**CSCI 360 Packed Decimal Instructions**

Please note that these are often wrongly considered to be the most difficult of all Assembler instructions, even by advanced programmers. The misconception most likely comes down to the fact that the material was inadequately taught! There is a lot of detail so pay attention!

**Declaring packed decimal fields**

label DS PL3 AN UNITIALIZED 3-BYTE PACKED DECIMAL FIELD

label DC PL5'0' AN INITIALIZED 5-BYTE PACKED DECIMAL FIELD

In this class, please declare ALL packed decimal fields with DC and initialize it to 0 if nothing else!

**To convert from zoned to packed**

Format: label PACK D1(L1,B1),D2(L2,B2)

- Packs the L2 byte zoned decimal number at D2(B2)

and stores it as a L1 byte packed decimal number at D1(B1)

- The packing process:

1. The zone and numeric digit of the **rightmost** byte of

the zoned decimal number are reversed and placed in the

rightmost byte of the packed decimal number

2. The remaining numeric digits are moved to the packed decimal

number, proceeding from **right to left**. If the zoned

number has more digits than the packed number can hold, the

extras are ignored. If the zoned number has less digits

than the packed number, the remaining packed digits are

filled with zero

- The second operand does NOT have to be a valid zoned decimal

number

Suppose FLD1 is 4 packed bytes and FLD2 is 7 zoned bytes, with the

initial values:

FLD1 00 00 00 00

FLD2 F9 F8 F7 F6 F5 F4 F3

Execution of:

PACK FLD1(4),FLD2(7) will set FLD1 to 98 76 54 3F

PACK FLD1(3),FLD2(7) will set FLD1 to 76 54 3F 00

PACK FLD1(4),FLD2(5) will set FLD1 to 00 98 76 5F

To swap the hexadecimal digits of any byte, pack the byte into

itself:

PACK BYTE(1),BYTE(1) will simply reverse the zone and

numeric digit

**To convert from packed to zoned**

Format: label UNPK D1(L1,B1),D2(L2,B2)

- Unpacks the L2 byte packed decimal number at D2(B2)

and stores it as a L1 byte zoned decimal number at D1(B1)

- The unpacking process:

1. The zone and numeric digit of the rightmost byte of the packed

decimal number are reversed and placed in the rightmost byte

of the zoned decimal number

2. The remaining numeric digits are padded with a zone digit of F

and moved to the zoned decimal number. If the packed number

has more digits than the zoned number can hold, the extras are

ignored. If the packed number has less digits than the zoned

number, the remaining digits are filled with F0

- The second operand does NOT have to be a valid packed decimal

number

Suppose FLD1 is 5 zoned bytes and FLD2 is 3 packed bytes

FLD1 00 00 00 00 00

FLD2 12 34 5C

Execution of:

UNPK FLD1(5),FLD2(3) will set FLD1 to F1 F2 F3 F4 C5

UNPK FLD1(1),FLD2(3) will set FLD1 to C5 00 00 00 00

UNPK FLD1(5),FLD2(2) will set FLD1 to F0 F0 F1 F2 43

**Add Packed**

Format: label AP D1(L1,B1),D2(L2,B2)

- The L2 byte packed decimal number at D2(B2) is added to the L1

byte packed decimal number at D1(B1). The sum is stored

at D1(B1).

- If L1 bytes is not large enough to hold all of the non-zero

digits of the result, overflow will occur.

- If either field is not a valid packed decimal number, a data

exception (SOC 7) will occur.

- Sets the condition code

Code Meaning

0 Result is 0

1 Result is negative

2 Result is positive

3 Overflow

**Subtract Packed**

Format: label SP D1(L1,B1),D2(L2,B2)

- The L2 byte packed decimal number at D2(B2) is subtracted from

the L1 byte packed decimal number at D1(B1). The difference

is stored at D1(B1).

- If L1 bytes is not large enough to hold all of the non-zero

digits of the result, overflow will occur.

- If either field is not a valid packed decimal number, a data

exception (SOC 7) will occur.

