Chapter 14
Binary Arithmetic

Objectives

Upon completion of this chapter you will be able to:

- Explain what is meant by doubleword, fullword, and halfword boundary alignment,
- Given a field's `LOC` indicate if it is doubleword, fullword, and/or halfword aligned,
- Use the following binary instructions: `L`, `LH`, `LR`, `A`, `AH`, `AR`, `S`, `SH`, `SR`, `C`, `CH`, and `CR`,
- Use the `CVB` instruction to convert a packed number to binary, and
- Use the `CVD` instruction to convert a binary number to packed.

Introduction

The System/370 computers can only do two types of arithmetic: packed and binary. We have already looked at packed arithmetic in detail. In this chapter we will discuss binary arithmetic. Binary arithmetic is also referred to as register arithmetic since each binary arithmetic operation will involve at least one register. There are several reasons why we would need to know binary arithmetic in addition to packed arithmetic:

- The file you are reading may have been created with binary fields.
- Binary arithmetic is required when processing variable length records.
- In some cases, saving numbers in binary form rather than in packed form can result in less storage. If the data is to be transmitted over a communications system, there is the additional benefit of reduced data transmission.
- Table processing in BAL (the subject of our next chapter) requires binary arithmetic.

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

Recall that computers are based on the binary numbering system, and that we use the hexadecimal numbering system as a convenient abbreviation of the binary. Each byte of memory has associated with it an address, which we can show in hexadecimal form. As with many things in computing, these address locations are numbered relative to zero; that is, they start at zero rather than one. So memory can be represented as follows:

000000	*(4 bytes*	*each column)*	00000F
FFFFF0			FFFFFF

Notice that each "row" in the figure is sixteen bytes in length (the rightmost, or low order, digit of the addresses range from `0` to `F` inclusive.) Memory is divided into "words", each of which is four bytes in length.

The first word occupies bytes `000000-000003` inclusive, the second word occupies bytes `000004-000007` inclusive, the third word occupies bytes `000008-00000B` inclusive. This continues up to and including the last word which occupies bytes `FFFFFC-FFFFFF` inclusive.

Recall from an earlier discussion that each register also occupies four bytes, and that it is the register size which determines the word size of a computer.

If the rightmost digit of the address of the first byte of a field is `0`, `4`, `8`, or `C` then we say the field is "fullword aligned", or "aligned on a fullword boundary". In the following example, fields `A` and `D` are fullword aligned. Why?

```
LOC
020104     A        DS   PL3
020107     B        DS   CL2
020109     C        DS   CL3
02010C     D        DS   CL4
```

We use a field type specification of `F` to define a fullword. For example:

```
FIVETHOU DC F'5000'
```

A field defined with a type specification of `F` is *forced* to a fullword boundary. This can be a *very* important consideration when defining record layouts. Consider the following example. Can you supply the missing `LOC` fields?

```
020117     INREC   DS   0CL80
______     INZIP   DS   PL3
______     INCODE  DS   CL1
______     INAMT   DS   F
______     INJUNK  DS   CL6
______     INLIMIT DS   F
```

(You should end up with `INLIMIT` starting at `LOC=020128` because `02011B`, `020126`, and `020127` are "skipped".)

I can solve the problem of skipping `02011B` by forcing the entire record to begin on a fullword boundary. This is done with a `DS 0F` as shown. Once again, can you supply the missing `LOC`s?

```
020118             DS   0F
______     INREC   DS   0CL80
______     INZIP   DS   PL3
______     INCODE  DS   CL1
______     INAMT   DS   F
______     INJUNK  DS   CL6
______     INLIMIT DS   F
```

(Again, you should end up with `INLIMIT` starting at `LOC=020128` because `020126`, and `020127` are still "skipped".)

Given these fields are defined in the order shown, there is no way I can force `INAMT` *and* `INLIMIT` to a fullword boundary. When using a record layout with fields defined as fullwords, it is common practice to:

- Use `DS 0F` to force the beginning of the record to a fullword boundary, and
- Make sure all fields defined as type `F` begin in column `(n*4)+1`; that is, columns 1, 5, 9, 13, ...

This second rule can be accomplished by defining a record such that all type `F` fields are defined before any other fields. This is a common practice in some shops. In fact, some installations will go so far as to say that all record layouts will be defined such that their total length is a multiple of four, even if this means "padding" the record with a few bytes of "filler". Of course, these are considerations made at the time the record layout is designed: *these are design issues which should have been answered before the program gets to you!*

Given a fullword is four bytes in length, it follows that a **doubleword** is eight bytes long. We use a field type specification of `D` to define a doubleword. For example:

```
BILLION  DC    D'1000000000'
```

If the rightmost digit of the address of the first byte of a field is `0` or `8` then we say it is "doubleword aligned" or "aligned on a doubleword boundary". A field defined with a type specification of `D` is *forced* to a doubleword boundary.

Similarly, a **halfword** is two bytes long. We use a field type specification of `H` to define a halfword. For example:

```
MINUSONE DC    H'-1'
```

If the rightmost digit of the address of the first byte of a field is `0`, `2`, `4`, `6`, `8`, `A`, `C`, or `E` then we say it is "halfword aligned" or "aligned on a halfword boundary". A field defined with a type specification of `H` is *forced* to a halfword boundary.

The following table may help to illustrate the "hierarchy" of doublewords, fullwords, halfwords, and bytes.

<table>
<tr><td colspan="8" align="center">DBL</td></tr>
<tr><td colspan="4" align="center">FULL1</td><td colspan="4" align="center">FULL2</td></tr>
<tr><td colspan="2">HALF1</td><td colspan="2">HALF2</td><td colspan="2">HALF3</td><td colspan="2">HALF4</td></tr>
<tr><td>BYTE1</td><td>BYTE2</td><td>BYTE3</td><td>BYTE4</td><td>BYTE5</td><td>BYTE6</td><td>BYTE7</td><td>BYTE8</td></tr>
</table>

How might these fields be defined in a program?

