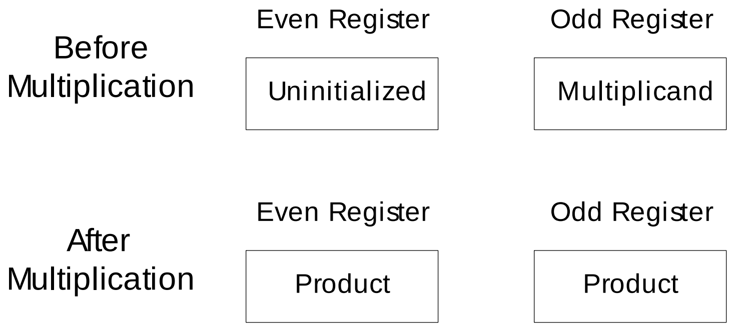
**M R1,D2(X2,B2)**

# Multiply

**RX**

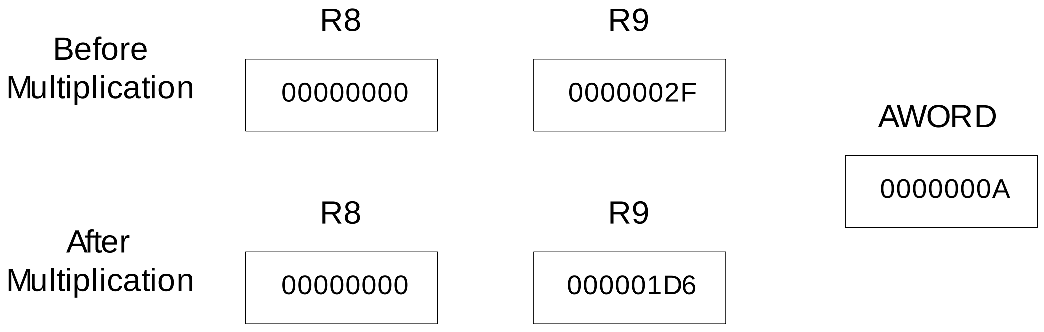
| Op Code | $R_1X_2$ | $B_2D_2$ | $D_2D_2$ |
|---|---|---|---|

The Multiply instruction performs 2’s complement binary multiplication. Operand 1 names an even register of an “even-odd” consecutive register pair. For instance, R2 would be used to name the R2 / R3 even-odd register pair, and R8 would be used to name the R8 / R9 even-odd register pair. Operand 2 is the name of a fullword in memory containing the multiplier. Before the multiplication, the even register can be left uninitialized, while the odd register contains the multiplicand. After the multiplication, the product occupies the even-odd register pair in 2’s complement format.



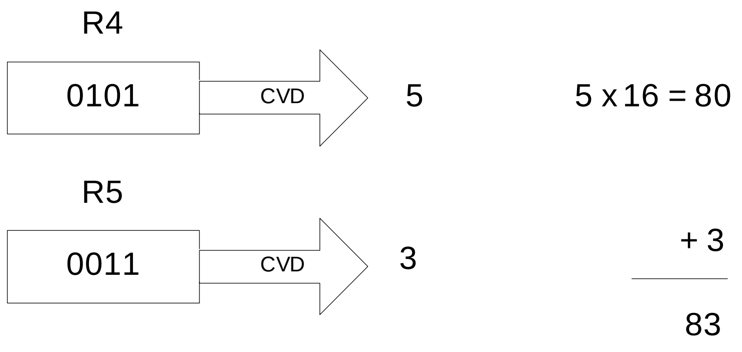
If the product is less than $2^{31} - 1$ = 2,147,483,647 then the answer can be found in the odd register. We may then use **CVD** and **ED** in order to print the product. Otherwise, the even-odd pair must be treated as a large (64 bit) 2’s complement integer. Printing such an integer requires special treatment which we will consider later. First, lets look at an example multiplication. We assume that AWORD is a fullword in memory containing the integer 10.

```
L     R9,=F’47’    PUT MULTIPLICAND IN ODD REGISTER
M     R8,AWORD     MULTIPLY 47 TIMES 10
```