- Sets the condition code

Code Meaning

0 Result is 0

1 Result is negative

2 Result is positive

3 Overflow

**Zero and Add Packed**

Format: label ZAP D1(L1,B1),D2(L2,B2)

- The L1 byte packed decimal field at D1(B1) is zeroed out

and the L2 byte packed decimal number at D2(B2) is added

to the L1 byte packed decimal number at D1(B1). The sum

is stored at D1(B1).

- If the second operand is not a valid packed decimal number, a

data exception (SOC 7) will occur.

- Sets the condition code

Code Meaning

0 Result is 0

1 Result is negative

2 Result is positive

3 Overflow

- Use this instruction to move packed numbers: ZAP FLD1(3),FLD2(2)

- Use this instruction to initialize a field: ZAP FLD1(4),=P‘0’

**Multiply Packed**

Format: label MP D1(L1,B1),D2(L2,B2)

- The L2 byte packed decimal number at D2(B2) and the L1 byte

packed decimal number at D1(B1) are multiplied together. The

product is stored at D1(B1).

- A specification exception (SOC 6) will occur if L2 > 8 or if

L2 >= L1.

- A data exception (SOC 7) will occur if the first L2 bytes

of the first operand are not all zeroes or if either field is not

a valid packed decimal number.

**Divide Packed**

Format: label DP D1(L1,B1),D2(L2,B2)

- The L2 byte packed decimal number at D2(B2) is divided into

the L1 byte packed decimal number at D1(B1). The quotient

AND remainder are stored at D1(B1).

- The length of the quotient is equal to (L1 - L2) bytes.

The quotient is stored at D1(B1) for (L1 - L2) bytes.

- The length of the remainder is L2 bytes. The remainder is

stored at D3(B1) where D3 = D1 + (L1 - L2).

- A specification exception (SOC 6) will occur if L2 > 8 or if

L2 >= L1.

- A data exception (SOC 7) will occur if either field is not a

valid packed decimal number.

- A decimal divide exception (S0C B) will occur if the quotient is

too large or if the second operand is zero

**Integer Division vs. floating Point Division**

Using the divide packed (DP) instruction as it is results in integer division. For example, when you divide a 10-byte field, the dividend, by a 3-byte, the divisor, the quotient of the integer division is in the left 7 bytes of the 10-byte field and the remainder is in the rightmost 3.

There IS a way to "fake" real number, or floating point, division, though. If you shift the number in the the dividend field using the SRP instruction ***the number of decimal places to which you want to round the answer plus 1*** to the left before the division, you will get a real number result. Then, to round it, you shift just the quotient part of the answer one back to the right with rounding using SRP. You then use the edit pattern to place the decimal place where you want it in the output answer.

See the extended example at the end of this document.

**Compare Packed**

Format: label CP D1(L1,B1),D2(L2,B2)

- A numeric comparison of the L2 byte packed decimal number

at D2(B2) and the L1 byte packed decimal number at D1(B1)

is performed and the condition code is set.

Code Meaning

0 Equality

1 Operand 1 is less than Operand 2

2 Operand 1 is greater than Operand 2

**Shift and Round Packed**

Format: label SRP D1(L,B1),D2(B2),i

- The L byte field at D1(B1) is shifted.

- The amount and direction of the shift is determined by D2(B2).

- i is used as the rounding factor. It is usually 0 or 5, but can

be between 0 and 9. On a RIGHT shift, i is added to the leftmost

digit shifted off the right. If the sum is greater than 9, the

result is rounded up by 1.