Solution

```
DBL      DS    0D     1-8
FULL1    DS    0F     1-4
HALF1    DS    0H     1-2
BYTE1    DS    CL1    1-1
BYTE2    DS    CL1    2-2
HALF2    DS    0H     3-4
BYTE3    DS    CL1    3-3
BYTE4    DS    CL1    4-4
FULL2    DS    0F     5-8
HALF3    DS    0H     5-6
BYTE5    DS    CL1    5-5
BYTE6    DS    CL1    6-6
HALF4    DS    0H     7-8
BYTE7    DS    CL1    7-7
BYTE8    DS    CL1    8-8
```

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

1. The following record is defined as 24 bytes, but will actually occupy more than that. Supply the missing `LOC`s and determine the number of bytes occupied by this record given that the `LOC` of the first byte is `020100`. (You should end up with `LOC=02011C` for `BIG7`.) Then reorder the fields in the record so that the record does, in fact, occupy 24 bytes only. (You should end up with `LOC=020117` for `BIG7`.)

```
LOC
020100     BIGMESS  DS    0CL24
______     BIG1     DS    CL5     1-5
______     BIG2     DS    D       6-13
______     BIG3     DS    CL3     14-16
______     BIG4     DS    H       17-18
______     BIG5     DS    CL1     19
______     BIG6     DS    F       20-23
______     BIG7     DS    CL1     24
```

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

Having discussed the three sizes of binary fields - doubleword, fullword, and halfword - we now look at the "capacity" of these fields; that is, the range of values that each can hold. We begin with the halfword as it is the easiest. We know that a halfword occupies two bytes, and we know that each byte can be represented by two hex digits, each of which represents four bits. Recall from our discussion of packed numbers, the rightmost hex digit represented the sign (`F`, `D`, or `C`). Clearly, some similar mechanism must be used to indicate the sign of a binary number.

The key to understanding binary numbers is to recognize that the leftmost (high order) bit is used to represent the sign: this bit will be off (`0`) for positive numbers and on (`1`) for negative numbers. (Note: Whereas a packed number could be unsigned, there are no unsigned binary numbers.) Therefore, the range of values for a *positive* halfword is:

Binary	**0**000	0000	0000	0000	$= 0_{10}$
Hex	0	0	0	0	

thru...

Binary	**0**111	1111	1111	1111	$= 32767_{10}$
Hex	7	F	F	F	

The range of values for a *negative* halfword is:

Binary	**1**111	1111	1111	1111	$= -1_{10}$
Hex	F	F	F	F	

thru...

Binary	**1**000	0000	0000	0000	$= -32768_{10}$
Hex	8	0	0	0	

The fact that `X'FFFF'` = `-1` is counterintuitive. But think of it this way...if we add `X'0001'` to `X'FFFF'` we get `X'10000'`, which when truncated to two bytes is `X'0000'`. Thus we see that `-1 + 1 = 0`. Therefore, `X'FFFF'` must equal `-1`.

A two byte packed field can range in value from -999 to +999 only. But as we have seen, a halfword can range in value from -32768 to +32767 inclusive. Now assume that we want to store a value which will never go out of that range. To store -32768 or +32767 in packed form would require three bytes of storage (`X'32768D'` and `X'32767C'`). But to store these same values in a halfword requires only two bytes of storage; a savings of 33%.

We know that a fullword occupies four bytes. Following the same pattern as above; that is, using the leftmost bit to indicate the sign, we see that the range of values for a *positive* fullword is:

Binary	**0**000	0000	0000	0000	0000	0000	0000	0000	$= 0_{10}$
Hex	0	0	0	0	0	0	0	0	

thru...

Binary	**0**111	1111	1111	1111	1111	1111	1111	1111	$= 2,147,483,647_{10}$
Hex	7	F	F	F	F	F	F	F	

The range of values for a *negative* fullword is:

Binary	**1**111	1111	1111	1111	1111	1111	1111	1111	$= -1_{10}$
Hex	F	F	F	F	F	F	F	F	

thru...

Binary	**1**000	0000	0000	0000	0000	0000	0000	0000	$= -2,147,483,648_{10}$
Hex	8	0	0	0	0	0	0	0	

Note that it would require six bytes to store this same number in packed format. (But, of course, those six bytes could store a packed number in the range ±99,999,999,999.)

The range of values for a doubleword are -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (big enough for most businesses!)

Comparing Two Binary Fields

Any binary arithmetic operation (including a comparison) will require at least one register. Suppose `A` and `B` are each fullwords. `A` has a value of +5 and `B` has a value of -5. They could be shown (in hex) as:

A= | 00 | 00 | 00 | 05 | B= | FF | FF | FF | FB |

We cannot use `CLC` to compare the fields: `CLC A,B` would result in `A` is low, which is clearly not the case. Instead, we use the load instruction (`L`) to place each fullword in a separate register, and the compare register instruction (`CR`) to compare the two registers:

```
L      R3,A
L      R4,B
CR     R3,R4
```

Alternatively, we could load just one of these fields into a register and then use the compare instruction (`C`) to compare that register to the other fullword:

```
L      R3,A            or...         L      R4,B
C      R3,B                          C      R4,A
```

What if we were comparing two halfwords? For example, suppose `C` is a halfword with a value of +2 and `D` is a halfword with a value of -2:

C= | 00 | 02 | D= | FF | FE |

We can use the load halfword instruction (`LH`) to place each halfword in a separate register, and the compare register instruction (`CR`) to compare the two registers:

```
LH     R3,C
LH     R4,D
CR     R3,R4
```

Alternatively, we could load just one of these fields into a register and then use the compare halfword instruction (`CH`) to compare that register to the other halfword:

```
LH     R3,C            or...         LH     R4,D
CH     R3,D                          CH     R4,C
```

Note that when the `LH` instruction is used, the sign is preserved; that is, the sign bit of the halfword (leftmost bit) is propagated throughout the remainder of the register. This can be shown as:

D=					
Binary	1111	1111	1111	1110	$D = -2_{10}$
Hex	F	F	F	E	

...after `LH R4,D` would becomes...

Binary	**1**111	1111	1111	1111	1111	1111	1111	1110	$R4 = -2_{10}$
Hex	F	F	F	F	F	F	F	E	

... *not* ...

