To: Distribution

From: Steve Webber

Subject: New data types within Multics

Date: 10/26/76

Introduction

This MTB describes proposed changes to various areas of the system necesary to support several new data types. The need for the new data types arises from several fronts including new customers (DCC) and potential customers (IRS). The need for formally defining other data types also arises from COBOL which already partially supports some of these (overpunched sign) because they are required of a standard COBOL implementation. It is also desired that the system not support an arbitrary number of different data types and that PL/I and COBOL have common data types to the degree possible.

The Issues

The following problems/questions have been brought up:

1. Should the data types supported by Multics be only those -- and all those -- supported by our hardware?

2. Should the data types supported by Multics also include data types supported by other important computers even though our hardware does not easily match it?

3. Should the data types supported be supported by the PL/I compiler? Or just the system, i.e., the debuggers, stu_, any_to_any_, etc?

Some interesting consequences arise with answers to each of the questions. In particular, it would seem quite reasonable, on first analysis, that we should support all and only the data types that our hardware does. This has the interesting problem of leaving out some important data types that we don't support today but that we probably should. One such data type is the ANSI standard default data type for COBOL!

The Hardware

The hardware supports the following 12 basic data types:

fixed decimal leading sign 9-bit
fixed decimal leading sign 4-bit
fixed decimal trailing sign 9-bit
fixed decimal trailing sign 4-bit
fixed decimal unsigned 9-bit
fixed deciaml unsigned 4-bit
float decimal (leading sign, trailing exponent) 9-bit
float decimal (leading sign, trailing exponent) 4-bit
fixed binary short
fixed binary long
float binary short
float binary long

The hardware works equally well on digit-aligned data as well as byte-, word-, or double-word-aligned data.

PL/I could use all of these data types and in addition a COMPLEX version of some of them. In particular, PL/I would reasonably understand the following:

					TOTAL	OLD	NEW
real complex	fixed	decimal	leading sign trailing sign	9BIT 4BIT	8	2	6
real	fixed	decimal	unsigned	9BIT 4BIT	2	-	2
real complex	float	decimal		9BIT 4BIT	4	2	2
real	fixed	binary	short long	signed unsigned	4	2	2
complex	fixed	binary	short long	signed	2	2	-
real complex	float	binary	short long	signed	4	4	-
					24	12	12

This leads to 24 arithmetic data types of which PL/I and the system currently support 12.

COBOL Requirements

The ANSI standard default data type (DISPLAY data) for COBOL-74 is

real	fixed	decimal	leading sign trailing sign	9BIT	2	-	2
					26	12	14

BUT with the sign "overpunched" with the first or last digit. Clearly this works only with 9BIT data and has its history from the (still active) world of CARDS. However, it should probably

be considered. COBOL, for example, must support this (at significant extra cost in execution speed of compiled programs since our hardware has no support of it) and hence, probably so should the system.

COMPACT Attribute

Since PL/I confuses the packing of data and the alignment (by forcing all specification with only the (un)aligned attribute) it is difficult to introduce packed decimal data into PL/I in an arbitrary way. It is proposed that a new attribute (keyword in PL/I) be added to PL/I to aid in the declaration of variables. The new attribute might be "packed" but because of an already existent use of this word in various descriptions I would propose "compact". This would indicate for decimal data that 4-bit instead of 9-bit digits are used. The alignment attribute would then mean either word-aligned, digit-aligned, or byte-aligned -- to be determined.

Note that I would also propose using the compact attribute for pointer data with the following defaults:

```
aligned    =>  not compact
unaligned  =>  compact
```

This would allow for compatibility with current programs and also provide for "aligned compact" data. (It would not be possible to specify "unaligned and not compact".)

Alignment

The prime question with 4-bit data is whether it is aligned on a byte boundary or on a digit boundary. The DCC (Burroughs) data type must be digit aligned because they have programs (that can't be changed) that "redefine" unsigned data. For example (in COBOL terms):

```
01  data pic 99 comp-8.
01  data1 redefines data.
  02  first pic 9 comp-8.
  02  second pic 9 comp-8.
```

Clearly there is no room for a sign and "second" must appear on a digit boundary (comp-8 data is packed decimal with or without sign).

The (illegal, but useful) PL/I equivalent of this might be:

```
dcl 1 data fixed decimal (2,0) unsigned compact;
dcl 1 data1 based (addr (data)),
    2 first fixed decimal (1,0) unsigned compact,
```

2 second fixed decimal (1,0) unsigned compact;

The problem with digit-aligned vs byte-aligned data is twofold. First, it is harder to change the compiler because there is no concept of a unit of storage of size 4.5 bits. Second, the runtime (stu_, debug, etc.) has a similar problem. In fact, the runtime symbol node (of the symbol table generated by -table) has two 2-bit unit size fields allowing sizes of 36, 1, 9, and 18 bits. It is proposed to use a currently unused bit in the runtime symbol node to extend these 2-bit fields to 3-bit fields thereby allowing specifications of a unit size of 1 digit. In particular, the mappings would be:

OFFSET	New bit	Old bits
word	0	00
bit	0	01
byte	0	10
halfword	0	11
undefined	1	00
undefined	1	01
undefined	1	10
digit	1	11

any_to_any_

A large task to be performed if we are to add new arithmetic data types to the system is the changing of any_to_any_ to handle them (any_to_any_ currently performs all legal conversions between arithmetic and/or string data according to the PL/I conversion rules). If we change the number of arithmetic data types from 12 to 26 this would mean changing the number of conversions from 14 (including 2 string types) to 28 or handling 28*27=756 instead of 14*13=182 conversion. Although this is potentially a big problem, it is made somewhat easier due to certain consistencies of the 68/80 decimal unit's handling of data of different types.

