

When A Comes After 1

Collating sequences, code pages and number formats: where a migrated sort goes quietly wrong, and how to stop it

Bill Ahlbrandt

President and CEO, Ahlbrandt Software

Abstract

A mainframe sort moved to Linux or Windows can end with return code zero, write exactly the right number of records, and still be wrong. The control statements describe fields by position and format. They don't say anything about the character set, the byte order or the number formats of the platform that wrote the data. This paper goes through, byte by byte, the things that make a migrated sort come out different: collating sequences, code pages, decimal signs, byte order, floating point, record descriptors and year windows. I computed every byte value and sort order shown here. There are two sound ways to handle it. You can keep the data in EBCDIC and sort it with an engine that reads it the way z/OS does, or you can convert it field by field from the record layout and change the sort to match. Converting whole files to ASCII isn't one of them.

What I found

- The same eleven keys sort into **four different orders** under ASCII, EBCDIC code page 037, EBCDIC code page 500 and a standard Linux locale. None of them gives you an error.
- **There's more than one EBCDIC.** Code pages 037 and 1047 put the square brackets, the caret and the not sign at different byte values, and they map different bytes to line feed.
- Convert a packed-decimal field as if it were text and you can get a **valid but wrong** number: -1,234 becomes +1,832, and nothing that reads it will complain.
- An unsigned zoned-decimal number converted to ASCII ends up with the zone X'3', and DFSORT's rules treat that as a **negative sign**. A clean-looking "1234" gets read as -1,234.
- Converting the whole file to ASCII is only safe when the file is all text. If a record mixes text with packed, binary or floating-point fields, you have two choices: a sort engine that understands EBCDIC, or field-level conversion driven by the copybook.

1. When a sort finishes and is still wrong

The title of this paper is the simplest case. On z/OS a capital A is stored as the byte X'C1' and the digit 1 as X'F1', so A sorts before 1. In ASCII, A is X'41' and 1 is X'31', so A sorts after 1. A statement like `SORT FIELDS=(1,10,CH,A)` is identical on both platforms. What it does isn't.

Most migration problems let you know about themselves. A job abends, or a count doesn't reconcile. Representation problems don't. A sort compares whatever bytes you hand it, and a numeric field that's been misread often still looks like a number. You get the same records in a different order, or with some totals changed. Downstream, a match-merge may drop or duplicate records and a control break may fire in the wrong place, and that can go on for months.

Another paper in this series covers how to prove a migrated sort gives identical output. This one is about why it so often doesn't. I've been working on sort for more than thirty years, and these are the failures that don't announce themselves.

2. EBCDIC order and ASCII order

DFSORT, IBM's sort for z/OS, says it uses “EBCDIC, the standard IBM collating sequence, or the ASCII collating sequence”, and that the collating sequence for character and binary data “is absolute”, meaning bytes are compared as unsigned values.¹ So format CH sorts by EBCDIC byte value, and the character classes come out in a different order on each platform:

- **EBCDIC:** space (X'40'), then most punctuation, then lowercase letters (X'81'–X'A9'), then uppercase (X'C1'–X'E9'), then digits (X'F0'–X'F9').
- **ASCII:** space (X'20'), punctuation mixed in among the other classes, then digits (X'30'–X'39'), then uppercase (X'41'–X'5A'), then lowercase (X'61'–X'7A').

Exhibit 1 runs eleven realistic keys through these rules.

Exhibit 1. Eleven keys, one sort statement, four orders

#	ASCII bytes (also Linux, C locale)	EBCDIC CP037 (and CP1047)	EBCDIC CP500	Linux sort, en_US.UTF-8
1	#4471	#4471	[TEMP]	100
2	100	smith	#4471	2ND AVE
3	2ND AVE	[TEMP]	smith	#4471
4	SMITH	Smith	Smith	smith
5	SMITH JR	SMITH	SMITH	Smith
6	SMITH- JR	SMITH JR	SMITH JR	SMITH
7	SMITH2	SMITH- JR	SMITH- JR	SMITH2
8	SMITHSON	SMITHSON	SMITHSON	SMITH JR
9	Smith	SMITH2	SMITH2	SMITH- JR
10	[TEMP]	100	100	SMITHSON
11	smith	2ND AVE	2ND AVE	[TEMP]

Computed by the author: Python 3.11 codecs cp037 and cp500, the ebcdic 2.0.1 package for cp1047, and GNU coreutils sort 9.4 on glibc 2.39. Each column is a plain ascending byte or locale sort of the same keys. CP037 and CP1047 give the same order for this list. They differ on the characters listed in section 3.²

The EBCDIC alphabet isn't contiguous either. A to I are X'C1'–X'C9', J to R are X'D1'–X'D9' and S to Z are X'E2'–X'E9'. In code page 037 there are 15 code points between A and Z that aren't letters, including the closing brace at X'D0' and the backslash at X'E0'.² So a range test written as “greater than or equal to A and less than or equal to Z” means something different in each encoding.