Binary	**0**000	0000	0000	0000	1111	1111	1111	1110	$R4 = +32{,}766_{10}$
Hex	0	0	0	0	F	F	F	E	

Be very careful that you use `L` to load a fullword and `LH` to load a halfword. Failure to do so will not cause a decimal exception (`S0C7`), but will certainly give erroneous results. For example, if `C` and `D` had been defined as above, *and in that order*, and if I had use `L` (instead of `LH`) to put `C` into register 3, that register would then contain:

Binary	0000	0000	0000	0010	1111	1111	1111	1110	$R3 = +196{,}606_{10}$
Hex	0	0	0	2	F	F	F	E	

It should be pointed out that any bit combination is a valid binary number. Therefore, unlike packed decimal arithmetic, binary arithmetic will never produce a data exception abend (`S0C7`). This does not mean that binary math should be favored over packed decimal math. As we've just shown, you can still make a mistake by treating binary fields incorrectly. The result is errors without warning. At least packed decimal operations warn you (however unpleasant that warning may be) that you have used nonnumeric data.

For completeness, we mention here that the load register instruction (`LR`) copies the value of one register (the second operand) to another register (the first operand).

You Try It...

2. Given `FULL` is a fullword and `HALF` is a halfword, show three different ways to compare `FULL` and `HALF`. In each case, branch to the label `LOW` if the value of `FULL` is less than the value of `HALF`.

Moving a Register to a Fullword or Halfword

In chapter four we saw that the opposite of the load instruction (`L`) is the store instruction (`ST`): the store instruction copies the value of a register to a fullword.

Similarly, the opposite of the load halfword instruction (`LH`) is the store halfword instruction (`STH`): the store halfword instruction will copy the value of a register to a halfword, maintaining the itegrity of the sign but also *truncating if necessary and without warning*.

What should you do if you know the value of a fullword will fit in a halfword and you want to copy that fullword to a halfword? For example, given:

```
FULL     DS    F
HALF     DS    H
```

...you might be tempted to do the following:

```
MVC    HALF,FULL+2
```

...but don't do it! You *must* go through a register, otherwise you compromise the integrity of the sign. The correct solution is:

```
L      R5,FULL
STH    R5,HALF
```

You Try It...

3. Write the code necessary to move the value of `HALF` to `FULL`.
4. Write the code necessary to move the greater of `HALF` and `FULL` to `MAX`, where `MAX` is defined as a fullword.

Binary Addition

A register is *always* the receiving field for any binary addition instruction. First, use the `L`, `LH`, or `LR` instruction to place one operand into a register. Then, use the add (`A`), add halfword (`AH`), or add register (`AR`) instruction depending on if the second operand is fullword, halfword, or register respectively. For example, given:

```
A      DS    F
B      DS    F
C      DS    H
D      DS    F
```

...to add `A + B + C` giving `D`, we could code:

```
L      R3,A
A      R3,B
AH     R3,C
ST     R3,D
```

Another example...Given `A` is a fullword, add 1 to `A`. There are many ways we can do this, including:

```
L      R3,A            or...           L      R3,A
A      R3,=F'1'                        AH     R3,=H'1'
ST     R3,A                            ST     R3,A
```

Again, there is no way to add to a binary number without going through a register. Put another way, *all binary math instructions are type RR or RX only: there are no type SS binary math instructions*.

You Try It...

5. Given `F1` and `F2` are fullwords, and `H1` and `H2` are halfwords. Write the code necessary to put the lesser of `(F1+F2)` and `(H1+H2)` in register 7. Your code should not change the values of `F1`, `F2`, `H1`, and `H2`. Do not define any other work fields. You will need two registers.

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

Binary Subtraction

For binary subtraction, as with binary addition, first use the `L`, `LH`, or `LR` instruction to place one operand into a register. Then, use the subtract (`S`), subtract halfword (`SH`), or subtract register (`SR`) instruction depending on if the second operand is fullword, halfword, or register respectively. For example, given:

```
A      DS     F
B      DS     F
C      DS     H
```

...to compute `A = A-(B+C)`, we first recognize that this is equivalent to `A = A-B-C`. We then code:

```
L      R3,A
S      R3,B
SH     R3,C
ST     R3,A
```

You Try It...

6. Given `F1` and `F2` are fullwords, and `H1` and `H2` are halfwords. Write the necessary code to compute `F1=(F1-F2)+(H1-H2)`.

Converting from Packed to Binary (CVB)

Assume that your input data is in character (zoned decimal) or packed form. How do you convert these numbers to their binary equivalents? This is done with the convert to binary instruction (`CVB`). This instruction converts a packed number stored in a doubleword, into a binary number stored in a register. For example, given:

```
IAMOUNT  DS   CL4    Input amount, unpacked, 99V99
OAMOUNT  DS   H      Output amount, binary, 99V99
DBLWORD  DS   D      Doubleword work area
```

We code the following:

```
PACK  DBLWORD,IAMOUNT   Must be packed to use CVB
CVB   R3,DBLWORD        Binary equivalent in R3
STH   R3,OAMOUNT        Binary equivalent to output
```

Note that the second operand of the `CVB` must be a valid packed number. If it is not, then you will get a data exception abend. Furthermore, this packed number must occupy all eight bytes of the doubleword.

It may seem a little strange that a packed number is being stored in a doubleword. But we can think of a doubleword in this case as eight bytes which happen to be doubleword aligned. If `DBLWORD` had been defined as `DC D'0'`, then it does not begin as a valid packed number, but the `PACK` instruction will replace the binary zero with a packed number.

Converting from Binary to Packed (CVD)

The opposite of `CVB` is `CVD`: convert to decimal. This instruction will convert the contents of register into a packed number and store that packed number in a doubleword. The `CVD` instruction, like the `ST` instruction, is one of the few instructions where the second operand specifies the receiving field (recall that in most cases, the second operand is the sending field and the first operand is the receiving field.)