- A shift to the LEFT is equivalent to multiplying by a power of 10

Left Shift Format: SRP D1(L,B1),N,i

N is a decimal number from 1 to 31 which is the number

of positions to shift

If non-zero digits are lost on the shift, decimal overflow

occurs

SRP NUM(6),3,0 left shift by 3, which is equivalent to

multiplying by 103

Before execution: NUM 00 00 01 20 75 9C

After execution: NUM 00 12 07 59 00 0C

- A shift to the right is equivalent to dividing by a power of 10

Right Shift Format: SRP D1(L,B1),(64-N),i

N is a decimal number from 1 to 32 which is the number

of positions to shift

SRP NUM(6),(64-2),0 right shift by 2, which is

equivalent to dividing by 102

Before execution: NUM 00 00 01 20 75 9C

After execution: NUM 00 00 00 01 20 7C

SRP NUM(6),(64-2),5

Before execution: NUM 00 00 01 20 75 9C

After execution: NUM 00 00 00 01 20 8C

- Sets the condition code

Code Meaning

0 Result is 0

1 Result is negative

2 Result is positive

3 Overflow

**Formatting Packed Decimal Numbers for Printing**

EDit Instruction

Format: label ED D1(L,B1),D2(B2)

- D1(B1) is the address of an L byte field that initially

contains a hexadecimal **pattern**. After execution of the

instruction, this field will contain the formatted result.

- D2(B2) is the **source field** and is the address of one or more

contiguous packed decimal numbers to be formatted

- The pattern is made up of four types of characters

1. X'20' **digit selector**

Used to print one packed decimal digit

2. X'21' **significance starter**

Used to print one packed decimal digit and turns the

significance indicator on **after** this byte

3. X'22' **field separator**

Used in formatting multiple packed numbers in one ED

instruction

4. Anything else is a **message character**. Common punctuation

marks are found on page 35 of the yellow card.

- The **significance indicator** is used to indicate when leading

zeroes should start to be printed. Initially is off.

- The first byte of the pattern is called a **fill character**.

Leading zeroes or message characters that are to be suppressed

are replaced by this character.

- The pattern and packed decimal number are both processed from

left to right. The pattern is processed one BYTE at a time,

while the packed decimal number is processed one DIGIT at a time.

- Execution proceeds as follows:

- If the character from the pattern is a *digit selector*, a

packed digit is examined.

- If the significance indicator is off and the packed

digit is a zero, the character in the pattern is

replaced by the fill character

- If the significance indicator is off and the packed

digit is non-zero, the digit is converted to zoned

format and the result replaces the character in the

pattern. The significance indicator is turned on.

- If the significance indicator is on, the packed digit

is converted to zoned format and the result replaces

the character in the pattern.

- If the character from the pattern is a *significance starter*,

the result is the same as above except that the

significance indicator is always turned on AFTER the

character in the pattern is replaced

- If the character from the pattern is a *field separator*, it

is replaced by the fill character and the significance

indicator is turned off.

- If the character from the pattern is a *message character*:

- If the significance indicator is off, the character is

replaced by the fill character

- If the significance indicator is on, the message

character is left unchanged

- A data exception (SOC 7) will occur if the second operand is

not a valid packed decimal number

- Sets the condition code

Code Meaning

0 The inspected  character in the last field is 0

1 The inspected character in the last field < 0

2 The inspected character in the last field > 0

- Most ED instructions are preceded by a MVC that moves the

pattern to D1(B1)

ED example 1 - Zero Suppression

Source Field: NUM 00 12 3C

Pattern Field: 40 20 20 20 20 20 <= a space as fill character

MVC NUMOUT(6),=X'402020202020'

ED NUMOUT(6),NUM

Output: NUMOUT 40 40 40 F1 F2 F3

When displayed: \_\_\_123 where \_\_\_ is 3 spaces

ED example 2 - Zero Suppression

Source Field: NUM 00 00 0C

Pattern Field: 40 20 20 20 20 20 <= a space as fill character

MVC NUMOUT(6),=X'402020202020'

ED NUMOUT(6),NUM

Output: NUMOUT 40 40 40 40 40 40

When displayed: 6 spaces

ED example 3 - Zero Suppression with significance indicator

Source Field: NUM 00 00 0C

Pattern Field: 40 20 20 20 21 20 <= a space as fill character

MVC NUMOUT(6),=X'402020202120'