The fifth column of Exhibit 1 adds a trap that has nothing to do with the mainframe: a Linux sort run under an ordinary language locale. GNU sort warns that “the locale specified by the environment affects sort order” and recommends setting LC_ALL=C if you want byte-value order.³ Under en_US.UTF-8, spaces, hyphens and brackets are ignored at the first level of comparison and case only breaks ties, so SMITH2 sorts ahead of SMITH JR. The

PostgreSQL documentation says the same thing from the database side. Collation settings “affect the sort order of indexes”, which is why they’re fixed when a database is created.⁴

Numbers go from the bottom of the file to the top. Lowercase goes from the top to the bottom. Names with suffixes trade places depending on the locale. Each order is “correct” by its own rules, and the programs downstream will act differently on each.

3. Code pages: which EBCDIC?

EBCDIC comes in dozens of national and functional variants, each identified by an IBM coded character set identifier (CCSID). They agree on letters, digits, space and the common punctuation. They disagree on a lot of the rest.⁵ Four of them matter in most Western shops:

- **CCSID 037** (United States, Canada and others): the traditional default for z/OS batch data in North America. CCSID 1140 is the same code page with the euro sign at X'9F'.²
- **CCSID 1047** (Latin 1/Open Systems): registered with IANA as “EBCDIC Latin 1/Open Systems”,⁶ and the code page most z/OS utilities assume for untagged files in z/OS UNIX.⁷
- **CCSID 500** (International Latin-1) and national pages such as **273** (Germany and Austria), **285** (United Kingdom) and **297** (France), each with its own euro variant.

The differences land on exactly the characters programmers use as delimiters and markers. The left square bracket is X'BA' in CP037, X'AD' in CP1047, X'4A' in CP500 and X'63' in CP273. The caret is X'B0' in CP037 and X'5F' in CP1047, and in CP037 X'5F' is the not sign. The at sign is X'7C' in CP037 and X'B5' in CP273, where X'7C' is the section sign.² IBM's own DevOps guidance says files created in IBM-037 show their brackets wrong when read as IBM-1047, and that the byte values for [and] “must be changed from x'BA' and x'BB' to x'AD' and x'BD'”.⁷ IBM has a similar note for CCSIDs 037 and 500.⁸

That has two consequences. First, the conversion table is part of the data. Convert a file with the CP037 table and again with the CP1047 table and every record with a bracket or caret comes out different. Second, the line-ending characters are different. CP037 maps X'25' to line feed and X'15' to the next-line control, and CP1047 does it the other way around.² IBM lists both bytes among the characters that don't survive a round trip between EBCDIC and UTF-8.⁷ Section 4 shows why that matters for numeric data.

4. Numeric fields

Zoned decimal and the overpunched sign

A COBOL field declared `PIC S9(4)` with usage `DISPLAY` stores one digit per byte, each with the zone X'F'. The sign replaces the zone of the last byte. IBM's COBOL documentation gives the convention: X'C' if the number is positive or zero, X'D' if it's negative, and X'F' for unsigned fields.⁹ DFSORT is more forgiving when it reads them. For its ZD and PD formats, F, E, C, A, 8, 6, 4, 2 and 0 are positive, and D, B, 9, 7, 5, 3 and 1 are negative.¹⁰

Look at a signed last byte as a CP037 character and you see a letter or a brace. X'CO' to X'C9' show as { and A to I, and X'D0' to X'D9' as } and J to R. So +1,230 shows as “123{” and -1,234 as “123M”.² Those overpunch characters come from the code page. Under CP273 the same +1,230 shows as “123ä”. A character conversion turns them into ordinary ASCII letters and braces, and the numeric meaning is gone. The unsigned field is the sneakier case. “1234” converts cleanly to X'31 32 33 34', which looks right to every text tool. But the zone of the last byte is now 3, and under the DFSORT sign rules zone 3 is negative. Anything that applies those rules to the converted field will read it as -1,234.