You will want to use the `CVD` instruction to convert to (packed) decimal if for no other reason than to print the number. There is no equivalent to the `ED` instruction for binary numbers: a binary number must be converted to packed form before editing. Consider the following example. Given:

```
GROSS    DS    F     Gross sales, 99999V99
SALESTAX DS    F     Sales tax,  9999V99
NET      DS    CL10  Net = Gross + Tax, BZZ,ZZ9.99
DBLWORD  DS    D     Work field
```

...then to add `SALESTAX` to `GROSS` giving `NET`, formatted, we could code:

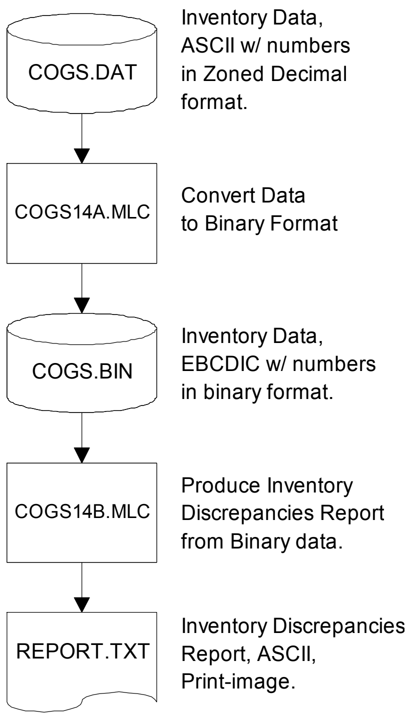
```
L     R4,GROSS       Gross sales in R4
A     R4,SALESTAX    Add sales tax
CVD   R4,DBLWORD     Convert to packed for printing
MVC   NET,=X'4020206B2021204B2020'   BZZ,ZZ9.99
ED    NET,DBLWORD+4  Seven digits in NET so edit the
                     last four bytes of DBLWORD only
```

You Try It...

7. Given `FIVE DC H'5'`, `THREE DC H'3'`, and `RESULT DC X'40202120'`, write a short program which will subtract `THREE` from `FIVE` giving `RESULT`. Use `WTO` to display `RESULT`. Define other fields as necessary, but do all arithmetic in binary; that is, use packed fields only as required for the `ED` command.

Sample Program - Converting Cogsworth's Data to Binary

To further illustrate the concepts which have been introduced above, we will look at two programming examples. Recall the Inventory Discrepancies report for Cogsworth Industries, presented in an earlier chapter. The inventory file was `COGS.DAT`. The numeric fields in that file were stored in zoned decimal format. We will write two programs. The first program will create a new file with all numeric fields stored as halfwords. The second program will read this new file and produce the Inventory Discrepancies report. This is shown in the system flowchart to the right.



Let's look first at the conversion program, `COGS14A.MLC`.

- All numeric fields will be stored as halfwords. Recall, therefore, from our earlier discussion that it is essential that we guarantee halfword alignment. The output record layout is shown below. Note that the halfwords can be defined after the description since the length of the description (10) is an even multiple of the length of a halfword (2).

```
         DS    0H                Force halfword alignment
OREC     DS    0CL28       1-28  Inventory record
ODESC    DS    CL10        1-10  Product description
OCALIF   DS    H          11-12  Units sold in Calif
OILL     DS    H          13-14  Units sold in Illinois
OUTAH    DS    H          15-16  Units sold in Utah
OWISC    DS    H          17-18  Units sold in Wisconsin
OBEGIN   DS    H          19-20  Beginning inventory
OPURCH   DS    H          21-22  Purchases throughout year
OQOH     DS    H          23-24  Actual quantity on hand
OCOST    DS    H          25-26  Cost (each) 99V99
OSELL    DS    H          27-28  Sell for (each) 99V99
```

Note that there is no `CR/LF` at the end of the record. The `CR/LF` is not required: we have used it in our earlier programs so that we could use DOS' `TYPE` command to view our data. Since this record contains binary numbers, the only field we could "view" is the description, so we will omit the `CR/LF`.

- We need a DCB for both files. We show here the DCB for the output file. There is nothing unusual here. Of course, the LRECL must match the length shown in the output record layout above (28).

```
BINARY   DCB   LRECL=28,RECFM=F,MACRF=P,
               DDNAME='COGS.BIN'
```

- Since our data contains binary data, ***we cannot use the EBCDIC to ASCII conversion!*** That type of conversion simply will not work, as there is not a one-to-one correspondence between the EBCDIC and ASCII codes. So we will omit the OI instruction which we are used to seeing prior the OPEN statement.

```
         OI    INVENTRY+10,X'08'  PC/370 ONLY - Convert all
*                                 input from ASCII to EBCDIC
         OPEN  INVENTRY
         OPEN  BINARY             NOTE: Output in EBCDIC
```

  Of course, to read this new file, the programmer must know that it is already EBCDIC and is without CR/LF.

- Within the FORMAT routine, each numeric field will be packed, converted to binary, and moved to the corresponding output field. The code to do so is as follows:

```
         PACK  DBL,ICALIF         Convert CALIF to binary
         CVB   R3,DBL
         STH   R3,OCALIF
         PACK  DBL,IILL           Convert ILL to binary
         CVB   R3,DBL
         STH   R3,OILL
         PACK  DBL,IUTAH          Convert UTAH to binary
         CVB   R3,DBL
         STH   R3,OUTAH
         PACK  DBL,IWISC          Convert WISC to binary
         CVB   R3,DBL
         STH   R3,OWISC
         PACK  DBL,IBEGIN         Convert BEGIN to binary
         CVB   R3,DBL
         STH   R3,OBEGIN
         PACK  DBL,IPURCH         Convert PURCH to binary
         CVB   R3,DBL
         STH   R3,OPURCH
         PACK  DBL,IQOH           Convert QOH to binary
         CVB   R3,DBL
         STH   R3,OQOH
         PACK  DBL,ICOST          Convert COST to binary
         CVB   R3,DBL
         STH   R3,OCOST
         PACK  DBL,ISELL          Convert SELL to binary
         CVB   R3,DBL
         STH   R3,OSELL
```

The complete program, COGS14A.MLC, follows.

```
         PRINT NOGEN
****************************************************************
*        FILENAME:  COGS14A.MLC                                *
*        AUTHOR  :  Bill Qualls                                *
*        SYSTEM  :  PC/370 R4.2                                *
*        REMARKS :  Create binary data file using COGS.DAT     *
****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'COGS14A ... Begin execution'
         BAL   R10,SETUP
MAIN     EQU   *
         CLI   EOFSW,C'Y'
         BE    EOJ
         BAL   R10,PROCESS
         B     MAIN
EOJ      EQU   *
         BAL   R10,WRAPUP
         WTO   'COGS14A ... 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
         OPEN  INVENTRY
         OPEN  BINARY              NOTE: Output in EBCDIC
         BAL   R10,READ
         L     R10,SVSETUP
         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
         B     READX
ATEND    EQU   *
         MVI   EOFSW,C'Y'
READX    EQU   *
         L     R10,SVREAD
         BR    R10
****************************************************************
*        FORMAT - Format a single detail line.                 *
****************************************************************
```

