definition (that is, we use C'1' instead of CL1'1').

Equated values cannot be used as operands for an MVC.

For example, the following *will not work*:

```
MVC   RACE,HISPANIC        where...        HISPANIC EQU C'1'
MVC   STATUS,DIVORCED                      DIVORCED EQU C'D'
```

* *

You Try It...

Replace the `MVI` with `MVC` in each of the following:

12. `MVI  TYPE,C'A'` *where*
```
TYPE   DC   CL1' '
```
13. `MVI  EOFSW,TRUE` *where*
```
EOFSW  DC   CL1' '
TRUE   EQU  C'1'
FALSE  EQU  C'0'
```
14. `MVI  ERRSW,FOUND` *where*
```
ERRSW  DC   CL1' '
FOUND  EQU  C'*'
```

Given `SWITCH DC CL1' '` replace the following `MVCs` with `MVIs`:

```
MVC   SWITCH,=CL1' '        Turn switch off
MVC   SWITCH,=CL1'X'        Turn switch on
```

15. ...without `EQUs`
16. ...with `EQUs`

* * * * * * * * * * * * * * * * * * * *

Our earlier program, `MOVE2C.MLC` has been changed to use an equated value and includes:

```
MVI   OHYPHEN,HYPHEN       where...        HYPHEN  EQU   C'-'
```

The new program, `MOVE2D.MLC`, follows:

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  MOVE2D.MLC                                  *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  Demonstrate character moves.                *
*****************************************************************
         START 0
BEGIN    BEGIN
         WTO   IPHONE
         MVC   OPFX,IPFX
         MVI   OHYPHEN,HYPHEN
         MVC   OLINE,ILINE
         WTO   OPHONE
         RETURN
*
*        Literals if any will go here
*
         LTORG
*
*        Other field definitions
*
HYPHEN   EQU   C'-'
*
```

(continued)

```
IPHONE   DS    0CL7
IPFX     DC    CL3'555'
ILINE    DC    CL4'1212'
*
OPHONE   DS    0CL8
OPFX     DS    CL3
OHYPHEN  DS    CL1
OLINE    DS    CL4
         END   BEGIN
```

```
A:\MIN>move2d
5551212
555-1212
```

If we examine the `.PRN` file and compare it to the `.PRN` files produced by `MOVE2C.MLC` we can see that:

- The new `MVI` instruction, using the equated value, created exactly the same object code as the earlier `MVI` instruction; that is, `9260D0A2`.
- The hexadecimal equivalent of a hyphen (`'60'`) is shown to the left of the `EQU` statement. This is the same `'60'` as appears in the instruction.

Taken from `MOVE2D.PRN`

```
00007C D202D09FD098   00AF   00A8   29          MVC      OPFX,IPFX
000082 9260D0A2              00B2   30          MVI      OHYPHEN,HYPHEN
000086 D203D0A3D09B   00B3   00AB   31          MVC      OLINE,ILINE


