Chapter 7
Packed Decimal Arithmetic

Objectives

Upon completion of this chapter you will be able to:

- Given a field in hexadecimal, identify the zone and numeric bits,
- Given a zoned decimal number, show its hexadecimal representation,
- Given a zoned decimal number, pack that number and show the hexadecimal representation of that packed number,
- Given a packed number, unpack that number and show the hexadecimal representation of that unpacked number,
- Use the `MVZ` instruction to remove the sign from an unpacked number,
- Use the `AP` instruction to add one packed number to another,
- Use the `SP` instruction to subtract one packed number from another,
- Use the `ZAP` instruction to move one packed number to another,
- Use the `CP` instruction to compare two packed numbers,
- Show how the `BC` instruction can be used to detect an overflow condition following an add or subtract operation, and
- Write a program requiring the use of packed decimal arithmetic and the display of the results of that arithmetic.

Introduction

Up to now, none of our programs have used arithmetic. There are only two ways to do arithmetic in `BAL`: packed decimal arithmetic or binary (register) arithmetic. In this chapter we introduce packed decimal arithmetic; specifically, addition and subtraction. We will discuss multiplication and division in a later chapter. We also put off our discussion of binary arithmetic, because you need to understand packed numbers before you can work with binary numbers (a binary number must be converted to packed before it can be printed.)

Introducing Cogsworth Industries

For this chapter we will introduce a new file. (The files of the Small Town Community College database which we have used to this point do not contain sufficient numeric fields to introduce arithmetic.) This file is the inventory file for Cogsworth Industries. Cogsworth sells three products (Gizmos, Widgets, and Junque) in four states (California, Illinois, Utah, and Wisconsin). The record layout for the inventory file, `COGS.DAT`, is shown on the next page. The data file is constructed as follows:

```
A:\>copy con cogs.dat
GIZMOS    02003002002001709902312252999
WIDGETS   01501001000202203401900110025
JUNQUE    02501501501803005201001550339
^Z
        1 File(s) copied
```

Field Nbr	Field Name	Description	Begins	Ends	Len	Format
1	DESC	Product desc	1	10	10	CH
2	CALIF	Calif sales	11	13	3	ZD
3	ILL	Illinois sales	14	16	3	ZD
4	UTAH	Utah sales	17	19	3	ZD
5	WISC	Wisconsin sales	20	22	3	ZD
6	BEGIN	Beginning inv.	23	25	3	ZD
7	PURCH	Purchases	26	28	3	ZD
8	QOH	Qty on hand	29	31	3	ZD
9	COST	Cost (each)	32	35	4	99V99
10	SELL	Sell for (each)	36	39	4	99V99
11	CRLF	PC/370 Only	40	41	2	CR/LF

```
A:\>dir cogs.dat

 Volume in drive A has no label
 Directory of  A:\

COGS     DAT      123   9-29-93   9:38a
        1 File(s)    130560 bytes free
```

In this chapter we will create two reports. The first report, the Sales Recap, will appear as follows:

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
                 COGSWORTH INDUSTRIES
                      Sales Recap

Product       Calif     Ill      Utah     Wisc    TOTAL
----------    -----    -----    -----    -----    -----
GIZMOS         020      030      020      020      090
WIDGETS        015      010      010      002      037
JUNQUE         025      015      015      018      073

003 records processed.
```

We need to know how to do addition in order to produce the `TOTAL` column and record count.

Packed Decimal Format

Packed decimal is a storage format unique to the IBM System/370 computers and compatibles: you won't find packed decimal format on PCs, except when emulating such a mainframe, as with PC/370.

In zoned decimal (unpacked) format, each digit occupies a single byte. For example, California sales of `JUNQUE` (above) is `C'025'`, the hexadecimal representation of which is `X'F0F2F5'`. Each byte consists of two parts: the **zone** (or **sign**) portion (all `F` in this case), and the **numeric** portion (`0`, `2`, and `5`). Each portion occupies four bits.

If we rewrite this number as:

```
| F | F | F | <-- ZONE
| 0 | 2 | 5 | <-- NUMERIC
```

...then we can see why the zone portion is sometimes thought of as the **upper half byte** and the numeric portion is sometimes thought of as the **lower half byte**.

The *rightmost* zone bits (in this case the `F` above the `5`) determine the sign of a number. The other `F`'s are redundant. The sign is determined as follows:

- `F` indicates the number is unsigned,
- `C` indicates the number is positive, and
- `D` indicates the number is negative.

(If you know `COBOL`, you will recall that an unsigned number corresponds to a `PIC 9` clause *without* an `S`, such as `PIC 9(3)`, whereas a signed number corresponds to a `PIC 9` clause *with* an `S`, such as `PIC S9(3)`.)

Examples:

- `X'F1F2F3'` is `123`, unsigned,
- `X'F4F5C6'` is `+456`, and
- `X'F7F8D9'` is `-789.`

We need to understand how each of these would appear if printed,or if displayed with `WTO`:

- `X'F1F2F3'` would print as `123`,
- `X'F4F5C6'` would print as `45F`, because `X'C6'` is `C'F'`, and
- `X'F7F8D9'` would print as `78R`, because `X'D9'` is `C'R'.`

It should be apparent why we will need to make adjustments for the sign when printing a number.

<u>You Try It...</u>

1. Given `A DC CL3'68P'` is a zoned decimal number. What is the numeric value of `A`? Is it signed? What would `WTO A` display?
2. Given `B DC X'F2C6'` is a zoned decimal number. What is the numeric value of `B`? Is it signed? What would `WTO B` display?
3. Given `C DC CL3'-39'`. Show the hexadecimal representation of `C`. Show why `C` is *not* a valid zoned decimal number.
4. Given `D DC CL3'39'`. Show the hexadecimal representation of `D`. Show why `D` is *not* a valid zoned decimal number.

Again, the rightmost zone bits determine the sign; the other F's are redundant. When a number is "packed", these extra zone bits are removed, and the rightmost zone bits (only) are retained. For example, we can pack X'F0F2F5' into X'025F'. Since the extra zone bits are removed when a number is packed, the packed number occupies less space, except in the case of a zoned decimal number of length 1 and 2, which will still occupy the same amount of space (1 and 2 bytes, respectively).

Consider the following examples:

- We can pack the number 1 (unsigned), which occupies one byte and is represented as X'F1', into one byte with a value of X'1F'.
- We can pack the number +12, which occupies two bytes and is represented as X'F1C2', into two bytes with a value of X'012C'.
- We can pack the number -123, which occupies three bytes and is represented as X'F1F2D3', into two bytes with a value of X'123D'.
- We can pack the number 1234 (unsigned), which occupies four bytes and is represented as X'F1F2F3F4', into three bytes with a value of X'01234F'.
- We can pack the number +12345, which occupies five bytes and is represented as X'F1F2F3F4C5', into three bytes with a value of X'12345C'.

We can summarize the amount of space occupied as follows:

A zoned decimal field of length...	**requires a packed decimal field of length...**
1	1
2	2
3	2
4	3
5	3
m (where m is odd)	(m+1) / 2
n (where n is even)	(n/2) + 1

You Try It...

5. A zoned decimal field of length seven (that is, containing seven digits) will require a packed decimal field of what length?
6. A packed decimal field of length seven will hold how many digits?

The PACK Instruction

The BAL instruction to pack a number is, appropriately, PACK. The code to implement each of the above examples is as follows:

- To pack the number 1, which occupies one byte, into one byte:

```
PACK  B,A          where        A   DC  CL1'1'
                                B   DS  PL1
```

- To pack the number +12, which occupies two bytes, into two bytes:

```
PACK  B,A          where        A   DC  CL2'1B'
                                B   DS  PL2
```

- To pack the number -123, which occupies three bytes, into two bytes:

```
PACK  B,A          where        A   DC  CL3'12L'
                                B   DS  PL2
```

- To pack the number 1234, which occupies four bytes, into three bytes:

```
PACK  B,A          where        A   DC  CL4'1234'
                                B   DS  PL3
```

- To pack the number +12345, which occupies five bytes, into three bytes:

```
PACK  B,A          where        A   DC  CL5'1234E'
                                B   DS  PL3
```

The field type specifier P, as in PL3, indicates a packed field. It is important to understand that the type specifier is usually* significant only when assigning an initial value to a field with a DC. We can PACK a number into a field defined as character (C), hexadecimal (X), or any other. Also, the field length refers to the number of bytes used for that field, not to the number of digits it holds. So, for example, a CL3 field can hold a packed number of up to five digits.

When the field type specifier P is used with a DC, the initial value is always signed. For example:

- J DC PL2'12' has a hexadecimal representation of X'012C',
- K DC PL2'+12' *also* has a hexadecimal representation of X'012C', and
- L DC PL2'-12' has a hexadecimal representation of X'012D'.

A number *may* be PACKed into a field which is too small to hold the number: the high order (left most) digits will be truncated, and no warning or error messages are given. So be careful! A number *may also* be PACKed into a field which is larger than necessary to hold that number. In this case, the number is padded with zeroes to the left (generally not a problem).

* *It is not uncommon for a user-written macro to treat a field differently depending on its type, such as a CLEAR macro which will move zero to a field if it is type P, blanks to a field if it is type C, or binary zeroes (X'00') to a field if it is type X.*

For example, given:

```
A    DC    CL4'1234'
B    DC    PL2'0'
C    DC    PL4'0'
```

...the instructions:

```
PACK  B,A
PACK  C,A
```

...will yield:

```
B = X'234F'
C = X'0001234F'
```

You Try It...

Given `N DC CL5'10639'`.

7. Given `W DC PL3'0'`, show the hex representation of `W` before and after `PACK W,N`
8. Given `X DC PL2'0'`, show the hex representation of `X` before and after `PACK X,N`
9. Given `Y DC CL4'0'`, show the hex representation of `Y` before and after `PACK Y,N`
10. Given `Z DC PL3'-12'`, show the hex representation of `Z` before and after `PACK N,Z` *(Be careful!)*

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

If you know IBM System/370 COBOL, you may recall that a packed number is one with a COMP-3 picture clause. For example, given the following field definitions,

```
05  FLDA            PIC S9(3).
05  FLDB            PIC S9(3)   COMP-3.
```

FLDB is a packed number, but FLDA is not. The COBOL instruction MOVE FLDA TO FLDB is equivalent to the BAL instruction PACK FLDB,FLDA ...and that's exactly what a COBOL compiler does: convert COBOL source instructions to the equivalent machine level instructions! (Those machine level instructions will, of course, vary depending on the machine type; that is, a PACK will be generated for an IBM/370 or compatible, but not for a PC.) This is one reason why COBOL programmers should learn assembler: it demystifies the COBOL complier. By knowing assembler, you become a better COBOL programmer (or any other language).

The AP Instruction

The AP (add packed) instruction is used to add one packed number to another. For example, AP A,B adds B to A, with B unchanged and the sum in A. Both operands *must* be valid packed numbers. For example, given

```
A    DC    CL4'1234'
B    DC    CL3'567'
C    DC    PL4'0'
```

...to add A and B giving C, A and B must be packed. At least one work field will be required:

```
PACK  C,A
PACK  PK2,B
AP    C,PK2
```

where

```
PK2    DC    PL2'0'
```

(I will often define fields, such as PK2 above, as work fields. For example, PK3 is DC PL3'0', PK4 is DC PL4'0', etc. Henceforth, if I refer to a field PKn, then it is to be understood that I am referring to a type P work field of size n. Use of this convention will simplify many of our illustrations.)

The ZAP Instruction

In coding the above example (add A and B giving C) I could have used use two work fields, as follows:

```
PACK  PK3,A
PACK  PK2,B
PACK  C,=C'0'
AP    C,PK3
AP    C,PK2
```

Note that moving a zero to C, then adding PK3 to C, is the same as if we had simply moved PK3 to C. There is a BAL instruction which will do just that: the ZAP (zero and add packed) instruction is used to move one packed field to another. For example, I could replace PACK C,=C'0' and AP C,PK3 with ZAP C,PK3. I could also replace PACK C,=C'0' with ZAP C,=P'0'.

You Try It...

11. Given X DC CL2' ', show the hex representation of X after PACK X,=CL1'1'.
12. Replace the PACK in (11) with a ZAP.
13. Given W DC PL2'10', X DC PL2'-6', Y DC CL2'15', and Z DC PL2'8'. Write the BAL code to determine Z = W + X + Y. Show the hex representation of Z after each instruction. Your final answer for Z should be X'019C'. (Reminder: Y must be packed before it can be added. Define a work field if necessary.)

Detecting an Overflow Condition

When adding (with AP) or moving (with ZAP), if the sum will not fit in the receiving field, the high order digits are truncated. The assembler will give no warning message that this might occur (as would most COBOL compilers), nor will the program abend at run time. Detecting such an "overflow" at run time is, however, a very simple process. Recall from our discussion of IFs in BAL that the condition code has four bits. The fourth bit is generally used to indicate an overflow condition:

EQUAL	LOW	HIGH	OVERFLOW
8	4	2	1
0	0	0	1

You can check for an overflow condition by using a branch on condition (`BC`) with a mask of `1`,or the extended mnemonic `BO` (branch on overflow). Conversely, you can use the mnemonic `BNO` (branch on no overflow). This can be demonstrated with the following program:

```
OVERFLOW BEGIN
         AP    A,B
         BO    OVER
         WTO   'There was NOT an overflow'
         B     DONE
OVER     EQU   *
         WTO   'There WAS an overflow'
DONE     EQU   *
         RETURN
A        DC    PL2'998'
B        DC    PL2'2'
         END   OVERFLOW
```

In this case, you will see the message `There WAS an overflow`, since `998+2=1000`, and `1000` will *not* fit in a two byte field. If you change the value of `B` from `2` to `1`, you will see the message `There was NOT an overflow`, since `998+1=999`, and `999` *will* fit in a two byte field.

The UNPK Instruction

After the above addition (adding `1234` to `567`), `C` will contain `+1801`, represented as `X'0001801C'`. Note that the result of any arithmetic operation (including `ZAP`) is signed (with `C` for positive, and `D` for negative), *even if the original numbers were unsigned!* (`PACK` and `UNPK` are *not* considered arithmetic operations.)

Of course, I cannot print field `C` in this format. The number must first be unpacked. The instruction for doing so is `UNPK`. This instruction works like the `PACK` instruction but in reverse. The sign from the packed number (which is the rightmost four bits) will go into the zone portion of the rightmost byte of the unpacked number. That field will also contain all of those redundant `F`'s which we discussed earlier. So, for example, `UNPK  WK5,C` where `WK5 DC CL5' '` will give `WK5 = X'F0F1F8F0C1'`.

(Just as in our discussion of work packed fields of the form `PKn` above, I will often define fields, such as `WK5` above, as work fields. For example: `WK3 DC CL3' '`, `WK4 DC CL4' '`, etc. Henceforth, if I refer to a field `WKn`, then it is to be understood that I am referring to a type `C` work field of size `n`.)

If the receiving field is too small to hold the unpacked number, the high order digits will be truncated. For example, `UNPK WK3,C` will give `WK3 = X'F8F0C1'` (where `WK3` was defined as `WK3 DC CL3' '`.) If the receiving field is larger than is necessary to hold the unpacked number, it will be padded on the left with `X'F0'`s. So `UNPK WK7,C` will give `WK7 = X'F0F0F0F1F8F0C1'` (where `WK7` was defined as `WK7 DC CL7' '`.)

You Try It...

Given `Q DC PL2'-708'`
14. Show `WK3` after `UNPK WK3,Q`
15. Show `WK2` after `UNPK WK2,Q`

Given `R DC PL1'9'`
16. Show `WK1` after `UNPK WK1,R`
17. Show `WK3` after `UNPK WK3,R`

The MVZ Instruction

In the above example, even after unpacking field `C`, I will have a problem in printing because of the sign: `C` would be displayed as `0180A` because `X'C1'` is the `EBCDIC` equivalent of `C'A'`. So I must somehow remove the sign. (In a later chapter, when we discuss the `ED` (edit) instruction, we will learn how to print the sign. For now, we will just remove it.)

To remove the sign, we make use of our knowledge of **zone** and **numeric** bits. First, let's rewrite this number in the vertical form we saw earlier:

F	F	F	F	C
0	1	8	0	1

And consider the following field: `ZEROES DC CL5'00000'`

F	F	F	F	F
0	0	0	0	0

If I move `ZEROES` to field `C` I would, of course, get all zeroes. But what if I moved *only the zone portion* of `ZEROES`? I would get the following:

F	F	F	F	F	<-- *changed*
0	1	8	0	1	<-- *unchanged*

Note that the sign would change from `C` (positive) to `F` (unsigned). This type of move is accomplished by using the `MVZ` (Move Zone) instruction. For example `MVZ C,ZEROES`

Of course, it is only the last (right most) byte that I need to change, so this could be coded as `MVZ C+4(1),=X'F0'`

If for some reason you change the length of field `C` (from `CL5`), you will need to remember to change the explicit displacement (`4`) in this instruction. There's a *very clever* way to code this instruction so you don't have to be concerned with the length of field `C`. Let the assembler determine that for you at assembly time with the **length operator**: `MVZ    C+L'C-1(1),=X'F0'` The assembler will use the definition of `C` to determine that `L'C = 5`, and therefore `L'C-1 = 5-1 = 4`.

The final result, `X'F0F1F8F0F1'`, will print as `01801`. So what about the leading zero: leading zeroes are usually suppressed? The edit instruction which we will learn in the next chapter will remove the leading zeroes as well as enable us to print the proper sign.

A similar instruction, `MVN`, will move the numeric bits only. For example, if I had used `MVN C,ZEROES` I would get `C` equal

F	F	F	F	C	*<-- unchanged*
0	0	0	0	0	*<-- changed*

You Try It...

Given `P DC PL3'314'`

18. Show the hex representation of `P`. (Recall that when a packed field is defined with a `DC`, the initial value is always signed.)
19. Write the instructions necessary to "move" `P` to `WK4` such that `WK4` is `X'F0F3F1F4'`.
20. Write the instructions necessary to "move" `P` to `WK2` such that `WK2` is `X'F1F4'`.

Programming Example: Producing the Sales Recap

We have discussed packed decimal format numbers and looked at the following instructions: `PACK`, `UNPK`, `AP`, `ZAP`, and `MVZ`. We are now ready to return to our programming problem: to produce the Sales Recap for Cogsworth Industries. The record layout for `COGS.DAT` was shown on the first page of this chapter. We will use the following record definition:

```
*****************************************************************
*        Input record definition                                *
*****************************************************************
IREC     DS    0CL41        1-41  Inventory record
IDESC    DS    CL10         1-10  Product description
ICALIF   DS    CL3         11-13  Units sold in Calif
IILL     DS    CL3         14-16  Units sold in Illinois
IUTAH    DS    CL3         17-19  Units sold in Utah
IWISC    DS    CL3         20-22  Units sold in Wisconsin
IBEGIN   DS    CL3         23-25  Beginning inventory
IPURCH   DS    CL3         26-28  Purchases throughout year
IQOH     DS    CL3         29-31  Actual quantity on hand
ICOST    DS    CL4         32-35  Cost (each) 99V99
ISELL    DS    CL4         36-39  Sell for (each) 99V99
ICRLF    DS    CL2         40-41  PC/370 only - CR/LF
```

The print layout for the Sales Recap is as follows:

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
                 COGSWORTH INDUSTRIES
                      Sales Recap

Product       Calif     Ill      Utah     Wisc    TOTAL
----------    -----    -----    -----    -----    -----
XXXXXXXXXX     XXX      XXX      XXX      XXX      XXX
XXXXXXXXXX     XXX      XXX      XXX      XXX      XXX
XXXXXXXXXX     XXX      XXX      XXX      XXX      XXX

XXX records processed.
```

...so we will use the following output record definition:

```
****************************************************************
*        Output (line) definition                              *
****************************************************************
OREC     DS    0CL62        1-62
ODESC    DS    CL10         1-10  Product description
         DS    CL5         11-15
OCALIF   DS    CL3         16-18  Units sold in Calif
         DS    CL6         19-24
OILL     DS    CL3         25-27  Units sold in Illinois
         DS    CL6         28-33
OUTAH    DS    CL3         34-36  Units sold in Utah
         DS    CL6         37-42
OWISC    DS    CL3         43-45  Units sold in Wisconsin
         DS    CL6         46-51
OTOTAL   DS    CL3         52-54  Units sold in all states
         DS    CL6         55-60
OCRLF    DS    CL2         61-62  PC/370 only - CR/LF
```

In this example, the input and output fields for sales by state are the same size, so there is no need to `PACK` them in order to print them; the `MVC` instruction will work fine:

```
         MVC   OCALIF,ICALIF
         MVC   OILL,IILL
         MVC   OUTAH,IUTAH
         MVC   OWISC,IWISC
```

But to get `OTOTAL`, the total units sold in all states, we *will* need to `PACK` each of the state sales figures, then add them to a total (also packed), then `UNPK` that total into `OTOTAL`, and remove the sign. We therefore define:

```
WCALIF   DC    PL2'0'             Units sold in Calif
WILL     DC    PL2'0'             Units sold in Illinois
WUTAH    DC    PL2'0'             Units sold in Utah
WWISC    DC    PL2'0'             Units sold in Wisconsin
WTOTAL   DC    PL2'0'             Units sold in all states
```

The code to find total sales is:

```
         PACK  WCALIF,ICALIF      Each product's sales must
         PACK  WILL,IILL           be packed so they can be
         PACK  WUTAH,IUTAH          added to total for this
         PACK  WWISC,IWISC           product...
         ZAP   WTOTAL,=P'0'       Initialize the total to zero
         AP    WTOTAL,WCALIF       and start adding...
         AP    WTOTAL,WILL
         AP    WTOTAL,WUTAH
         AP    WTOTAL,WWISC
         UNPK  OTOTAL,WTOTAL      Move total to output,
         MVZ   OTOTAL+2(1),=X'F0'  and remove the sign.
```

In this particular program it was not necessary to PACK each states' sales figures into a separate field. I could have used a single work field (such as PK2) as follows:

```
         ZAP   WTOTAL,=P'0'       Initialize total to zero
         PACK  PK2,ICALIF         Pack...
         AP    WTOTAL,PK2         ...then add Calif to total
         PACK  PK2,IILL           Pack...
         AP    WTOTAL,PK2         ...then add Illinois to total
         PACK  PK2,IUTAH          Pack...
         AP    WTOTAL,PK2         ...then add Utah to total
         PACK  PK2,IWISC          Pack...
         AP    WTOTAL,PK2         ...then add Wisc to total
         UNPK  OTOTAL,WTOTAL      Move total to output,
         MVZ   OTOTAL+2(1),=X'F0'  and remove the sign.
```

The decision to use a dedicated packed field for each input field (as in the former example) or to use a single packed field shared by all input fields (as in the latter example) will depend upon what you will do with the data: if you're going to need the number several times, the use of a dedicated packed field for each input field will save you from having to PACK each field more than once.

In order to determine the number of records processed, we will modify the READ routine as follows:

```
****************************************************************
*        READ - Read a record.                                 *
****************************************************************
READ     EQU   *
         ST    R10,SVREAD
         GET   INVENTRY,IREC      Read a single product record
         AP    #IN,=P'1'          Increment record count
         B     READX
ATEND    EQU   *
         MVI   EOFSW,C'Y'
READX    EQU   *
         L     R10,SVREAD
         BR    R10
```

where

```
#IN      DC    PL2'0'             Input record count
```

As this count is shown one time only, after all records have been processed, the logic to do so will appear in the `WRAPUP` section. We will reuse the output record area as follows:

```
****************************************************************
*        WRAPUP - Those things which happen one time only,     *
*                 after all records have been processed.       *
****************************************************************
WRAPUP   EQU   *
         ST    R10,SVWRAP
         MVC   OREC,BLANKS
         MVC   OCRLF,WCRLF        PC/370 only.
         BAL   R10,WRITE          Skip a line.
         MVC   OREC(22),=CL22'XXX records processed.'
         UNPK  OREC(3),#IN        Count
         MVZ   OREC+2(1),=X'F0'   Remove sign
         BAL   R10,WRITE
         CLOSE INVENTRY
         CLOSE REPORT
         WTO   'COGS7A ... Sales recap on REPORT.TXT'
         L     R10,SVWRAP
         BR    R10
```

The complete program, `COGS7A.MLC`, follows.

```
         PRINT NOGEN
****************************************************************
*        FILENAME:  COGS7A.MLC                                 *
*        AUTHOR  :  Bill Qualls                                *
*        SYSTEM  :  PC/370 R4.2                                *
*        REMARKS :  Produce report for COGSWORTH INDUSTRIES    *
*                   showing sales by state.                    *
****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'COGS7A ... Begin execution'
         BAL   R10,SETUP
MAIN     EQU   *
         CLI   EOFSW,C'Y'
         BE    EOJ
         BAL   R10,PROCESS
         B     MAIN
EOJ      EQU   *
         BAL   R10,WRAPUP
         WTO   'COGS7A ... Normal end of program'
         RETURN
****************************************************************
*        SETUP - Those things which happen one time only,      *
*                before any records are processed.             *
****************************************************************
SETUP    EQU   *
         ST    R10,SVSETUP
         OI    INVENTRY+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  INVENTRY
         OPEN  REPORT
```

(continued)

```
         BAL   R10,HDGS
         BAL   R10,READ
         L     R10,SVSETUP
         BR    R10
****************************************************************
*        HDGS - Print headings.                                *
****************************************************************
HDGS     EQU   *
         ST    R10,SVHDGS
         PUT   REPORT,HD1
         PUT   REPORT,HD2
         PUT   REPORT,HD3
         PUT   REPORT,HD4
         PUT   REPORT,HD5
         L     R10,SVHDGS
         BR    R10
****************************************************************
*        PROCESS - Those things which happen once per record.  *
****************************************************************
PROCESS  EQU   *
         ST    R10,SVPROC
         BAL   R10,FORMAT
         BAL   R10,WRITE
         BAL   R10,READ
         L     R10,SVPROC
         BR    R10
****************************************************************
*        READ - Read a record.                                 *
****************************************************************
READ     EQU   *
         ST    R10,SVREAD
         GET   INVENTRY,IREC       Read a single product record
         AP    #IN,=P'1'           Increment record count
         B     READX
ATEND    EQU   *
         MVI   EOFSW,C'Y'
READX    EQU   *
         L     R10,SVREAD
         BR    R10
****************************************************************
*        FORMAT - Format a single detail line.                 *
****************************************************************
FORMAT   EQU   *
         ST    R10,SVFORM
         MVC   OREC,BLANKS
         MVC   ODESC,IDESC
         MVC   OCALIF,ICALIF
         MVC   OILL,IILL
         MVC   OUTAH,IUTAH
         MVC   OWISC,IWISC
         PACK  WCALIF,ICALIF       Each product's sales must
         PACK  WILL,IILL            be packed so they can be
         PACK  WUTAH,IUTAH            added to total for this
         PACK  WWISC,IWISC             product...
         ZAP   WTOTAL,=P'0'        Initialize the total to zero
         AP    WTOTAL,WCALIF        and start adding...
         AP    WTOTAL,WILL
         AP    WTOTAL,WUTAH
         AP    WTOTAL,WWISC
```

