Data Representation, Arithmetic Instructions, And Computer Efficiency

An Honors Thesis (ID 499)

By

Carol A. Nichols

Thesis Director

Ball State University

Muncie, Indiana

May, 1981

Note: Spring, 1981

TABLE OF CONTENTS

PREFACE

In order that those with limited knowledge of data processing can comprehend the following research and understand its impact, it is necessary to begin with explanations of the terms unfamiliar to the reader. The purpose of this research is to determine which representation of numeric data in the computer causes the arithmetic operations of addition, subtraction, multiplication, and division to be executed the most efficiently. Numeric data, or numbers, can be represented inside the computer in several different ways. In the COBOL language, there is a USAGE clause in which one can specify the type of representation to be used.

One possibility is to have a USAGE IS DISPLAY clause for a numeric data item which causes the internal representation to be the standard character form of the number. A clause of USAGE IS COMPUTATIONAL (or COMP) defines another type of representation. In COBOL, the COMP data type causes the number to be in its equivalent binary integer form. The binary number system is based on powers of two and uses only the digits 0 and 1. A number represented in COMP, or binary integer, does not have to be an integer however; it is possible to have a decimal represented as COMP data by using an implied decimal point. COMPUTATIONAL-3 (or COMP-3) data is the last of the data types to be considered in this study. A packed decimal number is the same as a COMP-3 data item. When a decimal number is "packed" into a packed decimal number, two digits are stored into one byte (a unit of computer storage) except for the last byte which contains one digit in the left half of the byte and a letter code in the right half indicating the sign of the number.

Data Representation, Arithmetic Instructions, And Computer Efficiency

In data processing, emphasis is placed upon writing programs that are efficient in the use of various computer resources, especially time. The desirability of reducing the amount of time taken to execute a program cannot be argued against; the old adage that "time is money" certainly applies to the computer world. It is often difficult, however, to find a way to cause a program to run faster. In Digital's TOPS-10/TOPS-20 COBOL-74 Language Manual, it is recommended that "programming standards should insist on using the correct data types for certain operations." Before establishing these programming standards it must be known what data type to use for what operation.

The research under consideration is designed to match up appropriate data types and the normal arithmetic operations of addition, subtraction, multiplication, and division; the data types used are character, binary integer, and packed decimal representations. A DEC-10 computer with TOPS-10 operating system and FORTRAN and COBOL languages were used in this investigation.

To begin this research it was decided to use FORTRAN programs to generate 200 random numbers to be used as input data. These one hundred pairs of numbers were added, subtracted, multiplied, and divided in a COBOL program, referred to as the test program.

It was determined that the test program should be executed twenty-five times for each of the three data representations in order to get statistically accurate results. (See Appendix A for copies of the programs mentioned.)

In order to obtain the time elapsed during each of the four operations for each of the data types, the test program was executed under the control of COBDDT, a dynamic debugging utility program. COBDDT has a histogram feature which sets up a table in which various statistics about program behavior are recorded. The only statistic interesting to this study was the CPU (Central Processing Unit) time taken for each of the COBOL paragraphs in which addition, subtraction, multiplication, or division was performed. (See Appendix C for a sample histogram table.)

Prior to obtaining aby histogram results, the translation of COBOL statements into machine language statements for each of the three data types was examined. (See Appendix D for machine language code for each case.) It was found that for performing arithmetic on two numbers in character form, it is necessary to generate five statements in machine language code. The first two instructions in this case cause the numbers to become packed decimal numbers. In the next instruction, the packed decimal equivalents of the DISPLAY numbers are added, subtracted, multiplied, and divided, depending on the operation specified. The last two instructions cause the result of the appropriate operation to be unpacked so that it is once again in character form and then it is stored in the proper memory cell.

When dealing with two binary integer (COMP) numbers, the numbers do not have to be packed since the computer is capable

of binary arithmetic; this results in fewer machine language instructions being generated. For arithmetic to take place with two binary numbers, only three machine language instructions are needed. The first instruction moves the first operand to one of the registers of the computer. Next, the correct operation is performed on the two numbers. Finally the result of the arithmetic, which is located in the register, is stored in the appropriate location in memory.

Performing packed decimal arithmetic requires only two machine language statements. Since both numbers are already in packed decimal form, it is not necessary to change their form, as with DISPLAY numbers, or to move one of the numbers to a register, as in binary arithmetic. The COMP-3 numbers have arithmetic performed on them in the first instruction. In the second machine language instruction, the result of the operation must be stored in the specified memory cell.