ED NUMOUT(6),NUM

Output: NUMOUT 40 40 40 40 40 F0

When displayed: \_\_\_\_\_0 where \_\_\_\_\_ is 5 spaces

ED example 4 - Commas

Source Field: NUM 00 32 90 7C

Pattern Field: 40 20 6B 20 20 20 6B 20 21 20

MVC NUMOUT(10),=X'40206B2020206B202120'

ED NUMOUT(10),NUM

Output: NUMOUT 40 40 40 40 F3 F2 6B F9 F0 F7

When displayed: \_\_\_\_32,907 where \_\_\_\_ is 4 spaces

ED example 5 - Decimal Point

Source Field: NUM 15 32 90 7C

Pattern Field: 40 20 20 6B 20 21 20 4B 20 20

MVC NUMOUT(10),=X'4020206B2021204B2020'

ED NUMOUT(10),NUM

Output: NUMOUT 40 F1 F5 6B F3 F2 F9 4B F0 F7

When displayed: \_15,329.07 where \_ is a space

ED example 6 - Printing after a number

Source Field: NUM 90 7B

Pattern Field: 40 20 21 20 60

MVC NUMOUT(5),=X'4020212060'

ED NUMOUT(5),NUM

Output: NUMOUT 40 F9 F0 F7 60

When displayed: \_907- where \_ is a space

ED example 7 - Printing after a number

Source Field: NUM 90 7F

Pattern Field: 40 20 21 20 60

MVC NUMOUT(5),=X'4020212060'

ED NUMOUT(5),NUM

Output: NUMOUT 40 F9 F0 F7 40

When displayed: \_907\_ where \_ are spaces

ED example 8 - Printing more than one number

Source Field: NUM 36 0F 46 5F

Pattern Field: 40 20 21 20 22 20 21 20

MVC NUMOUT(8),=X'4020212022202120'

ED NUMOUT(8),NUM

Output: NUMOUT 40 F3 F6 F0 40 F4 F6 F5

When displayed: \_360\_465 where \_ are spaces

**Edit and Mark Instruction**

Format: label EDMK D1(L,B1),D2(B2)

- Performs exactly like the ED instruction but also sets a pointer

to the first non-zero digit of an edited number.

- The address of the first non-zero digit is stored in the last 3

bytes of register 1 **ONLY** if a X'20' or X'21' was replaced

by a source digit before a X'21' is reached

Example 1: Floating dollar sign

Source Field: NUM 46 78 23 9C

Pattern Field: 40 20 20 6B 20 21 20 4B 20 20

MVC NUMOUT(10),=X'4020206B2021204B2020'

EDMK NUMOUT(10),NUM

BCTR R1,0

MVI 0(R1),C'$'

Output: NUMOUT 5B F4 F6 6B F7 F8 F2 4B F3 F9

When displayed: $46,782.39

Since register 1 may not be altered by the EDMK instruction, it is a good idea to point register 1 to where the first non-blank character will occur.

Example 2:

Source Field: NUM 00 00 12 0C

Pattern Field: 40 20 20 6B 20 21 20 4B 20 20

LA R1,NUMOUT+6 address of 1st non-zero digit you want to print

MVC NUMOUT(10),=X'4020206B2021204B2020'

EDMK NUMOUT(10),NUM

BCTR R1,0

MVI 0(R1),C'$'

When displayed: \_\_\_\_\_$1.20 where \_\_\_\_\_ is 5 spaces

Example 3:

Source Field: NUM 00 00 0C

Pattern Field: 40 20 21 20 4B 20 20

LA R1,NUMOUT+3 address of 1st non-zero digit you want to print

MVC NUMOUT(7),=X'402021204B2020'

EDMK NUMOUT(7),NUM

BCTR R1,0

MVI 0(R1),C'$'

When displayed: \_\_$0.00 where \_\_ is 2 spaces

**Convert to Binary**

Format: label CVB R,D(X,B)