(continued)

```
FORMAT   EQU   *
         ST    R10,SVFORM
         MVC   ODESC,IDESC
         PACK  DBL,ICALIF          Convert CALIF to binary
         CVB   R3,DBL
         STH   R3,OCALIF
         PACK  DBL,IILL            Convert ILL to binary
         CVB   R3,DBL
         STH   R3,OILL
         PACK  DBL,IUTAH           Convert UTAH to binary
         CVB   R3,DBL
         STH   R3,OUTAH
         PACK  DBL,IWISC           Convert WISC to binary
         CVB   R3,DBL
         STH   R3,OWISC
         PACK  DBL,IBEGIN          Convert BEGIN to binary
         CVB   R3,DBL
         STH   R3,OBEGIN
         PACK  DBL,IPURCH          Convert PURCH to binary
         CVB   R3,DBL
         STH   R3,OPURCH
         PACK  DBL,IQOH            Convert QOH to binary
         CVB   R3,DBL
         STH   R3,OQOH
         PACK  DBL,ICOST           Convert COST to binary
         CVB   R3,DBL
         STH   R3,OCOST
         PACK  DBL,ISELL           Convert SELL to binary
         CVB   R3,DBL
         STH   R3,OSELL
         L     R10,SVFORM
         BR    R10
****************************************************************
*        WRITE - Write a single output record.                 *
****************************************************************
WRITE    EQU   *
         ST    R10,SVWRITE
         PUT   BINARY,OREC
         AP    #OUT,=P'1'
         L     R10,SVWRITE
         BR    R10
****************************************************************
*        WRAPUP - Those things which happen one time only,     *
*                 after all records have been processed.       *
****************************************************************
WRAPUP   EQU   *
         ST    R10,SVWRAP
         CLOSE INVENTRY
         CLOSE BINARY
         WTO   'COGS14A ... Binary file COGS.BIN created.'
         ED    MSG#OUT,#OUT
         WTO   MSG
         L     R10,SVWRAP
         BR    R10
****************************************************************
*        Literals, if any, will go here                        *
****************************************************************
         LTORG
```

(continued)

```
************************************************************
*        File definitions                                  *
************************************************************
INVENTRY DCB   LRECL=41,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='COGS.DAT'
BINARY   DCB   LRECL=28,RECFM=F,MACRF=P,
               DDNAME='COGS.BIN'
************************************************************
*        RETURN ADDRESSES                                  *
************************************************************
SVSETUP  DC    F'0'                SETUP
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)
#OUT     DC    PL2'0'              Count of records written
DBL      DC    D'0'                To convert packed to binary
         COPY  COGS
************************************************************
*        Output record definition                          *
************************************************************
         DS    0H                  Force halfword alignment
OREC     DS    0CL28         1-28  Inventory record
ODESC    DS    CL10          1-10  Product description
OCALIF   DS    H            11-12  Units sold in Calif
OILL     DS    H            13-14  Units sold in Illinois
OUTAH    DS    H            15-16  Units sold in Utah
OWISC    DS    H            17-18  Units sold in Wisconsin
OBEGIN   DS    H            19-20  Beginning inventory
OPURCH   DS    H            21-22  Purchases throughout year
OQOH     DS    H            23-24  Actual quantity on hand
OCOST    DS    H            25-26  Cost (each) 99V99
OSELL    DS    H            27-28  Sell for (each) 99V99
************************************************************
*        Output message (count of records written)         *
************************************************************
MSG      DS    0CL32
         DC    CL11'COGS14A ...'
MSG#OUT  DC    XL4'40202120'
         DC    CL17' records written.'
         END   BEGIN
```

Sample Program - Inventory Discrepancies Report from the Binary File

We now look at the second program, `COGS14B.MLC`. This program will produce the Inventory Discrepancies report from the new (binary) file. (Note this program is a modification of `COGS9B.MLC`, which produced the same report from the zoned decimal file.)

- The input record contains halfwords, therefore it must be halfword aligned just as when the record was created.

```
         DS    0H                  Force halfword alignment
IREC     DS    0CL28        1-28   Inventory record
```

- The `LRECL` of the input file `DCB` must match the length shown in the input record layout (28).

```
INVENTRY DCB   LRECL=28,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='COGS.BIN'
```

- The input file was created in EBCDIC form, therefore we omit the `OI` instruction we are used to seeing prior to the `OPEN` statement.

```
         OI    REPORT+10,X'08'     PC/370 ONLY - Convert all
*                                  output from EBCDIC to ASCII
         OPEN  INVENTRY            NOTE: Input in EBCDIC
         OPEN  REPORT
```

- All calculations are done in binary. See the `FORMAT` section. We used four registers for these calculations. We could have used fewer registers if we had been willing to store some intermediate results.

  It should be apparent in looking at the code in the `FORMAT` section that the code could become very confusing quite quickly. Liberal use of meaningful comments is essential!