Proposal

It is proposed that the following actions be taken (in time frames to be determined later):

1. Change any_to_any_ to support those data types used by COBOL for MR5.0. This includes COMP-5 (byte-aligned, right-justified, optional trailing sign, packed decimal -- the IBM format) and COMP-8 (digit-aligned, optional trailing sign, packed decimal -- the Burroughs format).

2. Change any_to_any_ to support conversions bewteen all arithmetic data types standardized in the system.

3. Change debug to understand COMP-5 and COMP-8 data for displaying purposes.

4. Change debug to understand all data types standardized in the system.

5. Add the unsigned attribute to the Multics PL/I language (for real fixed data).

5. Add the compact attribute to the Multics PL/I language (for decimal and pointer data).

Note that the proposed PL/I packed decimal data would be compatible with the COBOL COMP-8 data and be aligned as follows:

aligned	compact	new	resultant alignement
yes	no	no	word
yes	yes	yes	word
no	no	no	byte
no	yes	yes	digit

The debug changes in step 3 should be made available with MR5.0. The any_to_any_ changes in step 1 should also be made available with MR5.0, but this is not as critical. It would allow probe to work with these data types.

The unsigned attribute, proposed by the MSPL committee for real fixed binary data, would be extended to apply to real fixed decimal data as well.

DEBUG Changes

There are 2 major changes to debug that are necessary if it is to be helpful in debugging programs with the various new data types. These are, 1) displaying arbitrary data as if it were of a given type, and 2) assigning to arbitrary data a constant of a given data type. In order to do these it seems necessary that we establish a method of talking about the various data types, i.e. naming them. The proposed method for output is an extension of the current (signle-letter) output mode character convention to include many new (multiple-letter) output mode sequences. The proposed method for input is an extension of the current character and bit string conventions.

For output, the following new modes are proposed:

TYPE		DEBUG MODE	DESCRIPTION
1		lss	real fixed binary short
2		lsl	real fixed binary long
3		fls	real float binary short
4		fll	real float binary long
5		clss	complex fixed binary short
6		clsl	complex fixed binary long
7		cfls	complex float binary short
8		cfll	complex float binary long
9		ls9	real fixed decimal leading sign 9-bit
10		fl9	real float decimal 9-bit
11		cls9	complex fixed decimal leading sign 9-bit
12		cfl9	complex float decimal 9-bit
29		ls9o	real fixed decimal leading sign overpunched
30		ts9o	real fixed decimal trailing sign overpunched
31	NAP	cls9o	complex fixed decimal leading sign overpunched
32	NAP	cts9o	complex fixed decimal trailing sign overpunched
33		nss	real fixed binary short unsigned
34		nsl	real fixed binary long unsigned
35		us9	real fixed decimal unsigned 9-bit
36	NP	ts9	real fixed decimal trailing sign 9-bit
37	NAP	cts9	complex fixed decimal trailing sign 9-bit
38		us4	real fixed decimal unsigned 4-bit
39	NP	ts4	real fixed decimal trailing sign 4-bit
40	NAP	cts4	complex fixed decimal trailing sign 4-bit
41		ls4	real fixed decimal leading sign 4-bit
42		fl4	real float decimal 4-bit
43		cls4	complex fixed decimal leading sign 4-bit
44		cfl4	complex float decimal 4-bit
45		pp	packed pointer

These are derived from the possible combinations of attributes described by the following diagram:

$$[c]\begin{Bmatrix} fl \\ ls \\ ns \\ ts \end{Bmatrix}\begin{Bmatrix} l \\ s \\ 9 \\ 4 \end{Bmatrix}[o]$$

where:

c => complex
fl => floating point
ls => leading sign (or 2-s complement sign)
ns => unsigned
ts => trailing sign
l => long binary
s => short binary
9 => non-packed decimal
4 => packed decimal
o => overpunched sign

The codes NP and NAP indicate that it is proposed there be no support for this data type in PL/I or in PL/I and any_to_any_, respe tiv ly.
c e
There are some inconsistent combinations that are not included such as unsigned complex and unsigned overpunched sign. (The overpunch can only occur with non-packed, leading sign or trailing sign, fixed decimal, real or complex.)

For input, these modes could be immediately appended t^o a character string to indicate a value of the given mode:

"-321"ls4

would indicate a 4 digit compact decimal number. Similarly:

"12.3e-4"fl9

would indicate a real float decimal non-packed number. The value:

"12.3e-4"ls4

would supposedly be illegal.

Changes to the Symbol Table

The runtime symbol table would be changed (augmented) in the following way:

1. The size field associated with decimal data would include a count of the number of digits in the datum, not including any sign that might be present.

2. The data type number would be one of those given above in the debug modes table.

3. The unit fields would be extended as mentioned above.