0000A8                               42 *
0000A8                               43 *        Literals if any will go here
0000A8                               44 *
0000A8                               45          LTORG
0000A8                               46 *
0000A8                               47 *        Other field definitions
0000A8                               48 *
0000A8              00000060         49 HYPHEN   EQU      C'-'
0000A8                               50 *
0000A8                               51 IPHONE   DS       0CL7
0000A8 F5F5F5                        52 IPFX     DC       CL3'555'
0000AB F1F2F1F2                      53 ILINE    DC       CL4'1212'
0000AF                               54 *
0000AF                               55 OPHONE   DS       0CL8
0000AF                               56 OPFX     DS       CL3
0000B2                               57 OHYPHEN  DS       CL1
0000B3                               58 OLINE    DS       CL4
000000                               59          END      BEGIN
```

* *

Explicit Length and Displacement

Recall that the length of an MVC instruction is determined by the length of the receiving field. We've already looked at what can happen if the sending and receiving fields are of different lengths. Consequently, we have been careful to define our receiving fields (on report output) to be the same length as the sending fields. But this may not always be possible. For example, you may have a program which has two input files (call them A and B) and one output file (call it C). The name field (containing first and last name) is defined as 25 bytes on files A and C, but as 20 bytes on file B. Sometimes C will get the name from A, while other times it will get the name from B.

Moving the name from A to C is easy (`MVC CNAME,ANAME`), but such is not the case for moving the name from B to C. We cannot use `MVC CNAME,BNAME` because since the name on C is five bytes longer than the name on B, the five bytes immediately following the name in B will be moved as well. What we need is some way to override the length of the move. There is a simple way.

On any MVC, we can state an **explicit length**. This length overrides the default length which is the length of the receiving field. For example `MVC CNAME(20),BNAME` will move twenty bytes only. The problem with this is that bytes 21 through 25 of CNAME will remain unchanged; that is, if the field (or record) has not been cleared, data from ANAME may be leftover in those bytes. One way to avoid this is to clear the field (or record) before any moving is done. For example `MVC CNAME,=CL25'b'`

In all of our examples, we have assumed that the data we will move is to be placed in the first byte of the field. That is to say that there is a default displacement, or skip, of zero. This is usually, but not always, the case. Just as we can give an explicit length, we can also give an **explicit displacement**. By doing so, I will not need to clear the entire field (or record) as shown above. For example, if I am moving the name from file A, I can code:

```
MVC   CNAME,ANAME
```

as before, but if I am moving the name from file B, I will code

```
MVC   CNAME(20),BNAME
MVC   CNAME+20(5),=CL5'b'
```

What if the street address field is defined as 30 bytes on files A and C, but as 28 bytes on file B? Again, sometimes C will get the address from A, while other times it will get the address from B. As before, if I am moving the address from file A, I can code:

```
MVC   CSTREET,ASTREET
```

but if I am moving the address from file B, I will code

```
MVC   CSTREET(28),BSTREET
MVC   CSTREET+28(2),=CL2'b'
```

The excessive use of literals can make larger programs more difficult to maintain. Rather than have many literals of the type `=CL5'b'` and `=CL2'b'`, etc., it is probably better to define a single field called `BLANKS` or `SPACES`, of sufficient length, and to be used instead of these literals. For example, I might code:

```
BLANKS   DC   CL30'b'
```

and then code

```
MVC   CNAME,BLANKS
MVC   CSTREET,BLANKS
```

or (when moving data from file `B`)

```
MVC   CNAME(20),BNAME
MVC   CNAME+20(5),BLANKS
MVC   CSTREET(28),BSTREET
MVC   CSTREET+28(2),BLANKS
```

Note that an explicit displacement can be used with `MVI` as well. However, **an explicit length cannot be used with `MVI`**, even if the stated length is 1! For example, to move an asterisk to the third byte of `JUNK`, I can code:

```
MVC   JUNK+2(1),=CL1'*'
MVI   JUNK+2,C'*'
MVI   JUNK+2,STAR          (where STAR EQU C'*')
```

but I cannot code:

```
MVI   JUNK+2(1),C'*'
```

Reminder: equated values cannot be used as operands for an `MVC`, and therefore the following will not work:

```
MVC   JUNK+2(1),STAR
```

When using `MVC`, explicit displacement can be used on both the sending and receiving fields, but explicit length can be used on the receiving field only.

When using `MVI`, explicit displacement can be used on both the sending and receiving fields, but explicit length cannot be used at all.

This method of moving, with explicit displacement and length is sometimes abused. Some programmers will write whole programs using this method, so as to avoid the time and effort needed to code the record layout. But this is a dangerous practice, as it is very difficult to debug. If that were the only negative, it wouldn't be so bad, because the programmer is only hurting himself or herself. But these programs then get passed onto other programmers who must maintain them. Use explicit displacement and length sparingly. Its use *may* be appropriate when referring to a part of a field, but rarely would it be appropriate when referring to a part of a record.

The following program, `TEACH2B.MLC`, is functionally equivalent to program `TEACH2A.MLC` shown earlier, but uses explicit displacement and length only. Which would you rather maintain?

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  TEACH2B.MLC                                 *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  Don't do it this way!!!                     *
*****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'TEACH2B ... Begin execution'
         OI    TEACHERS+10,X'08'  PC/370 ONLY - Convert all
*                                 input from ASCII to EBCDIC
         OI    REPORT+10,X'08'    PC/370 ONLY - Convert all
*                                 output from EBCDIC to ASCII
         OPEN  TEACHERS
         OPEN  REPORT
LOOP     GET   TEACHERS,IREC
         MVC   OREC(3),IREC
         MVC   OREC+6(15),IREC+3
         MVC   OREC+24(4),IREC+18
         MVC   OREC+31(1),IREC+22
         MVC   OREC+35(4),IREC+23
         MVC   OREC+60(2),=X'0D25'
         PUT   REPORT,OREC        Write report line
         B     LOOP
*
*        EOJ processing
*
ATEND    CLOSE TEACHERS
         CLOSE REPORT
         WTO   'TEACH2B ... Teacher list on REPORT.TXT'
         WTO   'TEACH2B ... Normal end of program'
         RETURN
*
*        Literals, if any, will go here
*
         LTORG
*
*        File definitions
```

(continued)

```
*
TEACHERS DCB   LRECL=29,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='TEACHER.DAT'
REPORT   DCB   LRECL=62,RECFM=F,MACRF=P,
               DDNAME='REPORT.TXT'
*
*        Field definitions
*
IREC     DS    CL29              Teacher record
OREC     DC    CL62' '           Report line
         END   BEGIN
```

* *

Example #2 - A social security number stored as `XXXXXXXXX` is to be printed as `XXX-XX-XXXX`.

This example is similar to Example #1 above and is left as an exercise.

* *

Example #3 - A date stored as `YYMMDD` is to be printed as `MM/DD/19YY`.

- Defining all fields, and using `MVCs` only, without literals.

```
MVC   OMM,IMM
MVC   OSLASH1,SLASH
MVC   ODD,IDD
MVC   OSLASH2,SLASH
MVC   O19,NINETEEN
MVC   OYY,IYY
```

where

```
IDATE    DS    0CL6
IYY      DS    CL2
IMM      DS    CL2
IDD      DS    CL2
ODATE    DS    0CL10
OMM      DS    CL2
OSLASH1  DS    CL1
ODD      DS    CL2
OSLASH2  DS    CL1
O19      DS    CL2
OYY      DS    CL2
SLASH    DC    CL1'/'
NINETEEN DC    CL2'19'
```

- Defining all fields, but using `MVCs` and `MVIs` with literals.

```
MVC   OMM,IMM
MVI   OSLASH1,C'/'
MVC   ODD,IDD
MVI   OSLASH2,C'/'
MVC   O19,=CL2'19'
MVC   OYY,IYY
```

where all fields are defined as before.

- Using explicit displacement and length only, and using equated values.

```
MVC   ODATE(2),IDATE+2
MVI   ODATE+2,SLASH
MVC   ODATE+3(2),IDATE+4
MVI   ODATE+5,SLASH
MVC   ODATE+6(2),=CL2'19'
MVC   ODATE+8(2),IDATE
```

where

```
IDATE    DS    CL6
ODATE    DS    CL10
SLASH    EQU   C'/'
```

You Try It...

Given `A DC CL8' '` and `B DC CL6' '`. The field `A` contains a date stored in `MM-DD-YY` format. Move `A` to `B` such that `B` contains that date in `YYMMDD` format...

17. ...defining all fields and using `MVC`s only.
18. ...using explicit length and displacement only.
19. Repeat You Try It exercise 11 using explicit length and displacement only.

Adding Report and Column Headings

The earlier list of the records in the TEACHER file was referred to as a "quick and dirty" list: we said that by "quick-and-dirty" we mean a report without headings, page numbers, etc. Our purpose here is to add report and column headings to the list of teachers.