The complete program, `COGS14B.MLC`, follows.

```
         PRINT NOGEN
****************************************************************
*        FILENAME:  COGS14B.MLC                                *
*        AUTHOR  :  Bill Qualls                                *
*        SYSTEM  :  PC/370 R4.2                                *
*        REMARKS :  Produce report for COGSWORTH INDUSTRIES    *
*                   showing inventory discrepancies.           *
*                   Modify COGS9B.MLC to use binary input.     *
****************************************************************
         START 0
         REGS
BEGIN    BEGIN
         WTO   'COGS14B ... Begin execution'
         BAL   R10,SETUP
MAIN     EQU   *
         CLI   EOFSW,C'Y'
         BE    EOJ
         BAL   R10,PROCESS
         B     MAIN
EOJ      EQU   *
         BAL   R10,WRAPUP
         WTO   'COGS14B ... Normal end of program'
         RETURN
```

(continued)

```
**************************************************************
*        SETUP - Those things which happen one time only,     *
*                before any records are processed.            *
**************************************************************
SETUP    EQU   *
         ST    R10,SVSETUP
         OI    REPORT+10,X'08'    PC/370 ONLY - Convert all
*                                 output from EBCDIC to ASCII
         OPEN  INVENTRY           NOTE: Input in EBCDIC
         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
         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        Description
         LH    R3,IBEGIN          Beginning inventory
         CVD   R3,DBL
         MVC   OBEGIN,WMASK
         ED    OBEGIN,DBL+6
```

(continued)

```
         LH    R4,IPURCH          Purchases
         CVD   R4,DBL
         MVC   OPURCH,WMASK
         ED    OPURCH,DBL+6
         LH    R5,ICALIF          Each product's sales
         AH    R5,IILL             by state must be added to
         AH    R5,IUTAH              get total for product...
         AH    R5,IWISC
         CVD   R5,DBL
         MVC   OSALES,WMASK
         ED    OSALES,DBL+6
         LR    R6,R3              Ending Inventory =
         AR    R6,R4                Beginning + Purchases
         SR    R6,R5                 - Sales
         CVD   R6,DBL
         MVC   OENDING,WMASK
         ED    OENDING,DBL+6
         LH    R3,IQOH            Actual ending inventory
         CVD   R3,DBL               (Reusing register 3)
         MVC   OQOH,WMASK           (Reusing register 3)
         ED    OQOH,DBL+6
         SR    R6,R3              Difference =
         CVD   R6,DBL               Expected - Actual
         MVC   ODIFF,WMASK2
         ED    ODIFF,DBL+6
         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
         CLOSE INVENTRY
         CLOSE REPORT
         WTO   'COGS14B ... Discrepancies report on REPORT.TXT'
         L     R10,SVWRAP
         BR    R10
****************************************************************
*        Literals, if any, will go here                        *
****************************************************************
         LTORG
****************************************************************
*        File definitions                                      *
****************************************************************
INVENTRY DCB   LRECL=28,RECFM=F,MACRF=G,EODAD=ATEND,
               DDNAME='COGS.BIN'
REPORT   DCB   LRECL=62,RECFM=F,MACRF=P,
               DDNAME='REPORT.TXT'
```

(continued)

```
***************************************************************
*         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' '
WMASK    DC    X'40202120'         BZZ9
WMASK2   DC    X'4020202060'       BZZZ-
DBL      DC    D'0'                For packed/binary conversions
***************************************************************
*         Input record definition                             *
***************************************************************
         DS    0H                  Force halfword alignment
IREC     DS    0CL28        1-28   Inventory record
IDESC    DS    CL10         1-10   Product description
ICALIF   DS    H           11-12   Units sold in Calif
IILL     DS    H           13-14   Units sold in Illinois
IUTAH    DS    H           15-16   Units sold in Utah
IWISC    DS    H           17-18   Units sold in Wisconsin
IBEGIN   DS    H           19-20   Beginning inventory
IPURCH   DS    H           21-22   Purchases throughout year
IQOH     DS    H           23-24   Actual quantity on hand
ICOST    DS    H           25-26   Cost (each) 99V99
ISELL    DS    H           27-28   Sell for (each) 99V99
***************************************************************
*         Output (line) definition                            *
***************************************************************
OREC     DS    0CL62        1-62
ODESC    DS    CL10         1-10   Product description
         DS    CL3         11-13
OBEGIN   DS    CL4         14-17   Beginning inventory
         DS    CL4         18-21
OPURCH   DS    CL4         22-25   Purchases
         DS    CL4         26-29
OSALES   DS    CL4         30-33   Units sold
         DS    CL5         34-38
OENDING  DS    CL4         39-42   Ending inventory (expected)
         DS    CL4         43-46
OQOH     DS    CL4         47-50   Ending inventory (actual)
         DS    CL4         51-54
ODIFF    DS    CL5         55-59   Difference
         DS    CL1         60-60
OCRLF    DS    CL2         61-62   PC/370 only - CR/LF
***************************************************************
*         Headings definitions                                *
***************************************************************
HD1      DS    0CL62
         DC    CL40'                     COGSWORTH INDUSTRIES'
         DC    CL20' '
         DC    XL2'0D25'
```

(continued)

```
HD2      DS    0CL62
         DC    CL40'              Inventory Discrepancies R'
         DC    CL20'eport'
         DC    XL2'0D25'
HD3      DS    0CL62
         DC    CL60' '
         DC    XL2'0D25'
HD4      DS    0CL62
         DC    CL40'Product      Begin + Purch - Sales = Exp'
         DC    CL20'ect   Actual   Diff'
         DC    XL2'0D25'
HD5      DS    0CL62
         DC    CL40'----------   -----   -----   -----   ---'
         DC    CL20'---   ------   ----'
         DC    XL2'0D25'
         END   BEGIN
```

Viewing a Register While Testing

We will now see how to view a register while using PC/370's `TEST` facility. Consider the following, which was presented earlier as an exercise:

> Given `FIVE DC H'5'`, `THREE DC H'3'`, and `RESULT DC X'40202120'`, write a short program which will subtract `THREE` from `FIVE` giving `RESULT`. Use `WTO` to display `RESULT`. Define other fields as necessary, but do all arithmetic in binary; that is, use packed fields only as required for the `ED` command.

A solution to this problem is shown here:

```
         START 0
         REGS
BINARY   BEGIN
         LH    R7,FIVE
         SH    R7,THREE
         CVD   R7,DBL
         ED    RESULT,DBL+6
         WTO   RESULT
         RETURN
DBL      DC    D'0'
FIVE     DC    H'5'
THREE    DC    H'3'
RESULT   DC    X'40202120'
         END   BINARY
```

Recall that the `.PRN` listing is required for any testing session. That listing is shown on the next page. We will run a test session, stopping before the `LH`, `SH`, `CVD`, `ED`, and `WTO` instructions to view register 7. Within `TEST`, the `r` command will display all sixteen registers. We also make use of the `l` command to *limit* the trace; that is, by specifying a limit of one, the trace will step through the program one instruction at a time.