Because packed decimal arithmetic requires fewer machine language instructions, one can suppose that arithmetic using packed decimal numbers would be the fastest and most efficient to perform. However, research findings indicate that this is true in only one case.

For addition of two numbers, the lowest average time, in seconds, for one hundred additions was associated with COMP-3 (packed decimal) data; the average time was 0.33252. An average time of 0.33304 seconds for binary integer data was extremely close to the time for COMP-3 data and, therefore, the difference can be considered insignificant. For DISPLAY data, an average time of 0.35764 was slower than for the other types of data.

When subtracting two numbers, a binary integer representation provided the fastest average time of 0.41872 seconds for one hundred subtractions. An average time of 0.47136 seconds was recorded for the use of packed decimal data, the second fastest time. Once again, data in character form was the slowest of the three with an average time, in seconds, of 0.4924.

In the case of multiplication of one hundred pairs of numbers, COMP (binary integer) data again proved to have the quickest average time, in seconds, of 0.4338. DISPLAY data representation yielded the next fastest average time of 0.47656 seconds, bettering the average time of 0.48708 seconds when using COMP-3 data.

Division of two numbers produced the same ranking of data types as did subtraction. For COMP data, an average time of 0.3956 seconds for one hundred divisions was calculated, the lowest average time of the three representations. Packed decimal data was ranked next in terms of efficiency, with an average time of 0.4268. As expected, data in character form gave the slowest average time of 0.43076 seconds.

Overall, when comparing average times for each of the operations and data types, it can be observed that numbers represented in binary integer (COMP) form allow arithmetic operations to be performed the most efficiently and quickly. Because COMP data was shown to be faster than packed decimal data, as was originally expected to be the fastest, one may hypothesize as to why the results came out as they did. (See Appendix B for a summary of results.)

To justify the results of this study, one might assume that, in a DEC-10 computer, the loading to a register and storing from a register of a binary integer is performed faster than when simply moving a packed decimal number from one location in memory to another. Also, it might be that binary integer arithmetic in a DEC-10 computer takes place in less time than packed decimal arithmetic. These explanations are probably the most feasible ones as to why arithmetic carried out with COMP (binary integer) data representation is performed more quickly, for the most part, than either packed decimal (COMP-3) or character (DISPLAY) arithmetic. To prove these assumptions, however, requires more research into the operation of a DEC-10 computer, which is beyond the scope of this research.

In conclusion, this research has found that for COBOL programs with numerous arithmetic calculation, binary integer or COMP representation should be used to reduce the amount of time taken for execution of the program. The reader should keep in mind that this research and its findings are very much machine and language dependent. Similar results may or may not be obtained when executing the same or similar research programs on other computers and/or in other languages.

```
00100   IDENTIFICATION DIVISION.
00200   PROGRAM-ID.              DATAREP.
00300   AUTHOR.                  CAROL A NICHOLS.
00400
00500   ENVIRONMENT DIVISION.
00600   CONFIGURATION SECTION.
00700   SOURCE-COMPUTER.         DECSYSTEM-10.
00800   OBJECT-COMPUTER.         DECSYSTEM-10.
00900
01000   INPUT-OUTPUT SECTION.
01100   FILE-CONTROL.
01200           SELECT INPUT-FILE ASSIGN TO DSK.
01300
01400   DATA DIVISION.
01500   FILE SECTION.
01600   FD      INPUT-FILE
01700           RECORDING MODE IS ASCII
01800           LABEL RECORDS ARE STANDARD
01900           VALUE OF ID IS 'DATREPDAT'
02000           RECORD CONTAINS 20 CHARACTERS
02100           DATA RECORD IS INPUT-REC.
02200   01  INPUT-REC                      PIC X(20).
02300
02400   WORKING-STORAGE SECTION.
02500   77  EOF-SWITCH                     PIC X(3).
02600           88  EOF                            VALUE 'YES'.
02700   77  WS-RESULT                      PIC 9(12)V9(4)  USAGE IS DISPLAY,COMP,COMP-3.
02800
02900   01  IN-REC.
03000           05  IN-ONE                 PIC 9(6)V9(4)   USAGE IS DISPLAY,COMP,COMP-3.
03100           05  IN-TWO                 PIC 9(6)V9(4)   USAGE IS DISPLAY,COMP,COMP-3.
03200
03300   01  IN-RECORD.
03400           05  INR-ONE                PIC 9(6)V9(4).
03500           05  INR-TWO                PIC 9(6)V9(4).
03600
03700   PROCEDURE DIVISION.
03800
03900   MAIN-RTN.
04000           OPEN INPUT      INPUT-FILE.
04100           MOVE 'NO' TO EOF-SWITCH.
04200           PERFORM READ-RTN THRU READ-RTN-EXIT.
04300           PERFORM CONTROL-RTN THRU CONTROL-RTN-EXIT
04400                   UNTIL EOF.
04500           CLOSE   INPUT-FILE.
04600           STOP RUN.
04700
```

Note: The marked USAGE clauses are changed according to the data type under investigation.

