Doing Packed Decimal Arithmetic in Assembler

Steve Comstock
The Trainer's Friend, Inc.

steve@trainersfriend.com
http://www.trainersfriend.com
303-393-8716

August 27, 2009
SHARE Session 1214
Denver, Colorado

Section Preview

❒ **Packed Decimal**

- **Zoned Decimal Format**
- **Packed Decimal Format**
- **DC and DS for Zoned and Packed Type Data**
- **Packed Decimal Instruction Set: PACK, UNPK, ZAP, CP, AP, SP, MP, DP**
- **Arithmetic Concerns**
    - ✗ Signifcant Digits
    - ✗ Keeping Track of Decimal Points
- **Packed Decimal Arithmetic - An Example**
- **Rounding**
- **Summary**

Zoned Decimal

❒ **When you key in numeric characters, they are stored in memory in a format called "zoned decimal":**

- **Each numeric character, 'n', is stored in a byte with the hex digits 'Fn'**
- **For example, '1562' will be stored in memory as**

F1 F5 F6 F2

- **The left half of any byte (the 'F's above) is called the <u>zone</u> portion of the byte**
- **The right half is called the <u>numeric</u> portion**
- **So if the data in a field is all numeric characters we say the field is in zoned decimal format**

Zoned Decimal Data and Signs

- Traditionally, the sign of a zoned number (if any) is represented in the zone portion of the right most byte of the field

    - Hex digits 'A', 'C', 'E', and 'F' represent a positive sign (memory device: 'FACE')

    - Hex digits 'B' and 'D' indicate a negative number:

      F1 F5 F6 C2

      F1 F5 F6 D2

- This came about, historically, from the days of punched cards

    - To enter a signed number into a card, the keypuncher would key in the digits, backspace, and then overpunch a plus or a minus sign

    - The resulting hole combinations were mapped into the hex characters we now use today

Packed Decimal

- ❒ The CPU cannot perform arithmetic operations on zoned decimal data

    - ♦ There are no machine instructions to add, subtract, multiply, or divide zoned decimal fields

- ❒ Your program must convert zoned decimal data into one of the formats that the CPU has arithmetic instructions for:

    - ♦ Packed Decimal

    - ♦ Binary Integer

    - ♦ Floating Point

- ❒ At this point in the course, we discuss packed decimal data and instructions

    - ♦ Later we discuss binary integer data and instructions

    - ♦ We will not discuss floating point in this course

Packed Decimal Format

❒ Packed decimal data is a string of bytes such that each byte contains two digits except the right most hex digit, which represents the sign of the number:

DD DD DD DD DS

❒ Packed decimal fields may be any length from one byte to 16 bytes

- Thus the range of values is:

  \- 9,999,999,999,999,999,999,999,999,999,999

  to

  \+ 9,999,999,999,999,999,999,999,999,999,999

- Note that commas are not stored in the data, nor are decimal points

❒ Whenever the CPU executes a packed decimal instruction, it (the CPU) checks the operands are valid packed decimal format

- Each digit in the range 0-9

- The sign digit one of hex 'A' through 'F'

❒ If the data fails the test, a data exception occurs (program data interruption) that ultimately leads to an Abend

Packed Decimal Instructions

- So to perform arithmetic on numeric data, you must first convert the data into one of the valid formats and then use the appropriate instructions

- There are two instructions for converting data between zoned decimal format and packed decimal format:

PACK	—	convert from zoned to packed
UNPK	—	convert from packed to zone

- There is a small number of instructions available to work with packed decimal data; among them are:

AP	—	Add Packed
SP	—	Subtract Packed
MP	—	Multiply Packed
DP	—	Divide Packed
CP	—	Compare Packed
ZAP	—	Zero and Add Packed

- Packed decimal instructions are discussed in a separate chapter in the POO

Packed Decimal Instructions - Format

❒ All packed decimal instructions can work with the first and second operands being of different lengths!

- This requires the instructions to carry two length codes: one for each operand

- In explicit style, this shows up as:

$$xP \quad D_1(L_1,B_1),D_2(L_2,B_2)$$

- And in the machine format, the second byte (the length code for Storage / Storage instructions) uses the first half byte for the length of the first operand, and the second half byte for the length of the second operand:

op	L_1 L_2	B_1D	D D_1	B_2D	D D_2

- As before, the length codes are stored as one less than the actual length of the operands

❒ Packed decimal instructions belong to the class of instructions known as "Storage / Storage, two length codes"

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	________	____
	DC	Z'1239'	________	____
ANY	DC	Z'-92.45'	________	____
	DC	ZL2'8724'	________	____
XX	DC	ZL4'-34'	________	____

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'		
ANY	DC	Z'-92.45'		
	DC	ZL2'8724'		
XX	DC	ZL4'-34'		

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'	1 2 3 9	4
ANY	DC	Z'-92.45'		
	DC	ZL2'8724'		
XX	DC	ZL4'-34'		

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'	1 2 3 9	4
ANY	DC	Z'-92.45'	9 2 4 N	4
	DC	ZL2'8724'		
XX	DC	ZL4'-34'		

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'	1 2 3 9	4
			hex: F9 F2 F4 D5	
ANY	DC	Z'-92.45'	9 2 4 N	4
	DC	ZL2'8724'		
XX	DC	ZL4'-34'		

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'	1 2 3 9	4
			hex: F9 F2 F4 D5	
ANY	DC	Z'-92.45'	9 2 4 N	4
	DC	ZL2'8724'	2 4	2
XX	DC	ZL4'-34'		

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'	1 2 3 9	4
			hex: F9 F2 F4 D5	
ANY	DC	Z'-92.45'	9 2 4 N	4
	DC	ZL2'8724'	2 4	2
XX	DC	ZL4'-34'	0 0 3 M	4

DC For Zoned Decimal Type Data

Rules for DCs With Zoned Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with zoned zeros (X'F0'), if necessary
  - ✗ Truncate on left, if necessary
  - ✗ May only contain decimal characters (0-9) and possibly a leading plus or minus sign (which is stored as an overpunch in the rightmost byte)
  - ✗ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value	Length Attribute
FLD1	DC	ZL3'452'	4 5 2	3
	DC	Z'1239'	1 2 3 9	4
ANY	DC	Z'-92.45'	hex: F9 F2 F4 D5 9 2 4 N	4
	DC	ZL2'8724'	2 4	2
XX	DC	ZL4'-34'	hex: F0 F0 F3 D4 0 0 3 M	4

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only

- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	____________	____
	DC	P'1.3'	____________	____
XYZ	DC	PL3'456'	____________	____
EXTR1	DC	PL2'34345'	____________	____
	DC	P'-3'	____________	____
	DC	3P'13'	____________	____