We will not do page numbers at this time. Page numbers may seem like an easy thing to you if you already know some other language, but such is not the case in assembler. They do, after all, require arithmetic (i.e., add 1 to page counter) and printing the results of arithmetic is not a trivial thing in `BAL`. Our new listing will appear as follows:

```
         1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345678901234567890
                    LIST OF TEACHERS

 ID#         Name          Degr  Ten  Phone
 ---   ---------------     ----  ---  -----
XXX    XXXXXXXXXXXXXXX     XXXX   X   XXXX
XXX    XXXXXXXXXXXXXXX     XXXX   X   XXXX
XXX    XXXXXXXXXXXXXXX     XXXX   X   XXXX
```

We will use `DS` and `DC` to define the headings. Note:

- There are four heading lines (the second is all blanks),
- Each must be defined as 62 bytes since `LRECL=62`, and
- Each heading line must end with `CR/LF` (PC/370 only).

There are many ways to format headings. The choice is simply a matter of personal preference. I will show several of the more common methods, as well as the method I prefer. I will focus on the third heading line: the techniques used will apply to the other headings as well. I usually label my headings as `HD1`, `HD2`, etc. One technique for defining the above (third) heading is as follows:

```
HD3      DS    0CL62
         DC    CL3'ID#'                    3
         DC    CL8' '                      8
         DC    CL4'Name'                   4
         DC    CL9' '                      9
         DC    CL4'Degr'                   4
         DC    CL2' '                      2
         DC    CL3'Ten'                    3
         DC    CL2' '                      2
         DC    CL5'Phone'                  5
         DC    CL20' '                    20
         DC    XL2'0D25'                   2
                                           62
```

Note:

- More than one `DC` is used to define the heading, so `HD3` is a `DS` (not `DC`) with a multiplier of zero,
- The sum of the field lengths is 62, and
- Positions 61-62 of the heading are defined as a `CR/LF`.

I will occasionally use the above method, but when I do, I prefer to include the print positions as comments. This improves the maintainability of the program.

```
HD3      DS    0CL62
         DC    CL3'ID#'       1- 3         3
         DC    CL8' '         4-11         8
         DC    CL4'Name'     12-15         4
         DC    CL9' '        16-24         9
         DC    CL4'Degr'     25-28         4
         DC    CL2' '        29-30         2
         DC    CL3'Ten'      31-33         3
         DC    CL2' '        34-35         2
         DC    CL5'Phone'    36-40         5
         DC    CL20' '       41-60        20
         DC    XL2'0D25'     61-62         2
                                           62
```

Even though the spacing between columns is more than one byte wide, a single blank was used as the value for each of those `DC`s. This is because, as we said earlier, when defining character data (`DC`, with type `C`), the field will be padded with blanks; that is, the following are equivalent:

`DC  CL8'bbbbbbbb'` *is equivalent to* `DC  CL8'b'`

Many programmers will take this one step further, realizing that since this is the case, one could include that length in the length of the preceding field, thereby omitting the blank DCs entirely:

```
HD3      DS    0CL62
         DC    CL11'ID#'      1-11                  11
         DC    CL13'Name'    12-24                  13
         DC    CL6'Degr'     25-30                   6
         DC    CL5'Ten'      31-35                   5
         DC    CL25'Phone'   36-60                  25
         DC    XL2'0D25'     61-62                   2
                                                    62
```

I prefer to break down my headings into blocks of 40 characters. This size will easily fit on a single line, and since most print layout charts have grid lines every 10 characters, it is a simple task to transcribe the headings from the print layout chart to actual assembler code. For example:

```
HD3      DS    0CL62
         DC    CL40'ID#        Name         Degr  Ten  Phone'
         DC    CL20' '
         DC    XL2'0D25'
```

The benefit of this technique is more apparent when there are multiple heading lines, each over 80 characters wide. (Most mainframe reports are designed to be 132 characters wide, exclusive of the carriage control character. This is discussed in the appendix.)