Test Program For Gathering Histogram Data
(continued)

```
04700
04800 CONTROL-RTN.
04900         PERFORM ADD-RTN THRU ADD-RTN-EXIT.
05000         PERFORM SUBTRACT-RTN THRU SUBTRACT-RTN-EXIT.
05100         PERFORM MULTIPLY-RTN THRU MULTIPLY-RTN-EXIT.
05200         PERFORM DIVIDE-RTN THRU DIVIDE-RTN-EXIT.
05300         PERFORM READ-RTN THRU READ-RTN-EXIT.
05400 CONTROL-RTN-EXIT.
05500         EXIT.
05600
05700 ADD-RTN.
05800         ADD IN-ONE IN-TWO GIVING WS-RESULT.
05900 ADD-RTN-EXIT.
06000         EXIT.
06100
06200 SUBTRACT-RTN.
06300         SUBTRACT IN-ONE FROM IN-TWO GIVING WS-RESULT.
06400 SUBTRACT-RTN-EXIT.
06500         EXIT.
06600
06700 MULTIPLY-RTN.
06800         MULTIPLY IN-ONE BY IN-TWO GIVING WS-RESULT.
06900 MULTIPLY-RTN-EXIT.
07000         EXIT.
07100
07200 DIVIDE-RTN.
07300         DIVIDE IN-ONE BY IN-TWO GIVING WS-RESULT.
07400 DIVIDE-RTN-EXIT.
07500         EXIT.
07600
07700 READ-RTN.
07800         READ INPUT-FILE INTO IN-RECORD
07900                 AT END MOVE 'YES' TO EOF-SWITCH.
08000         MOVE INR-ONE TO IN-ONE.
08100         MOVE INR-TWO TO IN-TWO.
08200 READ-RTN-EXIT.
08300         EXIT.
08400
```

```
00100           SUBROUTINE RANDOM (X)
00200           INTEGER  MOD
00300           REAL FLOAT
00400           INTEGER A, MULT, BASE
00500           DATA A/19727/, MULT/25211/, BASE/32759/
00600           A = MOD(MULT*A, 32768)
00700           X = FLOAT(A) / FLOAT(BASE)
00800           RETURN
00900           END
```

GENDAT.FOR

```
00100           INTEGER I,N,IN1,IN2
00200           REAL RANDOM,N1,N2
00300
00400           DO 100 I = 1,100
00500                   CALL RANDOM(N1)
00600                   CALL RANDOM(N2)
00700                   IN1 = 10**9 * N1
00800                   IN2 = 10**9 * N2
00900                   WRITE (1,5) IN1,IN2
01000   5               FORMAT (I10,I10)
01100   100     CONTINUE
01200           READ (5,10,END = 25) N
01300   10      FORMAT(I2)
01400   25      END FILE1
01500           STOP
01600           END
```

Note: The subroutine RANDOM generates random numbers in the range of 0 to 0.999999999. The program GENDAT takes these numbers and converts them from decimal to whole number form.

DATA FOR DISPLAY
(Character)

	Addition	Subtraction	Multiplication	Division
1	.359	.482	.506	.499
2	.436	.473	.480	.502
3	.363	.637	.546	.429
4	.336	.473	.452	.428
5	.37[illegible]	.502	.496	.420
6	.33[illegible]	.471	.451	.429
7	.3[illegible]	.474	.452	.427
8	.3[illegible]	.501	.450	.428
9	.3[illegible]	.503	.453	.419
10	.3[illegible]	.504	.496	.421
11	.3[illegible]	.500	.495	.420
12	.336	.473	.497	.418
13	.373	.470	.452	.422
14	.371	.476	.496	.428
15	.333	.502	.497	.429
16	.33[illegible]	.501	.495	.430
17	.33[illegible]	.501	.452	.426
18	.371	.503	.453	.427
19	.336	.504	.451	.420
20	.3[illegible]	.502	.452	.428
21	.335	.471	.496	.421
22	.373	.471	.450	.426
23	.371	.472	.454	.420
24	.373	.470	.498	.428
25	.338	.474	.494	.424
Sum	8.941	12.31	11.914	10.769
$\bar{x}$	0.35764	0.4724	0.47656	0.43076