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only

- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	01 3C	2
	DC	P'1.3'		
XYZ	DC	PL3'456'		
EXTR1	DC	PL2'34345'		
	DC	P'-3'		
	DC	3P'13'		

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only
- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	01 3C	2
	DC	P'1.3'	01 3C	2
XYZ	DC	PL3'456'		
EXTR1	DC	PL2'34345'		
	DC	P'-3'		
	DC	3P'13'		

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only

- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	01 3C	2
	DC	P'1.3'	01 3C	2
XYZ	DC	PL3'456'	00 45 6C	3
EXTR1	DC	PL2'34345'		
	DC	P'-3'		
	DC	3P'13'		

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only

- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	01 3C	2
	DC	P'1.3'	01 3C	2
XYZ	DC	PL3'456'	00 45 6C	3
EXTR1	DC	PL2'34345'	34 5C	2
	DC	P'-3'		
	DC	3P'13'		

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only

- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	01 3C	2
	DC	P'1.3'	01 3C	2
XYZ	DC	PL3'456'	00 45 6C	3
EXTR1	DC	PL2'34345'	34 5C	2
	DC	P'-3'	3D	1
	DC	3P'13'		

DC For Packed Decimal Type Data

Rules for DCs With Packed Decimal Type Data

- **Value right-justified in storage**
  - ✗ Pad on left with binary zeros, if necessary
  - ✗ Truncate on left, if necessary
  - ✗ Value may only contain decimal characters (0-9), possibly a decimal point, and possibly a leading plus or minus sign (assembled sign placed in the rightmost hex digit)
    - ➢ Decimal points considered documentation only

- **Maximum length attribute: 16**

Examples

Definition as coded			Assembled Value (Hex)	Length Attribute
NO1	DC	P'13'	01 3C	2
	DC	P'1.3'	01 3C	2
XYZ	DC	PL3'456'	00 45 6C	3
EXTR1	DC	PL2'34345'	34 5C	2
	DC	P'-3'	3D	1
	DC	3P'13'	01 3C 01 3C 01 3C	2

More on Zoned and Packed Data Types

❒ **You may code DS statements for Z and P type data also**

❒ **Coding a field with a data type of Z or P does not guarantee the data in the field will always be of that type**

- **If you code a DC, the Assembler will ensure data is initialized properly**

- **But other instructions, MVC for example, can overlay any field with data of any type**

- **Similarly, describing a field in a record as any data type does not cause conversion when data is read into that area**

  - ✗ GET always brings in data as it is really formatted on the tape or disk or other media

❒ **In practice, people seldom code the Z data type in programs**

- **Most often a field containing character digits is described as character type: C**

PACK

```
PACK  COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL:

```
PACK  COMPFL+2(4),ZNFLD+1(2)
```

COMPFL:

```
PACK  FLDX,FLDX
```

```
PACK  FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA:

> **Machine Instruction**
>
> **Storage / Storage**
> **Two length codes**

❐ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD:

F1	F0	F3	F6

COMPFL:


```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL:


```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA:

67

FLDA:

Machine Instruction

Storage / Storage
Two length codes

- ❒ **Operands in PACK instructions are not checked for validity**

PACK

```
         PACK   COMPFL,ZNFLD
```

ZNFLD:

F1	F0	F3	F6

COMPFL:

						F

```
         PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL:


```
         PACK   FLDX,FLDX
```

```
         PACK   FLDA(1),FLDA(1)
```

FLDA:

67

FLDA:

> **Machine Instruction**
>
> **Storage / Storage**
> **Two length codes**

- ❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | | | | | | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL: | | | | | | |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | | | | | 3 | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL: | | | | | | |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | | | | | 03 | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL: | | | | | | | |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

> **Machine Instruction**
>
> **Storage / Storage**
> **Two length codes**

❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD:

F1	F0	F3	F6

COMPFL:

			1	03	6F

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL:


```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA:

67

FLDA:

Machine Instruction

Storage / Storage
Two length codes

- ❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD:

F1	F0	F3	F6

COMPFL:

00	00	00	01	03	6F

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

COMPFL:


```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA:

67

FLDA:

Machine Instruction

Storage / Storage
Two length codes

- ❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | 00 | 00 | 00 | 01 | 03 | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

F0F3

COMPFL: | | | | | | |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | 00 | 00 | 00 | 01 | 03 | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

FOF3

COMPFL: | | | | | | F |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- ❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | 00 | 00 | 00 | 01 | 03 | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

F0F3

COMPFL: | | | | | | 3F |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD:

F1	F0	F3	F6

COMPFL:

00	00	00	01	03	6F

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

F0F3

COMPFL:

				0	3F

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA:

67

FLDA:

Machine Instruction

Storage / Storage
Two length codes

❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK    COMPFL,ZNFLD
```

ZNFLD:	F1	F0	F3	F6		
COMPFL:	00	00	00	01	03	6F

```
PACK    COMPFL+2(4),ZNFLD+1(2)
```

F0F3

COMPFL:						
	—	—	00	00	00	3F

```
PACK    FLDX,FLDX
```

```
PACK    FLDA(1),FLDA(1)
```

FLDA: 67

FLDA:

Machine Instruction

Storage / Storage
Two length codes

❒ **Operands in PACK instructions are not checked for validity**

PACK

```
PACK   COMPFL,ZNFLD
```

ZNFLD: | F1 | F0 | F3 | F6 |

COMPFL: | 00 | 00 | 00 | 01 | 03 | 6F |

```
PACK   COMPFL+2(4),ZNFLD+1(2)
```

F0F3

COMPFL: | — | — | 00 | 00 | 00 | 3F |

```
PACK   FLDX,FLDX
```

```
PACK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | 76 |

Machine Instruction

Storage / Storage
Two length codes

- **Operands in PACK instructions are not checked for validity**

UNPK

```
UNPK  ZNFLD,COMPU
```

COMPU:	06	10	2C

ZNFLD:						

```
UNPK  FLDA(1),FLDA(1)
```

FLDA:	67

FLDA:	

Machine Instruction

Storage / Storage
Two length codes

- ❐ **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK  ZNFLD,COMPU
```

COMPU: | 06 | 10 | 2C |

ZNFLD: | | | | | | | |

```
UNPK  FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

❒ **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK   ZNFLD,COMPU
```

COMPU: | 06 | 10 | 2C |

ZNFLD: | | | | | | C2 |

```
UNPK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK   ZNFLD,COMPU
```

COMPU: | 06 | 10 | 2C |

ZNFLD: | | | | | F0 | C2 |

```
UNPK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

> **Machine Instruction**
>
> **Storage / Storage**
> **Two length codes**

❒ **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK   ZNFLD,COMPU
```