(continued)

```
         UNPK  OTOTAL,WTOTAL      Move total to output,
         MVZ   OTOTAL+2(1),=X'F0'  and remove the sign.
         MVC   OCRLF,WCRLF        PC/370 only.
         L     R10,SVFORM
         BR    R10
****************************************************************
*        WRITE - Write a single detail line.                   *
****************************************************************
WRITE    EQU   *
         ST    R10,SVWRITE
         PUT   REPORT,OREC        Write report line
         L     R10,SVWRITE
         BR    R10
****************************************************************
*        WRAPUP - Those things which happen one time only,     *
*                 after all records have been processed.       *
****************************************************************
WRAPUP   EQU   *
         ST    R10,SVWRAP
         MVC   OREC,BLANKS
         MVC   OCRLF,WCRLF        PC/370 only.
         BAL   R10,WRITE          Skip a line.
         MVC   OREC(22),=CL22'XXX records processed.'
         UNPK  OREC(3),#IN        Count
         MVZ   OREC+2(1),=X'F0'   Remove sign
         BAL   R10,WRITE
         CLOSE INVENTRY
         CLOSE REPORT
         WTO   'COGS7A ... Sales recap on REPORT.TXT'
         L     R10,SVWRAP
         BR    R10
****************************************************************
*        Literals, if any, will go here                        *
****************************************************************
         LTORG
****************************************************************
*        File definitions                                      *
****************************************************************
INVENTRY DCB   LRECL=41,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='COGS.DAT'
REPORT   DCB   LRECL=62,RECFM=F,MACRF=P,
               DDNAME='REPORT.TXT'
****************************************************************
*        RETURN ADDRESSES                                      *
****************************************************************
SVSETUP  DC    F'0'               SETUP
SVHDGS   DC    F'0'               HDGS
SVPROC   DC    F'0'               PROCESS
SVREAD   DC    F'0'               READ
SVFORM   DC    F'0'               FORMAT
SVWRITE  DC    F'0'               WRITE
SVWRAP   DC    F'0'               WRAPUP
****************************************************************
*        Miscellaneous field definitions                       *
****************************************************************
WCRLF    DC    X'0D25'            PC/370 ONLY - EBCDIC CR/LF
EOFSW    DC    CL1'N'             End of file? (Y/N)
BLANKS   DC    CL62' '
```

(continued)

```
WCALIF   DC    PL2'0'             Units sold in Calif
WILL     DC    PL2'0'             Units sold in Illinois
WUTAH    DC    PL2'0'             Units sold in Utah
WWISC    DC    PL2'0'             Units sold in Wisconsin
WTOTAL   DC    PL2'0'             Units sold in all states
#IN      DC    PL2'0'             Input record count
****************************************************************
*        Input record definition                               *
****************************************************************
IREC     DS    0CL41        1-41  Inventory record
IDESC    DS    CL10         1-10  Product description
ICALIF   DS    CL3         11-13  Units sold in Calif
IILL     DS    CL3         14-16  Units sold in Illinois
IUTAH    DS    CL3         17-19  Units sold in Utah
IWISC    DS    CL3         20-22  Units sold in Wisconsin
IBEGIN   DS    CL3         23-25  Beginning inventory
IPURCH   DS    CL3         26-28  Purchases throughout year
IQOH     DS    CL3         29-31  Actual quantity on hand
ICOST    DS    CL4         32-35  Cost (each) 99V99
ISELL    DS    CL4         36-39  Sell for (each) 99V99
ICRLF    DS    CL2         40-41  PC/370 only - CR/LF
****************************************************************
*        Output (line) definition                              *
****************************************************************
OREC     DS    0CL62        1-62
ODESC    DS    CL10         1-10  Product description
         DS    CL5         11-15
OCALIF   DS    CL3         16-18  Units sold in Calif
         DS    CL6         19-24
OILL     DS    CL3         25-27  Units sold in Illinois
         DS    CL6         28-33
OUTAH    DS    CL3         34-36  Units sold in Utah
         DS    CL6         37-42
OWISC    DS    CL3         43-45  Units sold in Wisconsin
         DS    CL6         46-51
OTOTAL   DS    CL3         52-54  Units sold in all states
         DS    CL6         55-60
OCRLF    DS    CL2         61-62  PC/370 only - CR/LF
****************************************************************
*        Headings definitions                                  *
****************************************************************
HD1      DS    0CL62
         DC    CL40'                 COGSWORTH INDUSTRIES   '
         DC    CL20' '
         DC    XL2'0D25'
HD2      DS    0CL62
         DC    CL40'                      Sales Recap       '
         DC    CL20' '
         DC    XL2'0D25'
HD3      DS    0CL62
         DC    CL60' '
         DC    XL2'0D25'
HD4      DS    0CL62
         DC    CL40'Product       Calif     Ill      Utah   '
         DC    CL20'  Wisc    TOTAL'
         DC    XL2'0D25'
HD5      DS    0CL62
         DC    CL40'----------    -----    -----    -----   '
         DC    CL20' -----    -----'
         DC    XL2'0D25'
         END   BEGIN
```

Programming Example: Producing the Inventory Discrepancies Report

Our next report, Inventory Discrepancies, will appear as follows:

```
         1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
                    COGSWORTH INDUSTRIES
               Inventory Discrepancies Report

Product      Begin + Purch - Sales = Expect   Actual     Result
----------   -----   -----   -----   ------   ------   ----------
GIZMOS        017     099     090      026      023      003 short
WIDGETS       022     034     037      019      019
JUNQUE        030     052     073      009      010      001 over

003 records processed.
001 indicate shortage.
001 indicate overage.
```

Notice that the difference (`Result`) is `Expect-Actual`, and this difference is printed *only* if other than zero. Therefore, in order to complete this program, we will need two more instructions: `SP` (subtract packed) and `CP` (compare packed).

The SP Instruction

The `SP` (subtract packed) instruction is used to subtract one packed number from another. For example, `SP A,B` subtracts `B` from `A`, with `B` unchanged and the difference in `A`.

Consider the following example. Given these field definitions:

```
ACTUAL   DS    CL6
BUDGET   DS    CL6
DIFFER   DS    CL6
```

...subtract `ACTUAL` from `BUDGET` giving `DIFFER`:

```
PACK  PK4,ACTUAL
PACK  PK5,BUDGET
SP    PK4,PK5
UNPK  DIFFER,PK4
MVZ   DIFFER+L'DIFFER-1(1),=X'F0'
```

(Note the use of `PKn`-type work fields, the definition of which should be obvious per our earlier discussion. `BUDGET` *could* fit in a four byte field, but `PK4` was already holding `ACTUAL`, so I used `PK5` instead.)

There should be some concern over the fact that we simply *removed* the sign in `DIFFER`. Certainly, being *over* budget is not the same as being *under* budget! We'll take care of this problem in the next chapter.

You Try It...

Given `A DC PL2'10'`, `B DC PL3'5'`, `C DC PL1'-3'`, and `D DC PL2'7'`. Show the hex representation of `A` after each of the following. (Start with fresh data each time.)

21. `SP A,B`
22. `SP A,C`
23. `SP D,A`

The CP Instruction

The `CP` (compare packed) instruction works just like the `CLC` (compare logical character) instruction, except:

- both operands *must* be valid packed fields, and
- the maximum length for each operand is 16.

The following program segment will illustrate the use of `SP` and `CP`. Given:

```
A        DS    PL3
B        DS    PL3
C        DS    PL4
```

...we are to subtract the lesser of `A` and `B` from `C`. The necessary `BAL` code is as follows:

```
         CP    A,B
         BH    USEB
         SP    C,A
         B     DONE
USEB     EQU   *
         SP    C,B
DONE     EQU   *
```

You Try It...

Given `A`, `B`, `C`, and `D` are all valid packed fields.

24. Write the code necessary to subtract one from `A` if `A` is greater than `B`.
25. Write the code necessary to add `A` to `B` if `B` is equal to `C`.
26. Write the code necessary to move the maximum of `A`, `B`, and `C` to `D`.

Programming Example Revisited

We are now ready to return to our programming problem: to produce the Inventory Discrepancies Report for Cogsworth Industries. We will use the same input record definition as was used in `COGS7A.MLC`. The print layout for the report is as follows:

```
         1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
                    COGSWORTH INDUSTRIES
               Inventory Discrepancies Report

Product      Begin + Purch - Sales = Expect   Actual     Result
----------   -----   -----   -----   ------   ------   ----------
XXXXXXXXXX    XXX     XXX     XXX      XXX     XXX      XXX XXXXX
XXXXXXXXXX    XXX     XXX     XXX      XXX     XXX      XXX XXXXX
XXXXXXXXXX    XXX     XXX     XXX      XXX     XXX      XXX XXXXX

XXX records processed.
XXX indicate shortage.
XXX indicate overage.
```

...so we will use the following output record definition:

```
****************************************************************
*        Output (line) definition                              *
****************************************************************
OREC     DS    0CL67         1-67
ODESC    DS    CL10          1-10  Product description
         DS    CL4          11-14
OBEGIN   DS    CL3          15-17  Beginning inventory
         DS    CL5          18-22
OPURCH   DS    CL3          23-25  Purchases
         DS    CL5          26-30
OSALES   DS    CL3          31-33  Units sold
         DS    CL6          34-39
OENDING  DS    CL3          40-42  Ending inventory (expected)
         DS    CL5          43-47
OQOH     DS    CL3          48-50  Ending inventory (actual)
         DS    CL6          51-56
ODIFF    DS    CL3          57-59  Difference
         DS    CL1          60-60
ORESULT  DS    CL5          61-65  'over' or 'short'
OCRLF    DS    CL2          66-67  PC/370 only - CR/LF
```