Packed decimal

Packed decimal (COBOL COMP-3, DFSORT format PD) stores two digits per byte, with the sign in the last half-byte. Those bytes aren't characters, and there's no correct character translation for them. Exhibit 2 shows the worst case. Take -1,234, which is X'01 23 4D'. A CP037 table turns it into X'01 83 28', and that's a perfectly valid packed number: +1,832.² Nothing abends. Sums, range tests and sort order just change. There's a structural problem too. The value +254 is X'00 25 4C', and X'25' is line feed in CP037. A text-mode transfer that honors line ends will split the record at that byte.

Exhibit 2. What careless conversion does to numeric fields

Value and field	Fmt	Bytes on z/OS	After CP037 → ISO 8859-1	What a program then sees
-1,234 · PIC S9(4)	ZD	F1 F2 F3 D4	31 32 33 4D "123M"	Last digit X'D' isn't a digit, so it's invalid. Zone 4 now reads as positive.
+1,234 · PIC S9(4)	ZD	F1 F2 F3 C4	31 32 33 44 "123D"	Still +1,234 under DFSORT rules, by luck. Shows as 123D.
1,234 · PIC 9(4)	ZD	F1 F2 F3 F4	31 32 33 34 "1234"	Clean text, but zone 3 is a negative sign under DFSORT rules: -1,234.
-1,234 · PIC S9(4) COMP-3	PD	01 23 4D	01 83 28	Valid packed number: +1,832. No error.
+254 · PIC S9(4) COMP-3	PD	00 25 4C	00 0A 3C	X'25' became a line feed, so a text transfer splits the record.
-1,234 · PIC S9(8) COMP	FI	FF FF FB 2E	Transferred unconverted	Read as a native x86 integer: +788,267,007.
1.0 · long hex float	FL	41 10 00 00 00 00 00 00	Transferred unconverted	Read as IEEE 754 double: 262,144.0.

Computed by the author with Python 3.11 (cp037 codec, struct module). Sign rules as documented for DFSORT ZD and PD formats. The CP1047 table gives the same results for rows 1–4. Under CP1047 the byte that becomes a line feed is X'15' instead of X'25'.^{2, 10}

Binary and floating point

Binary fields (COMP, DFSORT formats FI and BI) only survive the transfer if they aren't converted, and even then byte order will catch you. z/Architecture stores integers big-endian, most significant byte first. x86 processors are little-endian.¹¹ A four-byte -1,234 is X'FF FF FB 2E', and native x86 code reads that as +788,267,007. The sort itself is fine if it compares the bytes the way DFSORT does. But any program or converter that loads the field into a native integer has to swap the bytes, and any file written natively on x86 and sorted as FI comes out in the wrong order. Stored little-endian, the values 1, 2, 256 and 65,536 sort as 65,536, 256, 1, 2.²

Floating point adds a format difference on top of byte order. DFSORT's FL format is hexadecimal floating point: a sign bit, a seven-bit exponent of 16 and a fraction.¹⁰ IBM mainframes have also had IEEE 754 binary floating point in hardware since the S/390 G5 of 1998,¹² so you may have both formats in the same shop. You can't swap one for the other. The value 1.0 is X'41 10 00 00 00 00 00 00' in long hexadecimal format and X'3F F0 00 00 00 00 00 00' in IEEE double precision. Read as IEEE, the hexadecimal bytes mean 262,144.²

5. Record descriptors, spanned records and mixed layouts

Fixed-length files (RECFM F and FB) have no structure beyond their length. Every record is LRECL bytes. Variable-length files (V and VB) put a four-byte record descriptor word (RDW) in front of each record. Its first two bytes hold the record length, including the RDW itself, as a big-endian binary halfword. Blocks have a similar block descriptor word.¹³

In spanned files (VS and VBS) a logical record can be split across blocks, with flags in the descriptor marking each segment as first, middle or last.¹⁴ DFSORT positions for variable-length records count the RDW, so the first data byte is position 5, and every control statement written for that file depends on it.

Off the mainframe, none of this happens by itself. IBM notes that a variable-blocked file sent by FTP in binary mode has its descriptors “stripped out unless you specify RDW”.¹⁵ Without them a binary file has no record boundaries at all. With them, the receiving side has to read a big-endian length prefix, which IBM warns has to travel in binary mode to avoid translation problems.¹⁶ Spanned records have to be rebuilt exactly as z/OS would before they can be sorted.