COMPU: | 06 | 10 | 2C |

ZNFLD: | | | | F1 | F0 | C2 |

```
UNPK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK   ZNFLD,COMPU
```

COMPU: | 06 | 10 | 2C |

ZNFLD: | | | F6 | F1 | F0 | C2 |

```
UNPK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- ❒ **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK   ZNFLD,COMPU
```

COMPU:	06	10	2C			
ZNFLD:		F0	F6	F1	F0	C2

```
UNPK   FLDA(1),FLDA(1)
```

FLDA:	67
FLDA:	

Machine Instruction

Storage / Storage
Two length codes

❒ **Operands in UNPK instructions are not checked for validity**

UNPK

`UNPK  ZNFLD,COMPU`

COMPU: | 06 | 10 | 2C |

ZNFLD: | F0 | F0 | F6 | F1 | F0 | C2 |

`UNPK  FLDA(1),FLDA(1)`

FLDA: | 67 |

FLDA: | |

Machine Instruction

Storage / Storage
Two length codes

- ❒ **Operands in UNPK instructions are not checked for validity**

UNPK

```
UNPK   ZNFLD,COMPU
```

COMPU: | 06 | 10 | 2C |

ZNFLD: | F0 | F0 | F6 | F1 | F0 | C2 |

```
UNPK   FLDA(1),FLDA(1)
```

FLDA: | 67 |

FLDA: | 76 |

> **Machine Instruction**
>
> **Storage / Storage**
> **Two length codes**

- **Operands in UNPK instructions are not checked for validity**

ZAP

```
ZAP    FLD1,FLD2
```

Machine Instruction
Storage / Storage Two length codes

Before

FLD1: | 12 | 34 | 56 | 78 | 9F | FLD2 | 80 | 27 | 3C |

After

FLD1: | 00 | 00 | 80 | 27 | 3C |

❒ The condition code is set after the operation:

00	Result is zero
01	Result is negative
10	Result is positive
11	Overflow; high order digits have been lost

♦ May follow with something like these extended mnemonic branch instructions:

```
BZ     ZERO_VALUE
BM     NEGATIVE_VALUE
BP     POSITIVE_VALUE
BO     OVERFLOW
```

❒ A program data exception occurs if the <u>second operand</u> (only) is not correct packed decimal format

❒ This instruction is used to move packed decimal data to larger fields

♦ For example, to set up MP or DP correctly

CP

```
CP     FLDA,FLDB
```

Machine Instruction
Storage / Storage Two length codes

- The condition code is set after the compare operation:

00	Both operands are equal
01	First operand is low
10	First operand is high
11	-- does not apply to CP instruction --

  - Usually follow a 'CP' instruction with some conditional branch instruction ('BC' or 'BCR', or an extended mnemonic branch instruction)

  Note: this is a signed decimal compare:

If	FLDA:	| 01 | 0C |	and	FLDB:	| 02 | 0D |

Then	CP	FLDA,FLDB	will indicate FLDA > FLDB
But	CLC	FLDA,FLDB	will indicate FLDA < FLDB

- If the fields are of different lengths, the compare proceeds as if the shorter were extended on the left with zeros

- A program data exception occurs if both operands are not in the packed decimal format

AP, SP

```
+
AP      FLDA,FLDB

AP      FLDA,FLDA

–
SP      FLDA,FLDB

SP      FLDA,FLDA
```

Machine Instructions
Storage / Storage Two length codes

❒ **The condition code is set after either operation:**

00	Result is zero
01	Result is negative
10	Result is positive
11	Overflow; high order digits have been lost

♦ **May follow an 'AP' or 'SP' instruction with something like these extended mnemonic branch instructions:**

```
BZ      ZERO_RESULT
BM      NEGATIVE_RESULT
BP      POSITIVE_RESULT
BO      OVERFLOW
```

❒ **A program data exception occurs if both operands are not in the packed decimal format**

MP

```
MP      MPLCAND,MPLIER
```

Machine Instruction

Storage / Storage
Two length codes

Rules

- **Both operands must be valid packed decimal data (otherwise, a data exception)**
- **Second operand 8 bytes or less (otherwise, a specification exception)**
- **First operand longer than second operand (otherwise, a specification exception)**
- **First operand must have at least as many bytes of leading zeros as the total length of the second operand (otherwise, a data exception)**

Example:

MPLCAND: | 00 | 01 | 2C |

MPLIER: | 00 | 5C | **Will not work**

❒ **The condition code is not set after an MP operation**

DP

```
DP    DIVIDEND,DIVISOR
```

Machine Instruction
Storage / Storage Two length codes

<u>Rules</u>

- **Both operands must be valid packed decimal data (otherwise, a data exception)**
- **Second operand 8 bytes or less (otherwise, a specification exception)**
- **First operand longer than second operand (otherwise, a specification exception)**
- **Leftmost digit in dividend must be zero (otherwise, a decimal divide exception)**
- **If dividing by zero, a decimal divide exception occurs**
- **If not enough room for quotient, a decimal divide exception occurs; to test:**
  - ✗ align the left most digit of the divisor to the second digit of the dividend, padding the divisor to the left and right with zeros to match the length of the dividend
  - ✗ if the divisor, so aligned, is less than or equal to the dividend, the divide won't work (decimal exception)

DP, p.2.

Example

DIVIDEND: | 01 | 52 | 03 | 62 | 7C |

DIVISOR: | 00 | 34 | 5C |

doing the alignment test yields:

0 1 5 2 0 3 6 2 7

0 0 0 3 4 5 0 0 0

000345000 < 015203627 so divide won't work

But

- lop off the left most byte of the divisor, and, voilà:

0 1 5 2 0 3 6 2 7

0 3 4 5 0 0 0 0 0

034500000 > 015203627 therefore, OK!

in other words, in this case you could code:

```
DIVIDEND,DIVISOR+1(2)
```

DP, p.3.

```
DP    DIVIDEND,DIVISOR
```

Before the divide:

DIVIDEND:	dd	dd	dd	dd	ds		DIVISOR:	dd	ds

After the divide:

DIVIDEND:	qq	qq	qs	rr	rs		DIVISOR:	dd	ds
	- quotient -			- rem -				unchanged	

Notes

- **The length of the remainder equals the length of the divisor**
- **The quotient occupies the rest of the dividend field**
- **Be careful, after a divide, of code like:**

```
AP    FLDA,DIVIDEND
```

✗ The field DIVIDEND, in its entire length, no longer contains valid packed decimal data

✗ Use something like this:

```
AP    FLDA,DIVIDEND(3)
```

DP, p.4.