Our logic for formatting the detail lines is as follows:

Step1: Add California, Illinois, Utah, and Wisconsin sales to get total sales. Since the input fields are not packed, and since addition requires both fields be packed, we will need to use some work fields.

```
         PACK  WCALIF,ICALIF      Each product's sales must
         PACK  WILL,IILL           be packed so they can be
         PACK  WUTAH,IUTAH          add to total for this
         PACK  WWISC,IWISC           product...
         ZAP   WTOTAL,=P'0'       Initialize the total to zero
         AP    WTOTAL,WCALIF       and start adding...
         AP    WTOTAL,WILL
         AP    WTOTAL,WUTAH
         AP    WTOTAL,WWISC
         UNPK  OSALES,WTOTAL      Move total to output,
         MVZ   OSALES+2(1),=X'F0'  and remove the sign.
```

Step 2: The expected ending inventory is equal to the beginning inventory *plus* purchases made since then *minus* total sales. This, too, will require the use of some work fields.

```
         PACK  WBEGIN,IBEGIN
         PACK  WPURCH,IPURCH      Expected ending inventory =
         ZAP   WENDING,WBEGIN      Beginning
         AP    WENDING,WPURCH       + Purchases
         SP    WENDING,WTOTAL        - Sales
         UNPK  OENDING,WENDING
         MVZ   OENDING+2(1),=X'F0'
```

Step 3: Subtract the actual quantity on hand from the expected ending inventory. Display this difference if not equal to zero. Increment counters for number of records indicating overage or shortage.

```
         PACK  WQOH,IQOH
         CP    WQOH,WENDING       Compare actual vs. expected
         BE    FORMATX            Don't show difference if zero
         BL    SHORT
         AP    #OVER,=P'1'        Count overages
         MVC   ORESULT,=CL5'over'
         B     DODIFF
SHORT    EQU   *
         AP    #SHORT,=P'1'       Count shortages
         MVC   ORESULT,=CL5'short'
DODIFF   EQU   *
         ZAP   WDIFF,WENDING      Difference = Expected - Actual
         SP    WDIFF,WQOH
         UNPK  ODIFF,WDIFF
         MVZ   ODIFF+2(1),=X'F0'
FORMATX  EQU   *
```

Record counts are shown in the `WRAPUP` section. That code is included in the following (complete) program listing:

```
         PRINT NOGEN
****************************************************************
*        FILENAME:  COGS7B.MLC                                 *
*        AUTHOR  :  Bill Qualls                                *
*        SYSTEM  :  PC/370 R4.2                                *
*        REMARKS :  Produce report for COGSWORTH INDUSTRIES    *
*                   showing inventory discrepancies.           *
****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'COGS7B ... Begin execution'
         BAL   R10,SETUP
MAIN     EQU   *
         CLI   EOFSW,C'Y'
         BE    EOJ
         BAL   R10,PROCESS
         B     MAIN
```

(coninued)

```
EOJ      EQU   *
         BAL   R10,WRAPUP
         WTO   'COGS7B ... Normal end of program'
         RETURN
****************************************************************
*        SETUP - Those things which happen one time only,      *
*                before any records are processed.             *
****************************************************************
SETUP    EQU   *
         ST    R10,SVSETUP
         OI    INVENTRY+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  INVENTRY
         OPEN  REPORT
         BAL   R10,HDGS
         BAL   R10,READ
         L     R10,SVSETUP
         BR    R10
****************************************************************
*        HDGS - Print headings.                                *
****************************************************************
HDGS     EQU   *
         ST    R10,SVHDGS
         PUT   REPORT,HD1
         PUT   REPORT,HD2
         PUT   REPORT,HD3
         PUT   REPORT,HD4
         PUT   REPORT,HD5
         L     R10,SVHDGS
         BR    R10
****************************************************************
*        PROCESS - Those things which happen once per record.  *
****************************************************************
PROCESS  EQU   *
         ST    R10,SVPROC
         BAL   R10,FORMAT
         BAL   R10,WRITE
         BAL   R10,READ
         L     R10,SVPROC
         BR    R10
****************************************************************
*        READ - Read a record.                                 *
****************************************************************
READ     EQU   *
         ST    R10,SVREAD
         GET   INVENTRY,IREC      Read a single product record
         AP    #IN,=P'1'          Increment record count
         B     READX
ATEND    EQU   *
         MVI   EOFSW,C'Y'
READX    EQU   *
         L     R10,SVREAD
         BR    R10
****************************************************************
*        FORMAT - Format a single detail line.                 *
****************************************************************
FORMAT   EQU   *
         ST    R10,SVFORM
```

(continued)

```
         MVC   OREC,BLANKS
         MVC   ODESC,IDESC        Description
         MVC   OBEGIN,IBEGIN      Beginning inventory
         MVC   OPURCH,IPURCH      Purchases
         PACK  WCALIF,ICALIF      Each product's sales must
         PACK  WILL,IILL           be packed so they can be
         PACK  WUTAH,IUTAH          added to total for this
         PACK  WWISC,IWISC           product...
         ZAP   WTOTAL,=P'0'       Initialize the total to zero
         AP    WTOTAL,WCALIF       and start adding...
         AP    WTOTAL,WILL
         AP    WTOTAL,WUTAH
         AP    WTOTAL,WWISC
         UNPK  OSALES,WTOTAL      Move total to output,
         MVZ   OSALES+2(1),=X'F0'  and remove the sign.
         PACK  WBEGIN,IBEGIN
         PACK  WPURCH,IPURCH      Expected ending inventory =
         ZAP   WENDING,WBEGIN      Beginning
         AP    WENDING,WPURCH       + Purchases
         SP    WENDING,WTOTAL        - Sales
         UNPK  OENDING,WENDING
         MVZ   OENDING+2(1),=X'F0'
         MVC   OQOH,IQOH          Actual ending inventory
         MVC   OCRLF,WCRLF        PC/370 only.
         PACK  WQOH,IQOH
         CP    WQOH,WENDING       Compare actual vs. expected
         BE    FORMATX            Don't show difference if zero
         BL    SHORT
         AP    #OVER,=P'1'        Count overages
         MVC   ORESULT,=CL5'over'
         B     DODIFF
SHORT    EQU   *
         AP    #SHORT,=P'1'       Count shortages
         MVC   ORESULT,=CL5'short'
DODIFF   EQU   *
         ZAP   WDIFF,WENDING      Difference = Expected - Actual
         SP    WDIFF,WQOH
         UNPK  ODIFF,WDIFF
         MVZ   ODIFF+2(1),=X'F0'
FORMATX  EQU   *
         L     R10,SVFORM
         BR    R10
****************************************************************
*        WRITE - Write a single detail line.                   *
****************************************************************
WRITE    EQU   *
         ST    R10,SVWRITE
         PUT   REPORT,OREC        Write report line
         L     R10,SVWRITE
         BR    R10
****************************************************************
*        WRAPUP - Those things which happen one time only,     *
*                 after all records have been processed.       *
****************************************************************
WRAPUP   EQU   *
         ST    R10,SVWRAP
         MVC   OREC,BLANKS
         MVC   OCRLF,WCRLF        PC/370 only.
```

(continued)

```
         BAL   R10,WRITE          Skip a line.
         MVC   OREC(22),=CL22'XXX records processed.'
         UNPK  OREC(3),#IN        Count
         MVZ   OREC+2(1),=X'F0'   Remove sign
         BAL   R10,WRITE
         MVC   OREC(22),=CL22'XXX indicate shortage.'
         UNPK  OREC(3),#SHORT     Count
         MVZ   OREC+2(1),=X'F0'   Remove sign
         BAL   R10,WRITE
         MVC   OREC(22),=CL22'XXX indicate overage. '
         UNPK  OREC(3),#OVER      Count
         MVZ   OREC+2(1),=X'F0'   Remove sign
         BAL   R10,WRITE
         CLOSE INVENTRY
         CLOSE REPORT
         WTO   'COGS7B ... Discrepancies report on REPORT.TXT'
         L     R10,SVWRAP
         BR    R10
*****************************************************************
*        Literals, if any, will go here                         *
*****************************************************************
         LTORG
*****************************************************************
*        File definitions                                       *
*****************************************************************
INVENTRY DCB   LRECL=41,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='COGS.DAT'
REPORT   DCB   LRECL=67,RECFM=F,MACRF=P,
               DDNAME='REPORT.TXT'
*****************************************************************
*        RETURN ADDRESSES                                       *
*****************************************************************
SVSETUP  DC    F'0'               SETUP
SVHDGS   DC    F'0'               HDGS
SVPROC   DC    F'0'               PROCESS
SVREAD   DC    F'0'               READ
SVFORM   DC    F'0'               FORMAT
SVWRITE  DC    F'0'               WRITE
SVWRAP   DC    F'0'               WRAPUP
*****************************************************************
*        Miscellaneous field definitions                        *
*****************************************************************
WCRLF    DC    X'0D25'            PC/370 ONLY - EBCDIC CR/LF
EOFSW    DC    CL1'N'             End of file? (Y/N)
BLANKS   DC    CL67' '
WCALIF   DC    PL2'0'             Units sold in Calif
WILL     DC    PL2'0'             Units sold in Illinois
WUTAH    DC    PL2'0'             Units sold in Utah
WWISC    DC    PL2'0'             Units sold in Wisconsin
WTOTAL   DC    PL2'0'             Units sold in all states
WBEGIN   DC    PL2'0'             Beginning inventory
WPURCH   DC    PL2'0'             Purchases
WENDING  DC    PL2'0'             Ending inventory (expected)
WQOH     DC    PL2'0'             Ending inventory (actual)
WDIFF    DC    PL2'0'             Difference
#IN      DC    PL2'0'             Input record count
#OVER    DC    PL2'0'             Records showing overage
#SHORT   DC    PL2'0'             Records showing shortage
```