The test session follows.

```
 BINARY                                            PAGE    1
PC/370 CROSS ASSEMBLER  OPTIONS=LXACE
   LOC                ADR1   ADR2 LINE LABEL    OP       OPERANDS
000000                               1          START    0
000000                               2 *+++++++ REGS
000000              00000000         3 R0       EQU      0
000000              00000001         4 R1       EQU      1
000000              00000002         5 R2       EQU      2
000000              00000003         6 R3       EQU      3
000000              00000004         7 R4       EQU      4
000000              00000005         8 R5       EQU      5
000000              00000006         9 R6       EQU      6
000000              00000007        10 R7       EQU      7
000000              00000008        11 R8       EQU      8
000000              00000009        12 R9       EQU      9
000000              0000000A        13 R10      EQU      10
000000              0000000B        14 R11      EQU      11
000000              0000000C        15 R12      EQU      12
000000              0000000D        16 R13      EQU      13
000000              0000000E        17 R14      EQU      14
000000              0000000F        18 R15      EQU      15
000000                              19 *+++++   BEGIN
000000                              20 BINARY   CSECT
000000                              21          USING    *,15
000000 47F0F058              0058   22          B        KZHQX002
000004 0B                           23          DC       AL1(11)
000005 C2C9D5C1D9E84040             24          DC       CL11'BINARY  '
000010 0000000000000000             25 HZQKX002 DC       18F'0'
000058 90ECD00C              000C   26 KZHQX002 STM      14,12,12(13)
00005C 50D0F014              0014   27          ST       13,HZQKX002+4
000060 18ED                         28          LR       14,13
000062 41D0F010              0010   29          LA       13,HZQKX002
000066 50D0E008              0008   30          ST       13,8(0,14)
00006A                              31          DROP     15
00006A                              32          USING    HZQKX002,13
00006A 4870D090              00A0   33          LH       R7,FIVE
00006E 4B70D092              00A2   34          SH       R7,THREE
000072 4E70D088              0098   35          CVD      R7,DBL
000076 DE03D094D08E   00A4   009E   36          ED       RESULT,DBL+6
00007C                              37 *+++++++ WTO   RESULT
00007C 4300D098              00A8   38          IC       0,RESULT+L'RESULT
000080 925BD098              00A8   39          MVI      RESULT+L'RESULT,C'$'
000084 4120D094              00A4   40          LA       2,RESULT
000088 0AD1                         41          SVC      209
00008A 4200D098              00A8   42          STC      0,RESULT+L'RESULT
00008E                              43 *+++++++ RETURN
00008E 58DD0004              0004   44          L        13,4(13)
000092 98ECD00C              000C   45          LM       14,12,12(13)
000096 07FE                         46          BR       14
000098 4000000000000000             47 DBL      DC       D'0'
0000A0 0005                         48 FIVE     DC       H'5'
0000A2 0003                         49 THREE    DC       H'3'
0000A4 40202120                     50 RESULT   DC       X'40202120'
000000                              51          END      BINARY
```

```
A:\>binary T
TRACE EP A=07AB ID=370   370 A=000200 OP=47F0F058

  (Copyright message appears here)

TYPE H FOR HELP
+a
ADDR STOP ON
A=26a
    00026A  4870D090 4B70D092 4E70D088 DE03D094  .......k+..h...m
T(A-ADDR, E-DATA =, OR N-DATA <>)= a
+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
TRACE EP A=1649 ID=LA    370 A=000262 OP=41D0F010
TRACE EP A=17D1 ID=ST    370 A=000266 OP=50D0E008
ADDR STOP
    00026A  4870D090 4B70D092 4E70D088 DE03D094  .......k+..h...m
TRACE EP A=1676 ID=LH    370 A=00026A OP=4870D090
+r
 R0-7 00000000 00000080 00000000 00000000 00000000 00000000 00000000  00000000
 R8-F 00000000 00000000 00000000 00000000 00000000 00000210 00000138 00000200
 PSW 070C00000000026A ILC=4 CC=0
    000266  50D0E008 4870D090 4B70D092 4E70D088  &.\........k+..h
+l
TRACE LIMIT= 1
+t
TRACE SET
LIMIT COUNT
TRACE EP A=177D ID=SH    370 A=00026E OP=4B70D092
+r
 R0-7 00000000 00000080 00000000 00000000 00000000 00000000 00000000  00000005
 R8-F 00000000 00000000 00000000 00000000 00000000 00000210 00000138 00000200
 PSW 070C00000000026E ILC=4 CC=0
    00026A  4870D090 4B70D092 4E70D088 DE03D094  .......k+..h...m
+t
TRACE SET
LIMIT COUNT
TRACE EP A=14EF ID=CVD   370 A=000272 OP=4E70D088
+r
 R0-7 00000000 00000080 00000000 00000000 00000000 00000000 00000000  00000002
 R8-F 00000000 00000000 00000000 00000000 00000000 00000210 00000138 00000200
 PSW 070C200000000272 ILC=4 CC=2
    00026E  4B70D092 4E70D088 DE03D094 D08E4300  ...k+..h...m....
+l
TRACE LIMIT= 999
+t
TRACE SET
TRACE EP A=2127 ID=ED    370 A=000276 OP=DE03D094D08E
TRACE EP A=161C ID=IC    370 A=00027C OP=4300D098
TRACE EP A=1A85 ID=MVI   370 A=000280 OP=925BD098
TRACE EP A=1649 ID=LA    370 A=000284 OP=4120D094
TRACE EP A=26A3 ID=SVC   370 A=000288 OP=0AD1
   2    (this 2 is from the WTO)
TRACE EP A=1807 ID=STC   370 A=00028A OP=4200D098
TRACE EP A=162D ID=L     370 A=00028E OP=58DD0004
TRACE EP A=19D5 ID=LM    370 A=000292 OP=98ECD00C
TRACE EP A=0B4C ID=BCR   370 A=000296 OP=07FE
TRACE EP A=26A3 ID=SVC   370 A=000102 OP=0A1B

A:\>
```

Summary of Binary Load Instructions

`L  Rx,FW`	Copy the value of the fullword `FW` to register `Rx`, maintaining sign integrity.
`LH Rx,HW`	Copy the value of the halfword `HW` to register `Rx`, maintaining sign integrity.
`LR Rx,Ry`	Copy the value of register `Ry` to register `Rx`. Register `Ry` remains unchanged.