Notes, continued

- **The remainder is not additional decimal places, but rather an integer less than the divisor, for example:**

```
                         347          <— quotient
divisor —>       123 ) 42765          <— dividend
                       369
                        586
                        492
                         945
                         861
                          84          <— remainder
```

- **The condition code is not set after a DP instruction**

- **The test for a decimal divide is really trying to ensure there is enough room in the dividend field for the full quotient and the remainder**

  ✗ A reliable way to do this is to define the dividend field large enough to hold the greatest number of digits you actually expect in the dividend plus enough bytes for the remainder

  ➢ Since the size of the remainder is the length of the divisor, it is a good idea to keep the divisor defined as small as possible (yet large enough to hold the largest expected value)

  ✗ As you reach the limits (16 byte dividend or 8 byte divisor or numbers close together in length) you might have to programmatically test each situation

Arithmetic Concerns

❐ There are a number of issues the programmer needs to remember when working with arithmetic data in Assembler language programs

<u>Programmer responsibilities</u>

- ♦ Ensure data in the proper format for each instruction
- ♦ Plan enough room for significant digits in the result field
- ♦ Keep track of decimal point location

❐ The first point relates to knowing when to

- ♦ PACK data not already packed
- ♦ ZAP data to a larger field to satisfy MP and DP criteria
- ♦ Use a subset of a field to satisfy DP criteria

❐ We discuss significant digits and keeping track of the decimal point on the following pages ...

Significant Digits

❒ **To ensure arithmetic operations allow enough room in the result field, you must plan for the worst possible case (largest possible result)**

- **Do this by "playing computer" and keeping track of result digits, especially significant digits (non-leading-zero digits)**

- **As you plan your operations you may find you need to change the definitions of the work fields you are doing your calculations in (usually need to increase the size) or to ZAP data into larger fields before performing certain operations**

❒ **The result of AP and SP operations may contain at most one more significant digit than the longer of the two numbers**

e.g.: <u>DD DD DS</u> + <u>DD DS</u>

yields, at most, <u>0D DD DD DS</u>

- **Although repeated add or subtract instructions can create larger results (ultimately, you need to know your data)**

Significant Digits, 2

❒ **For MP operations, the largest possible number of significant digits in the product is the sum of the significant digits in the multiplier and the multiplicand**

e.g.: <u>DD DD DS</u> * <u>DD DS</u>

yields, at most, <u>0D DD DD DD DS</u>

- **However, recall the multiplicand must have at least as many bytes of leading zeros as total length of the multiplier, so we need to set up something like:**

e.g.: <u>00 00 DD DD DS</u> * <u>DD DS</u>

yields, at most, <u>0D DD DD DD DS</u>

❒ **For DP instructions, the largest possible number of significant digits in the quotient is**
(the number of significant digits in the dividend) minus
(the number of significant digits in the divisor) plus 1

e.g.: <u>DD DD DS</u> / <u>DD DS</u>

yields, at most, <u>DD DS</u>

- **Remember the special testing for divide exception also**

Keeping Track of Decimal Points

- Decimal points are only <u>assumed</u>

  - You are responsible for keeping track of where they belong and of making any necessary adjustments

- AP, SP, and CP instructions must have the same number of assumed decimal positions in both operands before performing the operation

  <u>For example, given</u>

```
FLDA:  00  DD  DD  DS        FLDB:  00  00  DS
              ^                               ^
```

  - Before we can add these numbers:

```
ZAP     FLDC,FLDB        so FLDC:  00  00  00  00  DS
                                                  ^

MP      FLDC,P1000       if P1000:             01  00  0C

              then       FLDC:     00  00  0D  00  0C
                                              ^
```

  - Then we can add the fields:

```
AP      FLDA,FLDC
```

Keeping Track of Decimal Points, 2

❒ **MP produces a product such that the number of assumed decimal places equals the sum of the decimal positions in the two operands**

For example, given

```
FLD1:  00  00  0D  DD  DS          FLD2:  DD  DS
                     ^                      ^
```

then

```
        MP      FLD1,FLD2
```

yields

```
        FLD1:  00  DD  DD  DD  DS
                         ^
```

Keeping Track of Decimal Points, 3

- ❒ **DP must always have the divisor adjusted to be treated as if it were an integer**

    - ♦ **That is, having no digits to the right of the decimal point**

- ❒ **The dividend must be adjusted in parallel with the divisor, logically shifting divisor and dividend decimal places in tandem:**

```
 DD DD                 DDD D
   ^                       ^
 -------   ------>    --------
   DD D                  DDD
      ^                      ^
```

- ♦ **<u>If the divisor has more significant decimal places</u> to the right of the decimal point than the dividend, you must multiply the dividend (number on top) by 10^n, where '*n*' is the number of decimal positions in the divisor (the number on the bottom) minus the number of decimal positions in the dividend**

    - ✗ The number of assumed decimal positions in the quotient will equal the number of decimal positions in the dividend after the adjustment

    - ✗ To carry extra decimal positions (for example, for rounding or for extra precision in the fractional part), multiply the dividend by 10 for each extra decimal position you want to carry (before you do the DP)

    - ✗ Also, any such multiplication may require the dividend field to be larger

Packed Decimal Arithmetic - An Example

- To put all this in perspective, let's look at a concrete example

- Given fields LM, LN, HI, HK, and JR coming in as zoned decimal numbers as follows

  - LM is an integer, up to three digits long
  - LN, HK, and JR contain five digits, including two digits to the right of the decimal point
  - HI is three digits long, all three being to the right of the decimal point
  - The definition for these fields, in an Assembler program might look like this:

```
LM         DS     CL3          999
LN         DS     CL5          999V99
HK         DS     CL5          999V99
JR         DS     CL5          999V99
HI         DS     CL3          V999
```

  - Code the instructions to calculate the value of

$$\frac{(LM + LN) * HI}{HK - JR}$$

  - Remember to check you are not dividing by zero (if you find you are, branch to a routine called ZERODIVIDE, which will be written later)

  - The result is to be placed into a field called FINAL_RESULT, in edited zoned decimal format for display on a terminal or in a report

Packed Decimal Arithmetic - An Example, 2

- **Since each of the input fields is in zoned format, the first step is to PACK each of the numbers**
  - **Where should we PACK them?**
    - Better define some work fields to hold the packed decimal version of these fields
    - What should we call these fields, and how big should they be?
      - Note that we can make a reasonable guess on the sizes of these fields; later, if we find we need to change the size, we can

- **On this page, code the instructions that define work fields to hold packed decimal versions of our numbers, and the PACK instructions themselves:**

```
         PACK

         PACK

         PACK

         PACK

         PACK
.
.
.
PACKLM   DS    PL
PACKLN   DS    PL
PACKHK   DS    PL
PACKJR   DS    PL
PACKHI   DS    PL
```