(continued)

```
****************************************************************
*        Input record definition                               *
****************************************************************
IREC     DS    0CL41        1-41  Inventory record
IDESC    DS    CL10         1-10  Product description
ICALIF   DS    CL3         11-13  Units sold in Calif
IILL     DS    CL3         14-16  Units sold in Illinois
IUTAH    DS    CL3         17-19  Units sold in Utah
IWISC    DS    CL3         20-22  Units sold in Wisconsin
IBEGIN   DS    CL3         23-25  Beginning inventory
IPURCH   DS    CL3         26-28  Purchases throughout year
IQOH     DS    CL3         29-31  Actual quantity on hand
ICOST    DS    CL4         32-35  Cost (each) 99V99
ISELL    DS    CL4         36-39  Sell for (each) 99V99
ICRLF    DS    CL2         40-41  PC/370 only - CR/LF
****************************************************************
*        Output (line) definition                              *
****************************************************************
OREC     DS    0CL67        1-67
ODESC    DS    CL10         1-10  Product description
         DS    CL4         11-14
OBEGIN   DS    CL3         15-17  Beginning inventory
         DS    CL5         18-22
OPURCH   DS    CL3         23-25  Purchases
         DS    CL5         26-30
OSALES   DS    CL3         31-33  Units sold
         DS    CL6         34-39
OENDING  DS    CL3         40-42  Ending inventory (expected)
         DS    CL5         43-47
OQOH     DS    CL3         48-50  Ending inventory (actual)
         DS    CL6         51-56
ODIFF    DS    CL3         57-59  Difference
         DS    CL1         60-60
ORESULT  DS    CL5         61-65  'over' or 'short'
OCRLF    DS    CL2         66-67  PC/370 only - CR/LF
****************************************************************
*        Headings definitions                                  *
****************************************************************
HD1      DS    0CL67
         DC    CL40'                    COGSWORTH INDUSTRIES'
         DC    CL25' '
         DC    XL2'0D25'
HD2      DS    0CL67
         DC    CL40'               Inventory Discrepancies R'
         DC    CL25'eport'
         DC    XL2'0D25'
HD3      DS    0CL67
         DC    CL65' '
         DC    XL2'0D25'
HD4      DS    0CL67
         DC    CL40'Product      Begin + Purch - Sales = Exp'
         DC    CL25'ect   Actual     Result  '
         DC    XL2'0D25'
HD5      DS    0CL67
         DC    CL40'----------   -----   -----   -----   ---'
         DC    CL25'---   ------   ----------'
         DC    XL2'0D25'
         END   BEGIN
```

The SS Instruction Format Revisited...

The PACK, UNPK, AP, ZAP, SP, and CP instructions are all type SS, or Storage-to-Storage, instructions. When we first introduced the SS instruction format, we said there was more than one form of SS instruction. All of these instructions are of the second form:

```
         +--------------+-------------+-------------+
         |  1st Halfword | 2nd Halfword | 3rd Halfword |
  +------+-------+---+---+---+---------+---+---------+
  |  SS  |Op Code|L1 |L2 |B1 |    D1   |B2 |    D2   |
  +------+-------+---+---+---+---------+---+---------+
```

The different form is necessary because these operations have *two* length fields; that is, the operands can be of different lengths. (Recall that the MVC and CLC instructions have one length operator only and that the length of the operation is determined by the length of the first operand.)

As with all assembler instructions, the first of these six bytes is the operation code. These op codes are as follows:

Instruction	Op Code
PACK	X'F2'
UNPK	X'F3'
AP	X'FA'
ZAP	X'F8'
SP	X'FB'
CP	X'F9'

(MVZ and MVN are also type SS instructions, but of the same form as the MVC and CLC instructions. The op code for MVZ is X'D3'. The op code for MVN is X'D1'.)

You can see most of these op codes in the following extract of COGS7B.PRN:

```
0001C6 F212D472D498   0482   04A8  161           PACK     WQOH,IQOH
0001CC F911D472D470   0482   0480  162           CP       WQOH,WENDING
0001D2 4780D1FE              020E  163           BE       FORMATX
0001D6 4740D1DA              01EA  164           BL       SHORT
0001DA FA10D478D2FC   0488   030C  165           AP       #OVER,=P'1'
0001E0 D204D4E1D2F5   04F1   0305  166           MVC      ORESULT,=CL5'over'
0001E6 47F0D1E6              01F6  167           B        DODIFF
0001EA               000001EA      168 SHORT     EQU      *
0001EA FA10D47AD2FC   048A   030C  169           AP       #SHORT,=P'1'
0001F0 D204D4E1D2F0   04F1   0300  170           MVC      ORESULT,=CL5'short'
0001F6               000001F6      171 DODIFF    EQU      *
0001F6 F811D474D470   0484   0480  172           ZAP      WDIFF,WENDING
0001FC FB11D474D472   0484   0482  173           SP       WDIFF,WQOH
000202 F321D4DDD474   04ED   0484  174           UNPK     ODIFF,WDIFF
000208 D300D4DFD2FA   04EF   030A  175           MVZ      ODIFF+2(1),=X'F0'
00020E               0000020E      176 FORMATX   EQU      *
```

The second byte of the instruction is the lengths. Each length occupies a halfbyte, or four bits. Recall that four bits can range in value from 0 (all zeroes) to 15 (all ones). But just as in our discussion of MVC and CLC, a field of length 0 doesn't make any sense. So the lengths are actually the length of each operand *minus one*. In this way, values from 0 to 15 indicate operation lengths (adds, subtracts, compares, etc.) of from 1 to 16. This is why these packed decimal operations are limited to fields of length 1 to 16.

For example, in line 161 we PACK IQOH into WQOH. WQOH was defined as PL2, so the first length will be 2-1=1. IQOH was defined as CL3, so the second length will be 3-1=2. That's why the Op Code of X'F2' is followed by X'12'.

The second halfword (third and fourth bytes) indicates the base and displacement of the first operand, and the third halfword (fifth and sixth bytes) indicates the base and displacement of the second operand, just as with MVC and CLC. Our earlier discussion of the ADR1 and ADR2 columns applies as well.

You Try It...

27. Show the object code if line 161 above were changed to UNPK IQOH,WQOH
28. Show the object code if line 169 above were changed to AP #SHORT,#OVER
29. Show the object code if line 172 above were changed to AP WDIFF,WENDING+1(1)
30. Show the object code if line 174 above were changed to UNPK ODIFF(2),WENDING
31. Show the object code if line 175 above were changed to MVN ODIFF+2(1),=X'F0'
32. Show the object code if line 175 above were changed to MVZ ODIFF+1(1),=X'F0'

Data Exception Errors

Now that you are working with packed decimal fields, you have a much greater chance of having runtime errors. These are sometimes referred to as **abends** for abnormal end (or abnormal termination.) By far the most common such error is the **data exception** error. This occurs whenever you attempt to perform a packed decimal arithmetic operation with operands which are not valid packed numbers. For example, if you forget to pack a number before adding it to an accumulator, your program will abend. In the IBM mainframe world this produces a **system 0C7** error code, or **S0C7** (read as "sock seven"). PC/370 will also produce an error, though it is not called a S0C7.

Recall that AP, ZAP, SP, and CP are considered arithmetic operations: PACK and UNPK are not. The first operand for a ZAP can be anything, but the second operand *must* be a valid packed decimal number. PACK and UNPK will never produce a S0C7, but forgetting to PACK or UNPK certainly may!

PC/370 has a test/debugger facility, but it is *very* primitive. Nevertheless, if you know what to look for it can be very helpful. Consider the following program, S0C7.MLC:

```
         START 0
BEGIN    BEGIN
* ---------------------------
*   S0C7.MLC BY BILL QUALLS
*   USING PC/370 V4.2
*   FORCE DECIMAL EXCEPTION
* ---------------------------
         AP    SUM,ONE
         AP    SUM,TWO
         RETURN
SUM      DC    PL2'0'
ONE      DC    PL1'1'
TWO      DS    PL1'2' (Error)
         END
```

This program contains a deliberate error: the definition for `TWO` should be a `DC` instead of a `DS`. When I attempt to add `TWO` to `SUM` (the second `AP`), I will get an abend.