Appendix B

DATA FOR COMPUTATIONAL
(Binary Integer)

	Addition	Subtraction	Multiplication	Division
1	.311	.385	.475	.413
2	.254	.413	.486	.371
3	.258	.457	.473	.421
4	.293	.430	.440	.396
5	.359	.464	.516	.477
6	.359	.358	.470	.369
7	.308	.471	.484	.401
8	.371	.431	.471	.363
9	.308	.449	.369	.409
10	.327	.370	.462	.348
11	.360	.477	.473	.443
12	.343	.397	.345	.404
13	.478	.452	.509	.411
14	.337	.332	.385	.371
15	.377	.391	.337	.311
16	.313	.397	.434	.442
17	.326	.406	.360	.321
18	.400	.465	.449	.438
19	.429	.364	.415	.444
20	.328	.439	.480	.339
21	.453	.514	.437	.398
22	.353	.392	.320	.304
23	.338	.358	.400	.449
24	.360	.415	.437	.420
25	.314	.441	.418	.427
Sum	8.326	10.468	10.845	9.89
$\bar{x}$	0.33304	0.41872	0.4338	0.3956

Appendix B

DATA FOR COMPUTATIONAL-3
(Packed Decimal)

	Addition	Subtraction	Multiplication	Division
1	.333	.476	.543	.353
2	.321	.448	.472	.309
3	.253	.465	.471	.431
4	.271	.441	.497	.453
5	.334	.522	.501	.417
6	.366	.466	.557	.477
7	.357	.471	.505	.477
8	.299	.451	.488	.441
9	.317	.497	.502	.376
10	.321	.502	.558	.406
11	.353	.352	.414	.384
12	.324	.503	.458	.428
13	.272	.413	.462	.421
14	.299	.551	.460	.506
15	.256	.486	.489	.386
16	.311	.372	.530	.490
17	.380	.463	.481	.384
18	.293	.430	.481	.437
19	.385	.540	.445	.424
20	.369	.517	.488	.401
21	.333	.489	.455	.466
22	.243	.437	.513	.466
23	.335	.533	.470	.436
24	.331	.527	.516	.418
25	.403	.432	.421	.483
Sum	8.313	11.784	12.177	10.67
$\bar{x}$	0.33252	0.47136	0.48708	0.4268

COBDDT ISTOGRAM FOR DATARE REPORT: 1

PROCEDURE	ENTRIES	CPU	ELAPSED
-GENERATED-SECTION-NAME-	0	5.562	2:05.964
MAIN-RTN	1	0.031	0.485
CONTROL-RTN	100	0.636	28.863
CONTROL-RTN-EXIT	100	0.519	16.549
ADD-RTN	100	0.321	2.353
ADD-RTN-EXIT	100	0.342	5.563
SUBTRACT-RTN	100	0.448	18.236
SUBTRACT-RTN-EXIT	100	0.541	5.984
MULTIPLY-RTN	100	0.472	11.379
MULTIPLY-RTN-EXIT	100	0.570	3.650
DIVIDE-RTN	100	0.309	2.238
DIVIDE-RTN-EXIT	100	0.525	12.445
READ-RTN	101	0.385	5.554
READ-RTN-EXIT	101	0.463	12.665

OVERHEAD: ELAPSED: 28.549 CPU: 0.231

Histogram Table

Note: This table was for the Test Program's second run using COMP-3 data. The underlined items are the items used for this research.

Machine Language Code

For DISPLAY data:

PACK _____,_____	Packs IN-ONE into temporary storage
PACK _____,_____	Packs IN-TWO into temporsry storage
AP _____,_____	Adds IN-ONE and IN-TWO, result is in temporary storage
UNPK _____,_____	Unpacks result into IN-TWO
MVC _____,_____	Moves result from IN-TWO to WS-RESULT

For COMP data:

LR _____,_____	Moves IN-ONE into a register
A _____,_____	Adds IN-ONE and IN-TWO, result is in the register
ST _____,_____	Stores result from register to WS-RESULT

For COMP-3 data:

AP _____,_____	Adds IN-ONE and IN-TWO, result is in IN-TWO
MVC _____,_____	Moves result from IN-TWO to WS-RESULT

Note: For each case the addition statement can be replaced by the appropriate statement. Substitute the first letter of the operation in place of the letter A in each of the addition instructions.