Packed Decimal Arithmetic - An Example, 2

- **Since each of the input fields is in zoned format, the first step is to PACK each of the numbers**
  - **Where should we PACK them?**
    - ✗ Better define some work fields to hold the packed decimal version of these fields
    - ✗ What should we call these fields, and how big should they be?
      - ➢ Note that we can make a reasonable guess on the sizes of these fields; later, if we find we need to change the size, we can
- **On this page, code the instructions that define work fields to hold packed decimal versions of our numbers, and the PACK instructions themselves:**

```
          PACK   PACKLM,LM      00   dd   d|s
          PACK
          PACK
          PACK
          PACK
.
.
.
PACKLM    DS     PL3
PACKLN    DS     PL
PACKHK    DS     PL
PACKJR    DS     PL
PACKHI    DS     PL
```

Packed Decimal Arithmetic - An Example, 2

❒ **Since each of the input fields is in zoned format, the first step is to PACK each of the numbers**

- **Where should we PACK them?**

  ✗ Better define some work fields to hold the packed decimal version of these fields

  ✗ What should we call these fields, and how big should they be?

  ➢ Note that we can make a reasonable guess on the sizes of these fields; later, if we find we need to change the size, we can

❒ **On this page, code the instructions that define work fields to hold packed decimal versions of our numbers, and the PACK instructions themselves:**

```
               PACK   PACKLM,LM     00   dd   d|s
               PACK   PACKLN,LN     dd   d|d  ds
               PACK
               PACK
               PACK
     .
     .
     .
PACKLM         DS     PL3
PACKLN         DS     PL3
PACKHK         DS     PL
PACKJR         DS     PL
PACKHI         DS     PL
```

Packed Decimal Arithmetic - An Example, 2

- **Since each of the input fields is in zoned format, the first step is to PACK each of the numbers**
    - **Where should we PACK them?**
        - ✗ Better define some work fields to hold the packed decimal version of these fields
        - ✗ What should we call these fields, and how big should they be?
            - ➢ Note that we can make a reasonable guess on the sizes of these fields; later, if we find we need to change the size, we can

- **On this page, code the instructions that define work fields to hold packed decimal versions of our numbers, and the PACK instructions themselves:**

```
                PACK  PACKLM,LM     00   dd   d|s
                PACK  PACKLN,LN     dd   d|d  ds
                PACK  PACKHK,HK     dd   d|d  ds
                PACK
                PACK
.
.
.
PACKLM          DS    PL3
PACKLN          DS    PL3
PACKHK          DS    PL3
PACKJR          DS    PL
PACKHI          DS    PL
```

Packed Decimal Arithmetic - An Example, 2

- **Since each of the input fields is in zoned format, the first step is to PACK each of the numbers**
  - **Where should we PACK them?**
    - Better define some work fields to hold the packed decimal version of these fields
    - What should we call these fields, and how big should they be?
      - Note that we can make a reasonable guess on the sizes of these fields; later, if we find we need to change the size, we can

- **On this page, code the instructions that define work fields to hold packed decimal versions of our numbers, and the PACK instructions themselves:**

```
         PACK  PACKLM,LM    00  dd  d|s
         PACK  PACKLN,LN    dd  d|d ds
         PACK  PACKHK,HK    dd  d|d ds
         PACK  PACKJR,JR    dd  d|d ds
         PACK
.
.
.
PACKLM   DS    PL3
PACKLN   DS    PL3
PACKHK   DS    PL3
PACKJR   DS    PL3
PACKHI   DS    PL
```

Packed Decimal Arithmetic - An Example, 2

❒ **Since each of the input fields is in zoned format, the first step is to PACK each of the numbers**

- **Where should we PACK them?**
  - ✗ Better define some work fields to hold the packed decimal version of these fields
  - ✗ What should we call these fields, and how big should they be?
    - ➢ Note that we can make a reasonable guess on the sizes of these fields; later, if we find we need to change the size, we can

❒ **On this page, code the instructions that define work fields to hold packed decimal versions of our numbers, and the PACK instructions themselves:**

```
          PACK  PACKLM,LM    00   dd   d|s
          PACK  PACKLN,LN    dd   d|d  ds
          PACK  PACKHK,HK    dd   d|d  ds
          PACK  PACKJR,JR    dd   d|d  ds
          PACK  PACKHI,HI   |dd   ds
.
.
.
PACKLM    DS    PL3
PACKLN    DS    PL3
PACKHK    DS    PL3
PACKJR    DS    PL3
PACKHI    DS    PL2
```

Packed Decimal Arithmetic - An Example, 3

- Now, what should we do next? We have two choices:
  - The add (LM + LN)
  - The subtract (HK - JR)
- Remember that we want to check that we don't divide by zero
  - It turns out it is more efficient to check for this before we do all the rest of the calculations only to find out we can't do the divide
- On this page, code the instructions to do the subtract and branch to ZERODIVIDE if the result is zero:

Packed Decimal Arithmetic - An Example, 3

❒ Now, what should we do next? We have two choices:

- ♦ The add (LM + LN)
- ♦ The subtract (HK - JR)

❒ Remember that we want to check that we don't divide by zero

- ♦ It turns out it is more efficient to check for this before we do all the rest of the calculations only to find out we can't do the divide

❒ On this page, code the instructions to do the subtract and branch to ZERODIVIDE if the result is zero:

```
SP     PACKHK,PACKJR       dd   d|d   ds
```

Packed Decimal Arithmetic - An Example, 3

- Now, what should we do next? We have two choices:
  - The add (LM + LN)
  - The subtract (HK - JR)
- Remember that we want to check that we don't divide by zero
  - It turns out it is more efficient to check for this before we do all the rest of the calculations only to find out we can't do the divide
- On this page, code the instructions to do the subtract and branch to ZERODIVIDE if the result is zero:

```
SP     PACKHK,PACKJR       dd  d|d  ds
BZ     ZERODIVIDE
```

Packed Decimal Arithmetic - An Example, 4

❒ **Now we can do the add: LM + LN**

- **Remember that both fields must have the same number of implied decimal places**

- **Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition**

❒ **On this page, write the code to add LM and LN:**

Packed Decimal Arithmetic - An Example, 4

❒ **Now we can do the add: LM + LN**

- **Remember that both fields must have the same number of implied decimal places**

- **Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition**

❒ **On this page, write the code to add LM and LN:**

```
MP    PACKLM,P100             dd  d|0  0s
```

Packed Decimal Arithmetic - An Example, 4

- Now we can do the add: LM + LN

  - Remember that both fields must have the same number of implied decimal places

  - Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition

- On this page, write the code to add LM and LN:

```
MP    PACKLM,P100           dd  d|0  0s
AP    PACKLM,PACKLN         dd  d|d  ds
```