The mixed record is where whole-file conversion falls apart. A typical customer or transaction record has names as text, amounts as packed decimal, counters as binary and dates as zoned or packed fields. One conversion table across that record gets the text right and quietly corrupts the rest (Exhibit 2). Here’s a simple test. If its copybook has COMP, COMP-1, COMP-2, COMP-3 or COMP-5 items, or signed DISPLAY items, it isn’t a text file, whatever it looks like in an editor.

6. What the control statements assume

Every field in a sort statement names a format, and every format means a particular representation. CH is unsigned EBCDIC character, ZD and PD are signed zoned and packed decimal, FI is signed binary, BI unsigned binary, FL signed floating point, AQ character under an alternate collating sequence, and AC character collated in ASCII order.¹⁷ Numeric formats collate algebraically.¹ A literal such as C‘SMITH’ in an INCLUDE condition is compared as EBCDIC bytes. A replacement sort engine has to honor all of this on the bytes it’s given, or the statements have to be changed to match the converted data.

Three features change the order without changing any format. **ALTSEQ** defines an alternate collating sequence, either as an installation default or in a run-time control statement. **CHALT** extends it from AQ fields to CH fields, so what an unchanged CH statement means depends on an option that may be set outside the job.^{1,17} **LOCALE**, also an installation or run-time option, makes DFSORT use “the collating sequence defined in the active locale”.¹ Then there are the **Y2** formats (Y2C, Y2Z, Y2P, Y2D, Y2S and Y2B, then Y2T to Y2Y for full dates). They read two-digit years against a century window set by the Y2PAST option, which can be fixed or sliding.^{10,18} A sliding window moves with the system date, so the same key can collate differently from one year to the next. With an illustrative sliding window of 80 years, “46” means 1946 in 2026 and 2046 in 2027. Dual running across a year-end, or a test system with a different clock, will show it.

Exhibit 3. DFSORT format codes and their migration risks

Format	Represents	Typical failure after migration	What has to be kept
CH	Unsigned EBCDIC character	Converted data sorts in ASCII order; literals compared in the wrong code	EBCDIC byte order, or conversion of both data and literals
AQ, CHALT	Character in alternate sequence	ALTSEQ table in the installation options is not migrated	The exact installation and run-time ALTSEQ table
AC	EBCDIC data in ASCII order	Treated as CH, or applied to already-converted data	ASCII order computed from EBCDIC bytes
ZD, CLO, CTO, CSL, CST	Zoned decimal and sign variants	Overpunch converted to letters; zone 3 read as negative	Sign rules exactly as documented for each format
PD, PDO	Packed decimal	Character conversion creates valid but wrong values	Bytes untouched, or decoded from the copybook
FI, BI	Signed and unsigned binary	Byte order reversed by native code; corrupted by conversion	Big-endian comparison
FL	Hexadecimal floating point	Mixed up with IEEE 754; converted as text	Hexadecimal format and its ordering
Y2C ... Y2Y	Two-digit years	Different Y2PAST default or system date	Same window, same reference year
CH with LOCALE	Locale-collated character	Locale missing or defined differently on target	Same locale rules, or a deliberate decision to drop them

Format definitions from IBM DFSORT documentation. Risk assessments are the author's.^{1, 10, 17}

7. Your options, and a checklist

Keep the data in EBCDIC. Move the files in binary, with their record descriptors, and sort them with an engine that reads every format the way DFSORT does: EBCDIC byte order for CH, the documented sign rules for ZD and PD, big-endian comparison for FI and BI, hexadecimal floating point for FL, and the same ALTSEQ, LOCALE and Y2PAST settings. Character literals in the control statements have to be converted to EBCDIC before they're compared. The advantage is that the control statements don't change and you can compare the output byte for byte with the mainframe's. The cost is that every other program on the new platform that touches the data has to understand EBCDIC too.

Convert field by field. Use the copybook to drive a converter that translates text fields with a named code page, decodes packed, zoned and binary fields to a declared target format, and rebuilds the record lengths. Then go through every sort statement. CH keys will now collate in ASCII order, ZD and PD keys may not be ZD and PD anymore, and literals, ALTSEQ tables and range tests have to be rewritten. The advantage is that you end up with native data. The cost is a converter for each layout, a review of every sort, and output you can only compare with the mainframe's after converting it back. And where records redefine the same bytes more than one way, the copybook alone may not tell you which layout a given record uses.

Either one works if you apply it consistently. Mixing them is how shops end up with results that are right in test and wrong at month-end.