If we assemble, link, and execute this program we get the following:

```
A:\MIN>s0c7
TRACE EP A=08E4  ID=BUG   370  A=000276 OP=58DD0004
 ***********************************************
 *    PC/370 System Release 4.2    01/07/88    *
 *    Copyright (C) 1988 Donald S. Higgins     *
 *                                             *
 * You are encouraged to copy and share this   *
 * package with other users on the condition   *
 * the package is not distributed in modified  *
 * form, and that no fee is charged.  If you   *
 * find PC/370 useful, send 45 dollars to the  *
 * address below to become registered user and *
 * support continued shareware development.    *
 * Registered users will receive notices of    *
 * future PC/370 releases.                     *
 *                                             *
 *      Don Higgins                            *
 *      6365 - 32 Avenue, North                *
 *      St. Petersburg, Florida 33710          *
 ***********************************************
TYPE H FOR HELP
+
```

If we type `H` (but don't press Enter), we get the following help screen:

```
A SELECT ADDRESS STOP OPTIONS
C CONTINUE TO NEXT TRACE ID
D DUMP 370/386 MEMORY AT SELECTED ADDRESS
E SET EBCDIC OR ASCII DUMP FORMAT
F SET FIND TRACE ID
I DISPLAY 370 INSTRUCTION COUNTER WORD
J JUMP TO NEW 370/386 INSTRUCTION ADDRESS
K SET KILL TRACE MODE OR RESTORE TRACES
L SET LIMIT FOR Q AND T
M MODIFY 370/386 MEMORY
N LIST TRACE TABLE
```

(continued)

```
P PRINTER ON/OFF
Q SET QUIET MODE
R LIST 370/386 REGISTERS
S SAVE/UNSAVE CURRENT TRACE ID FROM KILL
T SET TRACE MODE
W LIST 370/386 FREE STORAGE
X ASSIST LOG ON/OFF
Y MODIFY 370/386 REGISTER
Z SWITCH 370/386 MODE (AFFECTS D,E,J,O,M,R,W,Y)
<CR>=REPEAT DUMP, <BS>=DUMP BACK, <SP>=DUMP FORWARD
<ESC>=EXIT TO MS-DOS
```

(We press `Esc` to return to the `DOS` prompt.)

`ID=BUG` tells us that the program has ended, and `A=000276` tells us where. But that address (`000276`) means nothing to us without the assembly listing: we *must* have the `.PRN` listing in order to be able to do any debugging.

```
   LOC                ADR1   ADR2 LINE LABEL    OP       OPERANDS
000000                               1          START    0
000000                               2 *++++    BEGIN
000000                               3 BEGIN    CSECT
000000                               4          USING    *,15
000000 47F0F058              0058    5          B        KZHQX001
000004 0B                            6          DC       AL1(11)
000005 C2C5C7C9D5404040              7          DC       CL11'BEGIN    '
000010 0000000000000000              8 HZQKX001 DC       18F'0'
000058 90ECD00C              000C    9 KZHQX001 STM      14,12,12(13)
00005C 50D0F014              0014   10          ST       13,HZQKX001+4
000060 18ED                         11          LR       14,13
000062 41D0F010              0010   12          LA       13,HZQKX001
000066 50D0E008              0008   13          ST       13,8(0,14)
00006A                              14          DROP     15
00006A                              15          USING    HZQKX001,13
00006A                              16 * --------------------------
00006A                              17 *   S0C7.MLC BY BILL QUALLS
00006A                              18 *   USING PC/370 V4.2
00006A                              19 *   FORCE DECIMAL EXCEPTION
00006A                              20 * --------------------------
00006A FA10D070D072   0080   0082   21          AP       SUM,ONE
000070 FA10D070D073   0080   0083   22          AP       SUM,TWO
000076                              23 *+++++++ RETURN
000076 58DD0004              0004   24          L        13,4(13)
00007A 98ECD00C              000C   25          LM       14,12,12(13)
00007E 07FE                         26          BR       14
000080 000C                         27 SUM      DC       PL2'0'
000082 1C                           28 ONE      DC       PL1'1'
000083                              29 TWO      DS       PL1'2'  (Error)
000000                              30          END
```

The address refers to the location counter (`LOC`) on the far left. But notice that there isn't any `000276`! We subtract `X'200'` from the address (*always*). This gives an address of `000076`, which corresponds to line 23. But wait! *When PC/370 encounters a data exception error, it gives you the address of the next instruction that would have been executed had the program not abended!* So the actual instruction in error is at location `000070`, line 22, the `AP` which we said would cause the abend! That instruction would cause us to look at the values of `SUM` and `TWO`. `SUM` is okay (or it would have abended with the previous instruction). So we look at `TWO` and see that it should have been `DC` rather than `DS`.

It's not easy. And it's even worse when you are dealing with file input. What's needed is some way to look at any field while the program is running. To do so, you must run the program in test mode. To run a program in test mode, the program name is entered, followed by a blank and an *upper case* `T`. For example,

```
A:\>s0c7 T
TRACE EP A=07AB ID=370   370 A=000200 OP=47F0F058
 ********************************************
 *    PC/370 System Release 4.2    01/07/88    *
 *    Copyright (C) 1988 Donald S. Higgins     *
 *                                             *
 * You are encouraged to copy and share this   *
 * package with other users on the condition   *
 * the package is not distributed in modified  *
 * form, and that no fee is charged.  If you   *
 * find PC/370 useful, send 45 dollars to the  *
 * address below to become registered user and *
 * support continued shareware development.    *
 * Registered users will receive notices of    *
 * future PC/370 releases.                     *
 *                                             *
 *      Don Higgins                            *
 *      6365 - 32 Avenue, North                *
 *      St. Petersburg, Florida 33710          *
 ********************************************
TYPE H FOR HELP
+
```

We want to stop the program before the offending instruction is executed, so at the prompt (`+`) we type **a** for address. The instruction is at location `000070` so we add `X'200'` giving `000270`. Therefore, we enter `270` when prompted for the address. When asked for the type of address, type **a** again:

```
TYPE H FOR HELP
+a
ADDR STOP ON
A=270
    000270  FA10D070 D07358DD 000498EC D00C07FE  ..........q.....
T(A-ADDR, E-DATA =, OR N-DATA <>)= a
```

We now type **t** for trace. Each instruction is listed before it is executed.

```
+t
TRACE SET
TRACE EP A=1433 ID=BC    370 A=000200 OP=47F0F058
TRACE EP A=1F9B ID=STM   370 A=000258 OP=90ECD00C
TRACE EP A=17D1 ID=ST    370 A=00025C OP=50D0F014
TRACE EP A=0CAD ID=LR    370 A=000260 OP=18ED
```

(continued)

```
TRACE EP A=1649 ID=LA    370 A=000262 OP=41D0F010
TRACE EP A=17D1 ID=ST    370 A=000266 OP=50D0E008
TRACE EP A=2093 ID=AP    370 A=00026A OP=FA10D070D072
ADDR STOP
    000270  FA10D070 D07358DD 000498EC D00C07FE  ..........q.....
TRACE EP A=2093 ID=AP    370 A=000270  OP=FA10D070D073
+
```

We can see that the next instruction to be executed is `FA10D070D073`. (You can see this same code on line 22 of the `.PRN` listing.) From our discussion of instruction formats, we know this is an `AP`, and this is confirmed by the message `ID=AP`. We can also tell by the `X'10'` following the `X'FA'` that we will be adding a one byte field to a two byte field. Remember: the instruction has not been executed yet. We can now display the data in question before allowing the program to proceed. To do so, we type **`d`** (for display). The data in question begin at location `000080`, so we add `X'200'` giving `000280`: we type **`280`** as the address when prompted:

```
+d
A=280
    000280  001C1C00  00000000 00000000 0007C6F8  ..............F8
    000290  00000000 00000000 00000000 00000000  ................
+
```

We can see that `SUM`, which was defined as `PL2'0'`, now has a value of `+1` (`X'001C'`), so we know that the first `AP` worked. But we can also see that `TWO` has a value of `X'00'`, which is not a valid packed decimal number. We have found the error. We would change the `.MLC` code, assemble, link, and run.

With practice, PC/370's testing facility will become an invaluable tool.

Exercises

1. True or false. (For each of the following assume A DC CL3'71D', B DC CL3' ', C DC PL2'-2', D DC PL2'3', and E DC CL2'-2'. Start with fresh data for each question.)

T F a. The hex representation of A is X'F7F1C4'.
T F b. The numeric bits of A are 0111, 0001, and 0010.
T F c. PACK B,A makes B = X'00071D'.
T F d. UNPK B,C makes B = X'00002D'.
T F e. MVZ A+2(1),=X'F0' makes A = X'F7F1FD'.
T F f. AP C,D makes D = X'001C'.
T F g. The hex representation of E is X'002D'.
T F h. SP D,C makes D = X'005C'.
T F i. SP D,D makes D = X'000C'.
T F j. SP E,E makes E = X'000C'.
T F k. CP C,D results in a condition code of 4 (low).
T F l. ZAP B,C will cause an abend.
T F m. ZAP B,E will cause an abend.

