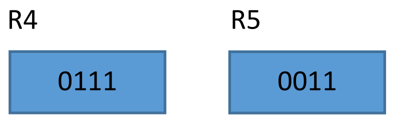
Techniques for Converting Double Precision Binary to Packed Decimal

Converting a binary single precision (one 32-bit register) value to packed decimal is easy: Invoke CVD (Convert to Decimal) to place the equivalent packed decimal result in an 8-byte (doubleword) field. Since the range of integers in a single register is −2,147,483,648 to 2,147,483,647, and the doubleword can contain up to 15 decimal digits, the result can always be converted without loss of precision.

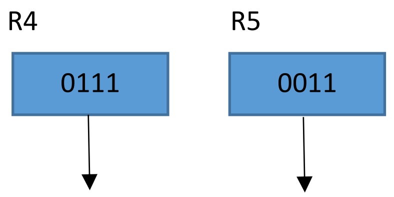
But how do you convert double precision (64-bit) results when the value to be converted is spit across two single 32-bit registers? This case occurs when we multiply two 32-bit values. To get us thinking about this problem, imagine that each register contains only four bits instead of 32 or 64. The possible 2's complement values and their decimal equivalents for such 4-bit registers are listed below:

2's Comp	**Decimal**
0000	0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7
1000	-8
1001	-7
1010	-6
1011	-5
1100	-4
1101	-3
1110	-2
1111	-1

Continuing this line of thought, a double precision value would occupy two registers and contain 8 bits. Assume that we have a CVD instruction that converts a single 4 bit register to decimal. Let's take a concrete example with a binary 8-bit value 01110011 contained in registers 4 and 5 as pictured below.

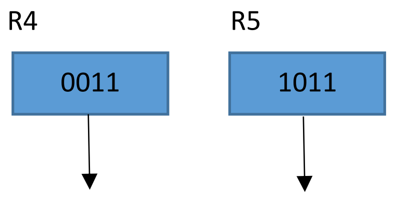


If we apply CVD to each register, we end up with two values: 7 and 3. The 3 correctly represents the value in R5, but the one bits in R4 are converted as if their values represent 4, 2, and 1 respectively, moving left to right in the register. In fact, those bits represent the values 64, 32, and 16 in our double precision value. Each bit is off by a factor of 16 = 2^4. To adjust for this we multiply 7 = 4 + 2 + 1 by 16, and then add the 3 (which was correctly converted) to produce the correct result.



(7 x 16) + 3 = 115

But hold on - there is one flaw in our conversion plan. What happens if the low order register (R5 above) contains a 1 in the high order bit? Let's take a concrete example with a binary 8-bit value 00111011 contained in registers 4 and 5. In this case, CVD converts R5 to its negative value, -5, when it should have been converted as +11. In this case we are off by 16 = 2^4. Negative values will always cause us to be off by 16. Moving to 32-bit registers with negative values, we will always be off by a factor of 2^{32}.



(3 X 16) + -5 = 43 (Adding 16 produces the correct answer 59)

In the case where R5 contains a negative value, we can easily correct by adding 1 to R4 (which would be equivalent to adding 16 = 2^4 for 4-bit registers, or 2^{32} in the case of 32-bit registers).

We can now formulate a plan for converting a double precision value contained in two 32-bit registers (an even/odd register pair) to packed-decimal.

1) First, test the odd register. If it contains a negative value, add 1 to the even register.
2) Use CVD to convert the even register and multiply the result by 4,294,967,296 = 2^{32}.
3) Use CVD to convert the odd register and add it to the result in 2).

This technique works for all values (positive and negative) contained in the two registers.

Here is the assembler code for generating a double precision result in R4/R5 and converting it back to packed-decimal in PRODPK. We assume X and Y are binary fullwords.

```
         L     R5,X
         M     R4,Y
         LTR   R5,R5                 DOES LOWER ORDER REG LOOK NEG?
         BNM   NOCORR                IF NOT...TAKE BRANCH
         A     R4,=F'1'              CORRECT BY FACTOR OF 2^32
NOCORR   EQU   *
         ZAP   PRODPK,=P'0'
         CVD   R4,PRODPKR
         MP    PRODPK,TWO32          RESULT IS OFF BY THIS FACTOR
```

```
          CVD   R5,DBLWD                GRAB THE LEAST SIGNIF PART
          AP     PRODPK,DBLWD           ADD IN THE LOW-ORDER PART
          …
          DS  0D
DBLWD     DS   D
PRODPK    DS  0PL16
PRODPKL   DS   PL8
PRODPKR   DS   PL8
TWO32     DC   P'4294967296'     2^32
```

We can also produce double precision results (assume X and Y are fullwords) by multiplying with grande instructions:

```
        LGF   R8,X
        MSGF  R8,Y
```

We end up with a double precision result (64 bits) in a single grande register. In this case we can invoke the grande version of CVD to get back to packed-decimal assuming XTIMESY is defined as PL16.

```
        CVDG  R8,XTIMESY
```

This grande solution suggests an alternative to the first technique we discussed for converting double precision results contained in an even/odd register pair. Why not move the even/odd value to a grande register and then invoke CVDG? Code for this technique is presented below:

```
          L      R5,X
          M      R4,Y
          ST     R4,XP
          ICMH  R5,B'1111',XP
          CVDG  R5,XTIMESY
          …
XP        DS     F
```

The first two lines produce a double precision result in even/odd registers R4 and R5. We then move the contents of R4 into the high-order part of grande register R5 with Store Fullword and Insert Characters Under Mask High instructions. This allows us to use CVDG on R5 to complete the conversion.

When converting between data types we have to give some thought to whether the conversion is even possible. In other words, can every 32-bit binary value be represented in 8-byte packed-decimal format if we invoke CVD, and can every 64-bit binary value be represented in 16-byte packed-decimal format if we invoke CVDG? In the case of 32-bit binary values, the range of integers is +2,147,483,647 to −2,147,483,648. Any number in this range can be represented using the 15 decimal digits in our packed decimal target. In the case of 64-bit values, the range of integers is +9,223,372,036,854,775,807 to −9,223,372,036,854,775,808. Since our packed-decimal target is 16 bytes, and can contain 31 decimal digits, converting from binary to packed will not be a problem. Packed-decimal is a “larger” data type than binary, so the only conversions that will cause an exception are those from packed-decimal back to binary.