- The packed decimal number at D(X,B) is converted to its binary

representation and stored in R

- D(X,B) is the address of an 8 byte field on a doubleword

boundary

- A specification exception (SOC 6) will occur if D(X,B) is not on

a doubleword boundary

- A data exception (SOC 7) will occur if the number at D(X,B) is

not a valid packed decimal number

- A fixed point divide exception (SOC 9) will occur if the number

at D(X,B) is too large to be represented in 32 bits

- This is the replacement for XDECI

Example 1:

DWORD is a double word whose contents are 00 00 00 00 00 00 01 0F

CVB R4,DWORD will place 0000000A into R4

Example 2:

Before CVB: XDECI R5,BUFFER getting a 6 digit stock number

With CVB: PACK TEMP(8),BUFFER(6)

CVB R5,TEMP

TEMP DS D to ensure a doubleword boundary

**Convert to Decimal**

Format: label CVD R,D(X,B)

- The binary number in R is converted to an 8 byte packed decimal

number and stored starting at D(X,B)

- D(X,B) must be on a doubleword boundary

- A specification exception (SOC 6) will occur if D(X,B) is not on

a doubleword boundary

- This is the replacement for XDECO

Example 1:

R7 contains FFFFFFFF

CVD R7,DWORD where DWORD DS D

will place 00 00 00 00 00 00 00 1D into DWORD

Example 2:

Before CVD: XDECO R5,NUMOUT where NUMOUT DS CL12

With CVD: CVD R5,TEMP

MVC NUMOUT(8),=X'4020202020202120'

ED NUMOUT(8),TEMP+4

TEMP DS D ensure a doubleword boundary

NUMOUT DS CL8

TEMP+4 is used so that the first 8 digits are

skipped because they will all be zeroes

**Example Packed Multiplication**

The following is an example of calculating interest:

**PACKED ACTUAL VALUES**

**Account Balance:** 03 12 34 44 8F 0312344.48

**Interest Rate:** x 00 27 5F x .0275   
 08 58 94 73 20 0F $8,589.47

^

If this packed decimal result is shifted and rounded four digits to the right, it yields the correct answer and can then be EDMK'd with the correct editing and two decimal

places.

Remember that the account balance from the input record must be packed into a 5-byte packed field (if it is zoned decimal or EBCDIC). It can then be EDMK'd directly into the print line from there. Then the 5-byte packed account balance can be ZAP'd into an 8-byte packed decimal field to prepare it for multiplication.

After the multiplication, the 8-byte field can be SRP'd four digits to the right with rounding and the answer will be in the rightmost five bytes of the 8-bytes packed decimal field, rounded to two decimal places!

**Extended Example Packed Division**

Numbers stored as packed decimal numbers are always stored as integers and have NO decimal point either stored or implied. Therefore, the programmer has to keep track of the decimal points so that the right results can be obtained when doing math with packed decimal numbers. This is usually most important when packed decimal numbers or the results of math need to be displayed or printed.

When an integer is divided by another, integer division occurs. The result is NOT like what results when two numbers are divided with a calculator. An integer quotient and an integer remainder are the results. The same thing happens when you simply divide one packed decimal number by another.

**For example:**

ZAP FIELD3(4),FIELD1(3) GET THE DIVIDEND INTO FIELD3

DP FIELD3(4),FIELD2(1) DIVIDE 45 BY

\* NOTE: AFTER THIS DIVIDE, THE QUOTIENT WILL BE IN THE FIRST 3 BYTES OF

\* FIELD3 AND THE REMAINDER – ALWAYS THE SAME SIZE AS THE DIVISOR – WILL BE IN

\* THE LAST BYTE OF FIELD3. HERE IS WHAT THE 4 BYTES OF FIELD3 LOOK LIKE

\* AFTER THE DIVISION: 00 01 0F 5F NOTE THAT THERE ARE 2 PACKED DECIMAL

\* NUMBERS STORED IN FIELD3. THE FIRST THREE ARE THE QUOTIENT OF 10 AND THE

\* LAST BYTE IS THE REMAINDER OF 5.

**and in storage:**

FIELD1 DC PL3'45' STORAGE: 00 04 5F

FIELD2 DC PL1'4' STORAGE: 4F

FIELD3 DC PL4'0' STORAGE: 00 00 00 0F

FIELD 3 is necessary to hold the result of the division. (Most commonly, just add the lengths of the field to be divided and the divisor.)