Exhibit 4. A pre-migration checklist for sort data

- 1 Inventory code pages.** Record the CCSID of every file and of the control statement library. Don't assume CP037.
- 2 Classify every file.** Text only, or mixed? Any COMP, COMP-3 or signed DISPLAY field makes it mixed.
- 3 Settle the transfer method.** Binary, with RDWs, for mixed files. Name the conversion table for text.
- 4 Capture installation options.** ALTSEQ table, CHALT, LOCALE, Y2PAST and any other defaults set outside the jobs.
- 5 Scan the control statements.** List every format code, literal and range test. Flag AQ, AC, FL, Y2x and LOCALE.
- 6 Build nasty test data.** Mixed case, leading digits, brackets and carets, negative and zero values, packed bytes of X'15' and X'25', maximum-length and spanned records.
- 7 Compare the bytes.** Check the output against the mainframe byte for byte, including record order, and run tests across a simulated year-end.

8. Bottom line

- **Sort order depends on the encoding.** The same keys and the same statement give different orders in EBCDIC, ASCII and a Linux locale.
- **The conversion table is part of the data.** Code pages differ on characters that matter, so name the table you use.
- **Numbers don't survive character conversion.** Zoned, packed, binary and floating-point fields change value, sign or record boundaries, a lot of the time without an error.
- **The control statements carry assumptions.** Format codes, ALTSEQ, LOCALE and Y2PAST encode how the mainframe read the data. Either the new engine honors them on EBCDIC bytes, or you convert the data and the statements together, field by field.

If you've run into a trap I haven't covered here, I'd like to hear about it.

About the author

Bill Ahlbrandt is President and CEO of Ahlbrandt Software, which he founded in 1993. Before that he was a systems programmer, Director of Information Security and Director of R&D at Acxiom. He has been working on mainframe sort and batch for more than thirty years, and his company's sort engine runs mainframe sort jobs on Linux and other platforms for banks, insurers and government agencies.

Full disclosure: my company sells sort software for platforms other than the mainframe, including an engine that sorts EBCDIC data in place. This paper describes both approaches and doesn't recommend a product.

References

1. IBM, *z/OS DFSORT Application Programming Guide*, V2R1, SC23-6878, "Control fields and collating sequences".
2. Author's computation, September 2026: Python 3.11.15 codecs cp037, cp273, cp500 and cp1140; ebcdic package 2.0.1 (cp1047); struct module; GNU coreutils sort 9.4 on glibc 2.39, locales C and en_US.UTF-8. Script available on request.
3. GNU coreutils 9.4, sort command help text.
4. PostgreSQL Global Development Group, *PostgreSQL Documentation*, current version, §23.1 "Locale Support".
5. IBM, character data representation architecture: coded character set identifier (CCSID) definitions for CCSIDs 37, 273, 285, 297, 500, 1047 and 1140.
6. IANA, character set registration "IBM1047", 27 September 2002, citing IBM Canada NLTC mapping of 10 November 1995.
7. IBM, *IBM Z DevOps Guide*, "Managing code page conversion", ibm.github.io.
8. IBM Support, "Conversion character differences between CCSID 037 and CCSID 500", modified 28 April 2025.
9. IBM, *Enterprise COBOL for z/OS 6.3 Programming Guide*, "NUMPROC".
10. IBM, *z/OS DFSORT Application Programming Guide*, V2R2, SC23-6878, "DFSORT data formats".
11. IBM, *z/Architecture Principles of Operation*, SA22-7832; Intel, *Intel 64 and IA-32 Architectures Software Developer's Manual*, Vol. 1.
12. E. M. Schwarz and C. A. Krygowski, "The S/390 G5 floating-point unit", *IBM Journal of Research and Development* 43(5/6), September 1999, pp. 707–721.
13. IBM, *z/OS DFSMS Using Data Sets*, SC23-6855, "Record descriptor word (RDW)".
14. Wikipedia, "Data set (IBM mainframe)", on RECFM=VBS segment flags; consulted September 2026.
15. IBM Support, "When C:D z/OS transfers a VB file there are 4 extra bytes at the beginning of the record that are not there when transferring the same VB file with FTP", 26 May 2020.
16. IBM, *z/OS Communications Server FTP documentation of the RDW option*, as quoted by J. McKown, IBM-MAIN mailing list, 10 January 2013.
17. IBM, *DFSORT Reference Summary*, Release 14, SX33-8001-14.
18. IBM, *DFSORT: Summary of Changes by Release* (Y2 formats and Y2PAST added with Release 13 and Release 14 PTFs).