Packed Decimal Arithmetic - An Example, 4

- Now we can do the add: LM + LN

  - Remember that both fields must have the same number of implied decimal places

  - Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition

- On this page, write the code to add LM and LN:

```
          MP    PACKLM,P100            dd  d|0  0s
          AP    PACKLM,PACKLN          dd  d|d  ds
```

but because of MP rules, need to go back and redefine PACKLM as PL4:

```
PACKLM   DS    PL4
P100     DC    P'100'
```

Packed Decimal Arithmetic - An Example, 4

- Now we can do the add: LM + LN

  - Remember that both fields must have the same number of implied decimal places

  - Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition

- On this page, write the code to add LM and LN:

```
          MP    PACKLM,P100            dd  d|0  0s
          AP    PACKLM,PACKLN          dd  d|d  ds
```

but because of MP rules, need to go back and redefine PACKLM as PL4:

```
PACKLM   DS    PL4
P100     DC    P'100'
```

then ...

```
          PACK  PACKLM,LM      gives  00  00  dd  d|s
```

Packed Decimal Arithmetic - An Example, 4

- Now we can do the add: LM + LN

  - Remember that both fields must have the same number of implied decimal places

  - Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition

- On this page, write the code to add LM and LN:

```
          MP    PACKLM,P100              dd  d|0  0s
          AP    PACKLM,PACKLN            dd  d|d  ds
```

but because of MP rules, need to go back and redefine PACKLM as PL4:

```
PACKLM  DS    PL4
P100    DC    P'100'
```

then ...

```
          PACK  PACKLM,LM      gives  00  00  dd   d|s
          MP    PACKLM,P100    gives  00  dd  d|0  0s
```

Packed Decimal Arithmetic - An Example, 4

- Now we can do the add: LM + LN

  - Remember that both fields must have the same number of implied decimal places

  - Since LM is '999' and LN is '999V99', you need to do some work prepatory to doing the addition

- On this page, write the code to add LM and LN:

```
          MP    PACKLM,P100             dd  d|0  0s
          AP    PACKLM,PACKLN           dd  d|d  ds
```

```
but because of MP rules, need to go back and redefine
PACKLM as PL4:
```

```
PACKLM   DS    PL4
P100     DC    P'100'
```

```
then ...

          PACK  PACKLM,LM      gives  00  00  dd   d|s
          MP    PACKLM,P100    gives  00  dd  d|0  0s
          AP    PACKLM,PACKLN  gives  0d  dd  d|d  ds
```

Packed Decimal Arithmetic - An Example, 5

- ❒ Now we're ready for the multiply: (LM + LN) * HI
  - ♦ Plan the size of the product field, and how many implied decimal places will there be?
  - ♦ Do you need to define a new work field, or can you use the field that currently holds the sum of LM + LN?
  - ♦ Remember to ensure you have enough leading zeros in the first operand
- ❒ On this page, code the instructions necessary to do the multiply:

Packed Decimal Arithmetic - An Example, 5

❒ **Now we're ready for the multiply: (LM + LN) * HI**

- **Plan the size of the product field, and how many implied decimal places will there be?**

- **Do you need to define a new work field, or can you use the field that currently holds the sum of LM + LN?**

- **Remember to ensure you have enough leading zeros in the first operand**

❒ **On this page, code the instructions necessary to do the multiply:**

To multiply PACKLM (which contains LM + LN) times PACKHI, PACKLM must have as many leading bytes of zeros as the length of PACKHI (which is two bytes long), so we must redefine PACKLM again:

```
PACKLM   DS    PL6
```

Packed Decimal Arithmetic - An Example, 5

- Now we're ready for the multiply: (LM + LN) * HI
  - Plan the size of the product field, and how many implied decimal places will there be?
  - Do you need to define a new work field, or can you use the field that currently holds the sum of LM + LN?
  - Remember to ensure you have enough leading zeros in the first operand
- On this page, code the instructions necessary to do the multiply:

To multiply PACKLM (which contains LM + LN) times PACKHI, PACKLM must have as many leading bytes of zeros as the length of PACKHI (which is two bytes long), so we must redefine PACKLM again:

```
PACKLM   DS    PL6
```

- so now our instructions and their results:

```
        PACK   PACKLM,LM       00   00   00   00   dd   d|s
```

Packed Decimal Arithmetic - An Example, 5

- Now we're ready for the multiply: (LM + LN) * HI

  - Plan the size of the product field, and how many implied decimal places will there be?

  - Do you need to define a new work field, or can you use the field that currently holds the sum of LM + LN?

  - Remember to ensure you have enough leading zeros in the first operand

- On this page, code the instructions necessary to do the multiply:

```
To multiply PACKLM (which contains LM + LN) times PACKHI,
PACKLM must have as many leading bytes of zeros as the
length of PACKHI (which is two bytes long), so we must
redefine PACKLM again:

PACKLM   DS    PL6

- so now our instructions and their results:


        PACK   PACKLM,LM       00   00   00   00   dd   d|s
        MP     PACKLM,P100     00   00   00   dd   d|0  0s
```

Packed Decimal Arithmetic - An Example, 5

- Now we're ready for the multiply: (LM + LN) * HI
  - Plan the size of the product field, and how many implied decimal places will there be?
  - Do you need to define a new work field, or can you use the field that currently holds the sum of LM + LN?
  - Remember to ensure you have enough leading zeros in the first operand

- On this page, code the instructions necessary to do the multiply:

```
To multiply PACKLM (which contains LM + LN) times PACKHI,
PACKLM must have as many leading bytes of zeros as the
length of PACKHI (which is two bytes long), so we must
redefine PACKLM again:

PACKLM   DS    PL6

- so now our instructions and their results:

        PACK   PACKLM,LM       00   00   00   00   dd   d|s
        MP     PACKLM,P100     00   00   00   dd   d|0  0s
        AP     PACKLM,PACKLN   00   00   0d   dd   d|d  ds
```

Packed Decimal Arithmetic - An Example, 5

❒ Now we're ready for the multiply: (LM + LN) * HI

- Plan the size of the product field, and how many implied decimal places will there be?

- Do you need to define a new work field, or can you use the field that currently holds the sum of LM + LN?