But, there is a way to turn this integer division into floating point division and get an answer similar to what would result using a calculator.

**For example:**

SRP FIELD2,2,0 FIRST, EVEN UP THE NUMBER OF IMPLIED DECIMAL

\* PLACES. BECAUSE FIELD1 HAS TWO IMPLIED,

\* FIELD2 SHOULD HAVE THE SAME BEFORE DOING ANY

\* THING ELSE

\*

ZAP FIELD3(11),FIELD1(8) COPY FIELD1 INTO FIELD3 TO PREPARE FOR

\* DIVISION

\*

SRP FIELD3(11),3,0 SHIFT THE NUMBER TO BE DIVIDED 3 DIGITS TO

\* THE LEFT TO ADD SOME "FAKE" DECIMAL PLACES.

\* THE RULE IS: SHIFT THE NUMBER OF DIGITS TO

\* ROUND TO PLUS 1 MORE.

\*

DP FIELD3(11),FIELD2(3) NOW DIVIDE

SRP FIELD3(8),64-1,5 SHIFT JUST THE QUOTIENT PART ONE TO THE

\* RIGHT WITH ROUNDING. THE RESULT IS NOW

\* ROUNDED TO TWO DECIMAL PLACES! JUST

\* REMEMBER THAT THE ANSWER DESIRED IS IN

\* THE FIRST 8 BYTES, THE QUOTIENT PART OF

\* FIELD3. IGNORE THE REMAINDER FIELD.

\*

**and in storage:**

FIELD1 DC PL8'345622.97' STORAGE: 00 00 00 03 45 62 29 7F

FIELD2 DC PL3'23' STORAGE: 00 02 3F

FIELD3 DC PL11'0' ONCE AGAIN, THE LENGTH OF THIS FIELD IS

\* EQUAL TO THE LENGTHS OF THE TWO FIELDS

\* INVOLVED IN THE DIVISION, FIELD1 AND FIELD2

**Pitfalls**

Before doing any of this, know the data and know it well!

For example, if the largest dollar amount ever to be divided is $99,999,999.99, a number which fits into a minimum number of 6 packed bytes. If the maximum number ever used to divide this by is $99.99, a number which fits into a minimum number of 3 packed bytes.

Do not let this trip you up, though! The programming might be tempted to think that the result will always fit into 6 bytes, right? But this is not true. To be a better Assembler programmer, it is important – and critical – to think in extremes: What if the largest dollar number is divided by the smallest divisor dollar amount? In other words, divide $99,999,999.99 by $0.01? The result would be $9,999,999,999.00 which would require 7 bytes minimum! AND! What if the result had to have more than just two implied decimal places? It could require even more than 7!

So, what is the moral of this story? ALWAYS KNOW THE DATA AND ANTICIPATE! This is what is commonly referred to as "defensive programming." Always take the largest amount ever to be divided and divide it by the smallest amount it will ever be divided by. Then, and only then, consider how many bytes are necessary before doing the divide pack!

**Example:**

ZAP FIELD3(10),FIELD1(6) PREPARE FOR DIVIDE

SRP FIELD3(10),3,0 SHIFT 3 TO ROUND RESULT TO 2

DP FIELD3(10),FIELD2(3) DIVIDE

SRP FIELD3(7),64-1,5 SHIFT 1 DIGIT RIGHT WITH ROUNDING

\* THE RESULT IS IN THE FIRST 7 BYTES OF FIELD3 AND ROUNDED TO AN IMPLIED 2

\* DECIMAL PLACES. IT IS READY TO BE EDMK'd INTO THE OUTPUT FIELD.

FIELD1 DC PL6'783452.75' STORAGE: 00 07 83 45 27 5F

FIELD2 DC PL3'3.98' STORAGE: 00 39 8F

FIELD3 DC PL10'0' FIELD INTO WHICH DIVISION WILL BE DONE