2. Determine the results of the following operations:

a. PACK B,A

A=	F1	F2	F3
B=			

b. PACK D,C

C=	F1	F2	F3	C4
D=				

c. PACK F,E

E=	D1
F=	

d. PACK H,G

G=	F1	F2	F3	C4
H=				

3. Determine the results of the following operations:

a. UNPK J,I

I=	12	34	5F		
J=					

b. UNPK L,K

K=	00	1C
L=		

c. UNPK N,M

M=	1D				
N=					

d. UNPK P,O

O=	12	3C
P=		

Exercises

4. Determine the results of the following operations:

a.	AP	B,A	A=	01	23	4F		
			B=	00	38	7F		*before*
			B=					*after*
b.	AP	D,C	C=	01	05	00	0C	
			D=	00	04	00	0D	*before*
			D=					*after*
c.	AP	F,E	E=	12	3C			
			F=	8F				*before*
			F=					*after*

5. Determine the results of the following operations:

a.	SP	B,A	A=	01	23	4F		
			B=	00	38	7F		*before*
			B=					*after*
b.	SP	D,C	C=	01	05	00	0C	
			D=	00	04	00	0D	*before*
			D=					*after*
c.	SP	F,E	E=	12	3C			
			F=	8F				*before*
			F=					*after*

6. Write the `BAL` code necessary to determine `TOTAL` where `TOTAL = GROSS + SALESTAX`. Use the following field definitions. Define any necessary work fields. `TOTAL` should be printable; that is, unsigned.

```
GROSS    DS    CL5
SALESTAX DS    CL4
TOTAL    DS    CL6
```

7. Write the `BAL` code necessary to determine `NET` where `NET = SALES - RETURNS`. Use the following field definitions. Define any necessary work fields. `NET` should be printable; that is, unsigned.

```
SALES    DS    CL5
RETURNS  DS    CL4
NET      DS    CL5
```

Exercises

8. Columns 8-14 of a card-image file contain the customer's account balance. Show the `PROCESS` and `WRAPUP` sections of a program which will display the number of customers and the total (sum) of the balances. Show all field definitions.

9. Given `PACK A,B` generated `F223D110D113` at `LOC=0000DA`.
   a. How long is `A`? `B`?
   b. What is the `LOC` of the next instruction?
   c. Show the object code if this line were changed to `PACK B,A`
   d. Show the object code if this line were changed to `UNPK B+2(2),A`

10. Given `AP C,D` generated `FA31D220D21E` at `LOC=0000C8`.
    a. How long is `C`? `D`?
    b. What is the `LOC` of the next instruction?
    c. Show the object code if this line were changed to `ZAP D,C`
    d. Show the object code if this line were changed to `SP  C+1(2),D+1(1)`

11. Write a program which will display a count of the number of courses offered in semester `W93`. Use the `OFFER` file of the Small Town Community College database. Your output should be by `WTO` only: there is no output file. Your message should appear as follows:

```
There were XXX courses offered in semester W93.
```

12. Modify program `COGS7A.MLC` to include totals by state; that is, your output should appear as follows:

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
                 COGSWORTH INDUSTRIES
                      Sales Recap

Product       Calif    Ill      Utah     Wisc    TOTAL
----------    -----    -----    -----    -----    -----
GIZMOS         020      030      020      020      090
WIDGETS        015      010      010      002      037
JUNQUE         025      015      015      018      073
----------    -----    -----    -----    -----    -----
TOTAL          060      055      045      040      200

003 records processed.
```

Exercises

13. Write a program which will produce counts by sex and marital status for the records in `STUDENT` file. Your output should appear as follows:

```
         1         2         3         4
1234567890123456789012345678901234567890
        STUDENT STATISTICS

Status      Male    Female   Total
--------   ------   ------   -----
Single      XXX      XXX      XXX
Married     XXX      XXX      XXX
--------   ------   ------   -----
Total       XXX      XXX      XXX
```

Note: A table of this type is called a "cross tabulation", or "cross tab".

14. a. (Refer to the Small Town Hardware Store database in More Datasets.) Write a program which will list *only* those items in the `TOOL` file which should be ordered. These are items where the sum of quantity on hand and quantity on order is less than or equal to the minimum quantity. The report should appear as follows:

```
         1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890
              SMALL TOWN HARDWARE STORE
                 ITEMS TO BE ORDERED

TID        Description        QOH + QOO = Sum   Min
---   ----------------------  ---   ---   ---   ---
XXX   XXXXXXXXXXXXXXXXXXXXXX  XXX   XXX   XXX   XXX
XXX   XXXXXXXXXXXXXXXXXXXXXX  XXX   XXX   XXX   XXX
XXX   XXXXXXXXXXXXXXXXXXXXXX  XXX   XXX   XXX   XXX
```

b. Modify the program in part (a) so that in addition to creating the report, a new `TOOL` file is created with the quantity on order field updated for those items that are ordered. The new quantity on order should be equal to the old quantity on order plus the economic order quantity. Use `DDNAME='NEWTOOL.DAT'` in the `DCB` for this new file. *All* tools should be written to this new file, even those for which there was no new order placed. Use `DOS'` `TYPE` command to view the file.

Exercises

15. a. Fill in the blanks (lines 15-19 and 26-29):

```
 STUFF7A                                             PAGE    1
PC/370 CROSS ASSEMBLER  OPTIONS=LXACE
   LOC                ADR1   ADR2 LINE LABEL    OP       OPERANDS
000000                               1 *++++++  BEGIN
000000                               2 STUFF7A  CSECT
000000                               3          USING    *,15
000000 47F0F058              0058    4          B        KZHQX001
000004 0B                            5          DC       AL1(11)
000005 E2E3E4C6C6F7C140              6          DC       CL11'STUFF7A '
000010 0000000000000000              7 HZQKX001 DC       18F'0'
000058 90ECD00C              000C    8 KZHQX001 STM      14,12,12(13)
00005C 50D0F014              0014    9          ST       13,HZQKX001+4
000060 18ED                         10          LR       14,13
000062 41D0F010              0010   11          LA       13,HZQKX001
000066 50D0E008              0008   12          ST       13,8(0,14)
00006A                              13          DROP     15
00006A                              14          USING    HZQKX001,13
00006A ____________   0097   0092   15          PACK     R,N
000070 ____________   0097   0095   16          AP       R,Q
000076 ____D080____   0090   0097   17          CP       M,R
00007C ________              0086   18          BE       S
000080 ____________   0095   0097   19          SP       Q,R
000086              00000086        20 S        EQU      *
000086                              21 *+++++++ RETURN
000086 58DD0004              0004   22          L        13,4(13)
00008A 98ECD00C              000C   23          LM       14,12,12(13)
00008E 07FE                         24          BR       14
000090                              25          LTORG
000090 ____                         26 M        DC       PL2'116'
000092 ______                       27 N        DC       CL3'123'
000095 ____                         28 Q        DC       PL2'-7'
000097 ________                     29 R        DC       PL4'5'
0000A0                              30          END
```

b. You be the computer...what are the values of `M`,`N`,`Q`, and `R` after this program runs?

M=__________ N=__________ Q=__________ R=__________

c. Key and assemble this program. Use PC/370's test facility to stop the program at `S`, and verify your answers to part (b). *(Note: Lines 1-14 of the program are the expansion of the* `BEGIN` *macro, and lines 21-24 are the expansion of the* `RETURN` *macro: code those macro statements as you u*

Exercises

16. a. Fill in the blanks (lines 15-19 and 24-27):

```
 STUFF7B                                             PAGE    1
PC/370 CROSS ASSEMBLER  OPTIONS=LXACE
   LOC               ADR1    ADR2 LINE LABEL    OP       OPERANDS
000000                               1 *+++++++ BEGIN
000000                               2 STUFF7B  CSECT
000000                               3          USING    *,15
000000 47F0F058             0058     4          B        KZHQX001
000004 0B                            5          DC       AL1(11)
000005 E2E3E4C6C6F7C240              6          DC       CL11'STUFF7B '
000010 0000000000000000              7 HZQKX001 DC       18F'0'
000058 90ECD00C             000C     8 KZHQX001 STM      14,12,12(13)
00005C 50D0F014             0014     9          ST       13,HZQKX001+4
000060 18ED                         10          LR       14,13
000062 41D0F010             0010    11          LA       13,HZQKX001
000066 50D0E008             0008    12          ST       13,8(0,14)
00006A                              13          DROP     15
00006A                              14          USING    HZQKX001,13
00006A F823D08BD091   009B   00A1   15          ______   ______________
000070 F322D08ED08B   009E   009B   16          ______   ______________
000076 D300D090D088   00A0   0098   17          ______   ______________
00007C D201D089D08F   0099   009F   18          ______   ______________
000082 FB32D091D08B   00A1   009B   19          ______   ______________
000088                              20 *+++++++ RETURN
000088 58DD0004             0004    21          L        13,4(13)
00008C 98ECD00C             000C    22          LM       14,12,12(13)
000090 07FE                         23          BR       14
000098                              24          LTORG
000098 F0                           24                   X'F0'
000099                              25 W        ______   ______________
00009B F5F5F5                       26 X        ______   ______________
00009E C1C2C3                       27 Y        ______   ______________
0000A1 0001234C                     28 Z        ______   ______________
0000A8                              29          END
```

b. You be the computer...what are the values of W, X, Y, and Z after this program runs?

W=_________ X=_________ Y=_________ Z=_________

c. Key and assemble this program. Use PC/370's test facility to stop the program before the RETURN, and verify your answers to part (b). *(Note: Lines 1-14 of the program are the expansion of the BEGIN macro, and lines 20-23 are the expansion of the RETURN macro: code those macro statements as you usually do.)*