The complete headings definitions are as follows:

```
*
*        Headings definitions
*
HD1      DS    0CL62
         DC    CL40'            LIST OF TEACHERS           '
         DC    CL20' '
         DC    XL2'0D25'
HD2      DS    0CL62
         DC    CL60' '
         DC    XL2'0D25'
HD3      DS    0CL62
         DC    CL40'ID#        Name         Degr  Ten  Phone'
         DC    CL20' '
         DC    XL2'0D25'
HD4      DS    0CL62
         DC    CL40'---   ---------------   ----  ---  -----'
         DC    CL20' '
         DC    XL2'0D25'
```

After the output file (REPORT) is opened, and before any records are read, each of the heading lines is written. The PUT command is used for this purpose. The new program is TEACH2C.MLC: the program and its output follow:

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  TEACH2C.MLC                                 *
*        AUTHOR  :  Bill Qualls                                 *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  List of teachers, with headings.            *
*****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'TEACH2C ... Begin execution'
         OI    TEACHERS+10,X'08'  PC/370 ONLY - Convert all
*                                 input from ASCII to EBCDIC
         OI    REPORT+10,X'08'    PC/370 ONLY - Convert all
*                                 output from EBCDIC to ASCII
         OPEN  TEACHERS
         OPEN  REPORT
         PUT   REPORT,HD1
         PUT   REPORT,HD2
         PUT   REPORT,HD3
         PUT   REPORT,HD4
LOOP     GET   TEACHERS,IREC      Read a single teacher record
         MVC   OTID,ITID          Move teacher ID Nbr to output
         MVC   OTNAME,ITNAME      Move teacher Name to output
         MVC   OTDEG,ITDEG        Move highest degree to output
         MVC   OTTEN,ITTEN        Move tenure to output
         MVC   OTPHONE,ITPHONE    Move phone nbr to output
         MVC   OCRLF,WCRLF        PC/370 ONLY - end line w/ CR/LF
         PUT   REPORT,OREC        Write report line
         B     LOOP
*
*        EOJ processing
*
ATEND    CLOSE TEACHERS
         CLOSE REPORT
         WTO   'TEACH2C ... Teacher list on REPORT.TXT'
         WTO   'TEACH2C ... Normal end of program'
         RETURN
*
*        Literals, if any, will go here
*
         LTORG
*
*        File definitions
*
TEACHERS DCB   LRECL=29,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='TEACHER.DAT'
REPORT   DCB   LRECL=62,RECFM=F,MACRF=P,
               DDNAME='REPORT.TXT'
*
*        Miscellaneous field definitions
*
WCRLF    DC    X'0D25'            PC/370 ONLY - EBCDIC CR/LF
```

(continued)

```
*
*        Input record definition
*
IREC     DS    0CL29              Teacher record
ITID     DS    CL3                Teacher ID nbr
ITNAME   DS    CL15               Teacher name
ITDEG    DS    CL4                Highest degree
ITTEN    DS    CL1                Tenured?
ITPHONE  DS    CL4                Phone nbr
ITCRLF   DS    CL2                PC/370 only - CR/LF
*
*        Output (line) definition
*
OREC     DS    0CL62
OTID     DS    CL3                Teacher ID nbr
         DC    CL3' '
OTNAME   DS    CL15               Teacher name
         DC    CL3' '
OTDEG    DS    CL4                Highest degree
         DC    CL3' '
OTTEN    DS    CL1                Tenured?
         DC    CL3' '
OTPHONE  DS    CL4                Phone nbr
         DC    CL21' '
OCRLF    DS    CL2                PC/370 only - CR/LF
*
*        Headings definitions
*
HD1      DS    0CL62
         DC    CL40'            LIST OF TEACHERS            '
         DC    CL20' '
         DC    XL2'0D25'
HD2      DS    0CL62
         DC    CL60' '
         DC    XL2'0D25'
HD3      DS    0CL62
         DC    CL40'ID#        Name          Degr  Ten  Phone'
         DC    CL20' '
         DC    XL2'0D25'
HD4      DS    0CL62
         DC    CL40'---   ---------------   ----  ---  -----'
         DC    CL20' '
         DC    XL2'0D25'
         END   BEGIN
```