- Remember to ensure you have enough leading zeros in the first operand

❒ On this page, code the instructions necessary to do the multiply:

```
To multiply PACKLM (which contains LM + LN) times PACKHI,
PACKLM must have as many leading bytes of zeros as the
length of PACKHI (which is two bytes long), so we must
redefine PACKLM again:

PACKLM   DS    PL6

- so now our instructions and their results:


        PACK   PACKLM,LM        00   00   00   00   dd   d|s
        MP     PACKLM,P100      00   00   00   dd   d|0  0s
        AP     PACKLM,PACKLN    00   00   0d   dd   d|d  ds
        MP     PACKLM,PACKHI    00   dd   dd | dd   dd   ds
```

Packed Decimal Arithmetic - An Example, 6

❒ We're finally ready to do the divide:

$$\frac{(LM + LN) * HI}{(HK - JR)}$$

- Plan the result: how big is the remainder, how big is the quotient (that is, how many bytes long are these fields?)
- Remember to adjust when divisor has decimal places; how many decimal places will the quotient have?
- Remember to avoid dividing by zero

❒ On this page, write the instruction(s) necessary to accomplish the divide:

Packed Decimal Arithmetic - An Example, 6

- We're finally ready to do the divide:

$$\frac{(LM \quad + \quad LN) \quad * \quad HI}{(HK \quad - \quad JR)}$$

  - Plan the result: how big is the remainder, how big is the quotient (that is, how many bytes long are these fields?)
  - Remember to adjust when divisor has decimal places; how many decimal places will the quotient have?
  - Remember to avoid dividing by zero

- On this page, write the instruction(s) necessary to accomplish the divide:

First, in terms of our data fields, we are trying to divide PACKLM by PACKHK, where the contents of the fields look like this:

```
PACKLM:   00   dd   dd | dd   dd   ds
          ---------------------------
PACKHK:             dd   d|d   ds
```

we have to move both fields' implied decimal position to the right two places, so we are dividing by an integer; since PACKLM has more decimal places than PACKHK we don't need to multiply; but after this divide we see that there would be a remainder of 3 bytes and a quotient of 3 bytes; since the dividend value can have 9 digits (5 bytes) and the divisor is 3, re-size PACKLM:

```
PACKLM  DS   PL8
```

Packed Decimal Arithmetic - An Example, 6

- We're finally ready to do the divide:

$$\frac{(LM + LN) * HI}{(HK - JR)}$$

  - Plan the result: how big is the remainder, how big is the quotient (that is, how many bytes long are these fields?)
  - Remember to adjust when divisor has decimal places; how many decimal places will the quotient have?
  - Remember to avoid dividing by zero

- On this page, write the instruction(s) necessary to accomplish the divide:

```
Now the operands for our divide look like this:

PACKLM:   00  00  00  dd  dd  dd  d|d  ds
          -------------------------------
PACKHK:             dd    dd    ds


Then after the divide we will have this:


PACKLM:   00 qq  qq  q|q  qs  rr rr rs
          ----------------------------
          ---- quotient ---- | - rem -
```

Packed Decimal Arithmetic - An Example, 6

❒ We're finally ready to do the divide:

$$\frac{(LM \quad + \quad LN) \quad * \quad HI}{(HK \quad - \quad JR)}$$

- Plan the result: how big is the remainder, how big is the quotient (that is, how many bytes long are these fields?)
- Remember to adjust when divisor has decimal places; how many decimal places will the quotient have?
- Remember to avoid dividing by zero

❒ On this page, write the instruction(s) necessary to accomplish the divide:

```
So now we can safely do the divide:

        DP    PACKLM,PACKHK  00 qq qq q|q qs | rr rr rs
```

Packed Decimal Arithmetic - An Example, 6

❒ We're finally ready to do the divide:

$$\frac{(LM + LN) * HI}{(HK - JR)}$$

- Plan the result: how big is the remainder, how big is the quotient (that is, how many bytes long are these fields?)
- Remember to adjust when divisor has decimal places; how many decimal places will the quotient have?
- Remember to avoid dividing by zero

❒ On this page, write the instruction(s) necessary to accomplish the divide:

```
So then we have:

     PACK  PACKLM,LM        00 00 00 00 00 00 dd d|s
     PACK  PACKLN,LN        dd d|d ds
     PACK  PACKHK,HK        dd d|d ds
     PACK  PACKJR,JR        dd d|d ds
     PACK  PACKHI,HI       |dd ds
     SP    PACKHK,PACKJR    dd d|d ds
     BZ    ZERODIVIDE
     MP    PACKLM,P100      00 00 00 00 00 dd d|0 0s
     AP    PACKLM,PACKLN    00 00 00 00 0d dd d|d ds
     MP    PACKLM,PACKHI    00 00 00 dd dd | dd dd ds
     DP    PACKLM,PACKHK    00 qq qq q|q qs| rr rr rs
```

Packed Decimal Arithmetic - An Example, 7

- **Finally, we're ready to display the result by editing the quotient into the field FINAL_RESULT**
  - **Here we just demonstrate the necessary code and data areas**
    - ✗ Intricacies of the ED instruction and edit patterns is grist for a different session some other day

```
       MVC   FINAL_RESULT,EDPAT1
       ED    FINAL_RESULT,PACKLM+1

EDPAT1 DC    X'4020206B2021204B2020'
```

Packed Decimal Arithmetic - An Example, Summary

❒ Consolidate all the code from the example on this one page, for reference and discussion

```
*   Pack incoming fields
          PACK  PACKLM,LM         00 00 00 00 00 00 DD DS
          PACK  PACKLN,LN         DD DD DS  (2 dec pos)
          PACK  PACKHK,HK         DD DD DS  (2 dec pos)
          PACK  PACKJR,JR         DD DD DS  (2 dec pos)
          PACK  PACKHI,HI         DD DS     (3 dec pos)

*   PACKHK = (HK-JR), result has 2 dec pos
          SP    PACKHK,PACKJR
          BZ    ZERODIVIDE        don't divide by zero

*   adjust PACKLM to contain 2 implied decimal places
          MP    PACKLM,P100       00 00 00 00 00 DD D0 0S

*   PACKLM = (LM+LN), result still has 2 dec pos
          AP    PACKLM,PACKLN     00 00 00 00 0D DD DD DS

*   PACKLM = (LM+LN)*HI, 5 dec pos
          MP    PACKLM,PACKHI     00 00 00 DD DD DD DD DS

*   PACKLM =((LM+LN)*HI)/(HK-JR),  quot. has 3 dec pos
          DP    PACKLM,PACKHK     00 DD DD DD DS|DD DD DS
*                                 --quotient---- --rem---

          MVC   FINAL_RESULT,EDPAT1
          ED    FINAL_RESULT,PACKLM+1
*
.
.
.
PACKLM    DS    PL8
PACKLN    DS    PL3
PACKHK    DS    PL3
PACKJR    DS    PL3
PACKHI    DS    PL2
FINAL_RESULT DS CL10
EDPAT1    DC    X'4020206B2021204B2020'
P100      DC    P'100'
```