Summary of Binary Compare Instructions

`C  Rx,FW`	Arithmetic compare of the contents of register `Rx` with the fullword `FW`
`CH Rx,HW`	Arithmetic compare of the contents of register `Rx` with the halfword `HW`
`CR Rx,Ry`	Arithmetic compare of the contents of registers `Rx` and `Ry`

Summary of Binary Store Instructions

`ST  Rx,FW`	Copies the value of register `Rx` to the fullword `FW`.
`STH Rx,HW`	Copies the value of register `Rx` to the halfword `HW`, truncating if necessary and without warning.

Summary of Binary Add Instructions

`A  Rx,FW`	Add the value of the fullword `FW` to the value in register `Rx`, with the sum in register `Rx`.
`AH Rx,HW`	Add the value of the halfword `HW` to the value in register `Rx`, with the sum in register `Rx`.
`AR Rx,Ry`	Add the value in register `Ry` to the value in register `Rx`, with the sum in register `Rx`.

Summary of Binary Subtraction Instructions

`S  Rx,FW`	Subtract the value of the fullword `FW` from the value in register `Rx`, with the difference in register `Rx`.
`SH Rx,HW`	Subtract the value of the halfword `HW` from the value in register `Rx`, with the difference in register `Rx`.
`SR Rx,Ry`	Subtract the value in register `Ry` from the value in register `Rx`, with the difference in register `Rx`.

Exercises

1. True or false. Given `H1` and `H2` are halfwords, `F1` and `F2` are fullwords, and `D1` and `D2` are doublewords...

   - T F a. If `F1` begins at `LOC=02010C` then the last byte of `F1` is at `LOC=02010F`.
   - T F b. If `H1` begins at `LOC=020100` and `D1` is defined immediately after `H1`, then `D1` begins at `LOC=020108`.
   - T F c. If the fields are defined in the following order - `D1`, `D2`, `F1`, `F2`, `H1`, `H2` - then collectively they occupy 32 bytes.
   - T F d. All doublewords are fullword aligned, but not all fullwords are doubleword aligned.
   - T F e. The correct sequence of instructions to move `H1` to `F1` is `LH`, `ST`.
   - T F f. The correct sequence of instructions to move `F1` to `H1` is `L`, `ST`.
   - T F g. The correct sequence of instructions to add `H1` to `H2` is `LH`, `LH`, `A`, `STH`.
   - T F h. The correct sequence of instructions to subtract `H1` from `F2` is `LH`, `S`, `STH`.
   - T F i. To subtract the value in `R3` from the value in `R4` use `S R4,R3`.
   - T F j. To move the value in `R3` to `R5` use `ST R5,R3`.
   - T F k. To move the number 5 to `R6` use `CVB R6,=P'5'`
   - T F l. The correct sequence of instructions to convert `H1` to a packed field of size `PL3` is `LH`, `CVD`, `ZAP`.
   - T F m. The correct sequence of instructions to move a packed field of size `PL5` to `F1` is `ZAP`, `CVB`, `ST`.

2. Given `F1`, `F2`, and `F3` are fullwords. Write the BAL code to:

   a. place the sum `(F1+F2) in F3`.
   b. place the difference `(F1-F2) in F3`.
   c. place the sum `(F1+5)` in `F3`.
   d. place the greater of `F1`, `F2`, and `F3` into register 4.
   e. place the sum `(F1+F2+F3)` into register 5.
   f. place the edited sum `(F1+F2+F3)` into `WK8`, using the edit mask `=X'4020202020202120'`.

3. Given `H1`, `H2`, and `H3` are halfwords. Write the BAL code to:

   a. place the sum `(H1+H2)` in `H3`.
   b. place the difference `(H1-H2)` in `H3`.
   c. place the sum `(H1+5)` in `H3`.
   d. place the greater of `H1`, `H2`, and `H3` into register 4.
   e. place the sum `(H1+H2+H3)` into register 5.
   f. place the edited sum `(H1+H2+H3)` into `WK8`, using the edit mask `=X'4020202020202120'`.

Exercises

4. Given `ZD` is a zoned decimal (unpacked) field defined as `CL3`, and `HW` is a halfword. Write the BAL code to:

   a. place the sum `(ZD+HW)` in register 4. Use binary addition.
   (You will need to convert `ZD` to binary.)
   b. place the sum `(ZD+HW)` in `PK3`, a packed field three bytes long.
   Use packed addition. (You will need to convert `HW` to packed.)

5. (Similar to Exercise 8 of Chapter 7) 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. Do all arithmetic in binary; that is, use packed fields only as required for the `ED` command. Show all field definitions.

6. (Similar to Exercise 11 of Chapter 7) 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. Use a register as the counter. Use packed fields only as required for the `ED` command. Your message should appear as follows:

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

7. Refer to `COGS9A.MLC` in chapter 9. This program produces the Sales Recap for Cogsworth Industries. Rewrite this program so that it reads `COGS.BIN` instead of `COGS.DAT`. Do all arithmetic in binary; that is, use packed fields only as required for the `ED` command.

8. Refer to the Small Town Hardware Store database in <u>More Datasets</u>. Write a program which will convert the numeric fields in the `TOOL` file to binary. All count fields should be stored as halfwords. All dollar-and-cent fields should be stored as fullwords. The output file should be left in EBCDIC form. Do not use `CR/LF`. Call your file `TOOL.BIN`. Note: You may need to reorder the fields within the record, or add unused fields (fillers), so as to guarantee that the fullwords and halfwords can, in fact, be appropriately aligned.

9. Write a program which will read the file produced in Exercise 8 above and create the report shown in Exercise 14(a) of Chapter 7. Although that program did not call for edited output, use the `ED` command to suppress leading zeroes. Do all arithmetic in binary; that is, use packed fields only as required for the `ED` command.

Exercises

10. (Similar to Exercise 14(b) of Chapter 7) Write a program which will read the file produced in Exercise 8 above and will create a new `TOOL` file 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 (which has the same record layout, including fullwords and halfwords). *All* tools should be written to this new file, even those for which there was no new order placed.