```
A:\MIN>teach2c
TEACH2C ... Begin execution
TEACH2C ... Teacher list on REPORT.TXT
TEACH2C ... Normal end of program

A:\MIN>type report.txt
            LIST OF TEACHERS

ID#        Name          Degr  Ten  Phone
---   ---------------   ----  ---  -----
732   BENSON, E.T.      PHD    N   5156
218   HINCKLEY, G.B.    MBA    N   5509
854   KIMBALL, S.W.     PHD    Y   5594
626   YOUNG, B.         MBA    Y   5664
574   SMITH, J.         MS     Y   5320
```

Exercises

1. True or false.

   T F a. The instruction `MVC FLDA,FLDB` will move (copy) `FLDA` to `FLDB`.
   T F b. The number of characters moved with a single `MVC` is determined by the length of the sending field unless overridden.
   T F c. A maximum of 256 characters can be moved with a single `MVC`.
   T F d. When using an `MVC`, if the receiving field is shorter than the sending field, the rightmost characters are truncated (not moved).
   T F e. When using an `MVC`, if the receiving field is longer than the sending field, the extra (rightmost) bytes of the receiving field are padded with blanks.
   T F f. `DOLLARS DC 5CL4'$$$'` will allocate a total of 15 bytes.
   T F g. The `MVC` and `MVI` instructions occupy the same amount of memory when each is moving a one byte field.
   T F h. Explicit displacement can be used with `MVC` and `MVI`.
   T F i. Explicit length can be specified on an `MVI` only if that length is one.
   T F j. The use of equated values can make `MVIs` more readable.
   T F k. Given `SEVEN DC CL1'7'`, an `MVI` should be used instead of an `MVC` to move `SEVEN` to another one-byte field.
   T F l. The following are equivalent: `X DC CL5'ABC'` and `X DC CL5'ABCbb'`
   T F m. All `DSs` and `DCs` require field names (labels).

2. Given the following adjacent field definitions, determine the result for each of the following instructions. Start with new data for each question.

```
A   DC   CL5'JKLMN'
B   DC   CL4'PQR '
C   DC   CL3'ST'
X   DC   CL1'W'
Y   EQU  C'Z'
```

```
(a)   MVC B,A              B will be:
(b)   MVC A,B              A will be:
(c)   MVC B,C              B will be:
(d)   MVC A+2(2),C         A will be:
(e)   MVC A+3(2),B+3       A will be:
(f)   MVC C(1),X           C will be:
(g)   MVC C(1),=C'X'       C will be:
(h)   MVI B+2,C'Y'         B will be:
(i)   MVI B+2,Y            B will be:
```

Exercises

3. Show how equates might be used to make the following statements more meaningful:

```
a.      MVI   SEX,C'M'      b.     MVI   DAYOFWK,C'1'  Sunday
        MVI   SEX,C'F'             MVI   DAYOFWK,C'2'  Monday
                                                :
                                   MVI   DAYOFWK,C'7'  Saturday
```

4. Using the examples given in this chapter (MOVE2A.MLC, MOVE2B.MLC, MOVE2C.MLC, and MOVE2D.MLC), write and execute four BAL programs to demonstrate the different methods by which you could complete the moves described in Example #2: A social security number stored as XXXXXXXXX is to be printed as XXX-XX-XXXX. Look at the resulting .PRN files. Discuss the differences and similarities in terms of ease of coding, execution time, memory usage, and maintainability.

5. In direct marketing (aka junk mail) one common task is the merge/purge. In the merge/purge, multiple lists (such as mailing lists from different mail order houses and list brokers) are combined, and duplicates are dropped so as to minimize printing and postage costs. In order to identify duplicates, a match code is usually generated. There are many ways of generating a match code. For example, the match code may consist of the zip code, the first, third, and fourth letters of the last name, and the first (up to six) consecutive numbers from the street address.

   Thus, given:
   ```
   KATHY BLACK
   618 S ANZA
   PASADENA CA 91106
   ```

   The match code would be: `91106BAC000618`

   Note the following names would give the same match code: KATHI BLACK, CATHY BLACK, K. BLACK, and C.J. BLACK.

   Your task is as follows: The input record is 80 bytes long, with the zip code in positions 1-5, the last name in positions 21-32, and the street address in positions 33-62. All other positions are unused in this example. The match code is 14 bytes long as illustrated above. You are to code the DS and MVC necessary to move the zip code to positions 1-5 of the match code, and to move positions 1, 3, and 4 of the last name to positions 6, 7, and 8 (respectively) of the match code. (We will do nothing with the street address in this example.)

   Challenge - Code the DS and MVC such that the name can be moved with two MVCs.

Exercises

6. Produce a formatted list of the records in the student file. Each line should be 33 bytes long, exclusive of the CR/LF. The report should include report and column headings. The desired format is as follows:

```
          1         2         3
12345678901234567890123456789012345
       STUDENT MASTER LIST

ID#    Student Name      Sex   Mar
---   ---------------    ---   ---
XXX   XXXXXXXXXXXXXXX     X     X
XXX   XXXXXXXXXXXXXXX     X     X
XXX   XXXXXXXXXXXXXXX     X     X
```

7. Produce a formatted list of the records in the GRADE file. Show the semester, course ID, student ID, and grade earned only, in that order. Each line should be 50 bytes long, exclusive of the CR/LF. The report should include report and column headings. The desired format is as follows:

```
          1         2         3         4         5
12345678901234567890123456789012345678901234567890
                 GRADE MASTER LIST
                  (Confidential)

              Course   Student
         Sem  Number     ID#     Grade
         ---  ------   -------   -----
         XXX  XXXXX      XXX       X
         XXX  XXXXX      XXX       X
         XXX  XXXXX      XXX       X
```

8. Produce a formatted list of the records in the course file. Show the course ID and course description only. Show the course number as the department (first two positions of the course ID) and the course number (third, fourth, and fifth positions of the course ID) are separated by a single space. Each line should be 30 bytes long, exclusive of the CR/LF. The report should include report and column headings. The desired format is as follows:

```
          1         2         3
123456789012345678901234567890
        COURSE MASTER LIST

     Course    Description
     ------  ---------------
     XX XXX  XXXXXXXXXXXXXXX
     XX XXX  XXXXXXXXXXXXXXX
     XX XXX  XXXXXXXXXXXXXXX
```

Exercises

9. The following program makes (excessive) use of explicit length and displacement to produce a list of selected fields from the course offerings file. Run the program as is to determine the output. Then make the necessary changes to clean up the code while still producing the same results. *Your solution should not use explicit length and displacement at all!*

```
         PRINT NOGEN
*****************************************************************
*        FILENAME:  OFFER2A.MLC                                 *
*        AUTHOR  :                                              *
*        SYSTEM  :  PC/370 R4.2                                 *
*        REMARKS :  A quick-and-dirty list of offerings.        *
*****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         OI    OFFER+10,X'08'
         OI    REPORT+10,X'08'
         OPEN  OFFER
         OPEN  REPORT
LOOP     GET   OFFER,IREC
         MVC   OREC(2),IREC+3
         MVI   OREC+2,C'-'
         MVC   OREC+3(3),IREC+5
         MVI   OREC+6,C'-'
         MVC   OREC+7(1),IREC+8
         MVC   OREC+11(1),IREC
         MVC   OREC+12(3),=C'*19'
         MVC   OREC+15(2),IREC+1
         MVC   OREC+20(4),IREC+12
         MVC   OREC+27(3),IREC+9
         MVC   OREC+31(2),=C'**'
         MVC   OREC+33(2),=X'0D25'
         PUT   REPORT,OREC
         B     LOOP
ATEND    CLOSE OFFER
         CLOSE REPORT
         RETURN
         LTORG
OFFER    DCB   LRECL=18,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='OFFER.DAT'
REPORT   DCB   LRECL=35,RECFM=F,MACRF=P,
               DDNAME='OFFER.TXT'
IREC     DS    CL18
OREC     DC    CL35' '
         END   BEGIN
```