Rounding

- To "round a number to 'n' decimal places" means dropping all digits after the n^{th} decimal place, perhaps doing some adjustment in the n^{th} position

  - **Round up - if the truncated portion is not zero, add 1 to the n^{th} position**
    Round 215.73 up to units: —> 216

  - **Round down - truncate with no adjustment**
    Round 215.73 down to tenths: —> 215.7

  - **Half-adjust - add '5' to the $(n+1)^{st}$ position then truncate**

    - ✗ If the sum carries into the n^{th} position, you have automatically rounded up

    - ✗ If the sum doesn't carry into the n^{th} position, you have automatically rounded down

  - **Half-adjust 215.73 to units:**

$$\begin{array}{l} 215.73 \\ \underline{+\quad\ 5\ \ } \\ 216.23 \end{array} \qquad \textbf{therefore, value is 216}$$

  - **Half the time, half-adjusting results in rounding up; half the time it results in rounding down**

Rounding, 2

❒ **The effect is slightly different for negative numbers:**

- **Half-adjust -215.73 to units:**

```
-215.73
+     5
-------
-215.23      therefore, value is -215
```

- **Makes sense to add five to the absolute value of a number (or add -5 to a negative number and +5 to a positive one)**

❒ **Rounding only makes sense if the value being rounded has more significant positions than the rounding precision**

- **Therefore, in planning calculations where rounding is to take place, you need to allow for carrying enough positions (at least one more position than the precision of the rounding)**

- **For example, if rounding to hundredths, plan calculations to allow for results to thousandths**

Rounding - An Example

❒ **To see how one might deal with the need to round a calculation, recall our example**

- **Calculating**

$$\frac{(LM+LN) * HI}{(HK-JR)}$$

  ✗ produced a quotient in the eight byte field PACKLM

  ✗ the remainder was three bytes, the quotient five bytes, with three implied decimal positions:

```
PACKLM:      00  DD  DD  DD  DS  DD  DD  DS
                      ^
             ------ Quotient ------   --- Rem. ---
```

❒ **Suppose now the requirement is to half-adjust the result to two decimal places**

- **First, we need to add '5' to the last digit in the quotient:**

```
             AP     PACKLM(5),P5
```

- **where P5 is defined as:**

```
P5           DC     P'5'
```

Rounding - An Example, p.2.

❒ **If the quotient could be negative, we actually need to have some code like this:**

```
          CP    PACKLM(5),P0
          BL    SUB5
          AP    PACKLM(5),P5
          B     COMMON
SUB5      SP    PACKLM(5),P5
COMMON    ...
```

❒ **This certainly takes care of the first part of half-adjusting**

- **But the result is still five bytes with three implied decimal positions:**

```
PACKLM:     00 DD DD DD DS DD DD DS
                   ^
            ------ Quotient ------   --- Rem. ---
```

- **How to now get rid of the undesired third decimal position?**

- **There are three posibilities**

  ✗ Divide PACKLM by 10 (with all the complications of DP)

  ✗ Use the MVO instruction (an older, cumbersome style)

  ✗ Use the SRP instruction (described on the next page)

SRP - Shift and Round Packed

- The best option for solving our rounding problem is the SRP instruction

  Syntax

  ```
  SRP    FLDA,3,5
  ```

  - Shift the contents of FLDA left 3 decimal digits (no rounding)

  ```
  SRP    FLDA,61,5
  ```

  - Shift the contents of FLDA right 3 decimal digits after adding 5 to the last digit to be shifted out

  - If FLDA is not a valid packed decimal number, a data exception occurs

Machine Instruction
Storage / Storage one length code

... and some mysteries ...

SRP, continued

❐ **The explicit form of the SRP instruction is**

```
SRP      D1(L1,B1),D2(B2),I3
```

- **In machine format:**

F 0	**L_1 I_3**	**B_1---D_1---**	**B_2 ---D_2---**

- **Looking at a specific example:**

```
SRP      FLDA,3,5
```

F 0	**6 5**	**B_1 ---**	**D_1--**	**0 0**	**0 3**

L'FLDA-1

0 + 003

= 00 00 00 03_{16}

= 0000 0000 0000 0000 0000 0000 0000 0011

shift specification

<u>**Shift specification**</u>

- **Left shift: specify number of digits to shift (0 to 31)**
- **Right shift: specify 64-(number of digits to shift)**

SRP - Shift Specifications

Binary	Dec	Shift	Binary	Dec	Shift
000000	0	none	100000	32	right 32
000001	1	left 1	100001	33	right 31
000010	2	left 2	100010	34	right 30
000011	3	left 3	100011	35	right 29
000100	4	left 4	100100	36	right 28
000101	5	left 5	100101	37	right 27
000110	6	left 6	100110	38	right 26
000111	7	left 7	100111	39	right 25
001000	8	left 8	101000	40	right 24
001001	9	left 9	101001	41	right 23
001010	10	left 10	101010	42	right 22
001011	11	left 11	101011	43	right 21
001100	12	left 12	101100	44	right 20
001101	13	left 13	101101	45	right 19
001110	14	left 14	101110	46	right 18
001111	15	left 15	101111	47	right 17
010000	16	left 16	110000	48	right 16
010001	17	left 17	110001	49	right 15
010010	18	left 18	110010	50	right 14
010011	19	left 19	110011	51	right 13
010100	20	left 20	110100	52	right 12
010101	21	left 21	110101	53	right 11
010110	22	left 22	110110	54	right 10
010111	23	left 23	110111	55	right 9
011000	24	left 24	111000	56	right 8
011001	25	left 25	111001	57	right 7
011010	26	left 26	111010	58	right 6
011011	27	left 27	111011	59	right 5
011100	28	left 28	111100	60	right 4
011101	29	left 29	111101	61	right 3
011110	30	left 30	111110	62	right 2
011111	31	left 31	111111	63	right 1

❒ **Possible values in six bits as unsigned binary integer**

Rounding - An Example, p.3.

- Now, to handle the half adjust situation we just code this:

```
SRP    PACKLM(5),63,5
```

Summary

- ❐ Coding in Assembler is part understanding the instructions
    - ♦ And part knowing your data
    - ♦ And part knowing how to string together a series of instructions to accomplish a particular task
- ❐ In this session we have focused on the issues around doing packed decimal arithmetic calculations, including
    - ♦ Allowing enough room for sufficient precision
    - ♦ Keeping track of implied decimal points
    - ♦ Avoiding division by zero
    - ♦ Rounding the results of calculations
- ❐ The important thing here has been the process, so when you must do numeric calculations follow these guidelines as they apply to the problem you are faced with

Good luck. Don't panic!

Contact us for a copy of the Assembler program used to test lecture points.