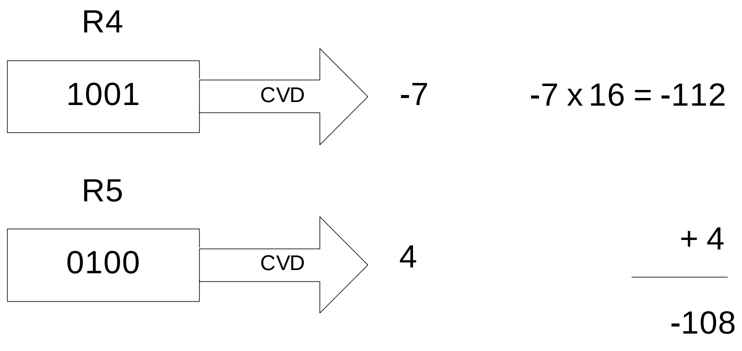


First the multiplicand, 47, is loaded into the odd register. The even register is left uninitialized. The multiplier, AWORD, contains a 10 and is not affected by the multiplication. After the multiplication, the register pair R8 / R9 contains a 64 bit 2's complement integer. Since the product is sufficiently small, R9 by itself contains a valid representation of the product.

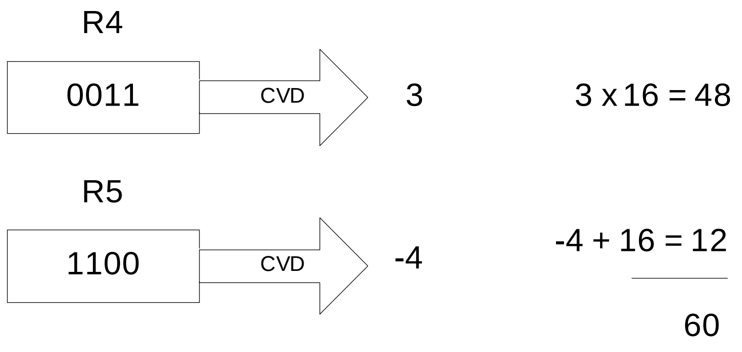
When the product will not fit in the odd register, we must provide special handling in order to convert the product to a packed representation. If it takes two registers to hold the a result, we will call the answer a "double precision" result. Unfortunately, there is no instruction that will convert a 2's complement double precision integer to a packed decimal format. **CVD** can be used to convert a single register to packed format, so we will investigate how this instruction can be used on both registers. To simplify the computations, we will assume that the registers contain 4 bits instead of 32. Suppose a multiplication has produced a double precision product of 83 in registers R4 and R5. Then, since 83 = B'01010011', and assuming 4 bit registers, R4 = B'0101' and R5 = B'0011'. If we use **CVD** to convert R4 we would get 5, when in fact, the bits in R4 represent 80 if we look at the 2's complement integer contained in R4 and R5. We are off by a factor of $2^4$ =16 since 5 x 16 = 80. If we use **CVD** to convert R5 we would get 3, which is what the bits in the double precision integer represent. The true answer can be recovered by adding (5 x16) + 3. This is illustrated in the diagram below.



This procedure also works for some negative double precision integers. Consider the double precision integer -108. Using 4-bit registers R4 and R5, we see that R4 contains B'1001' and R5 contains B'0100'. **CVD** converts R4 to -7 when, in fact, the bits in R4, B'1001', represent -112 = -7 x 16. R5 = B '0100' is converted to 4. Adding -112 + 4 we get the correct double precision answer -108. This is illustrated below.



A problem with this method occurs when the odd register contains a 1 in the high-order bit. Consider the double precision integer 60 = B'00111100'. Assume R4 contains B'0011' and R5 contains B'1100'. The conversion is illustrated below.



R4 is converted to 3, multiplied by 16, and correctly converted to 48. On the other hand, R5 is converted to -4 since the high order bit was a 1. R5 should have been converted to 12. We are off by a factor or 16. If we add 16, the conversion to 12 will be correct.

These examples lead us to a conversion algorithm for double precision results:

1) Test the odd register to see if it is negative. If it is, we need to add $2^{32}$ = 4,294,967,296 ( we are using 32-bit registers instead of 4-bit registers ) to the final result. An easy way to do this is by adding 1 to the even register - the rightmost bit in the even register represents $2^{32}$.

2) Convert the even register to packed decimal and multiply the result by $2^{32}$.

3) Add in the result of converting the odd register to packed decimal.

Here is an example in assembler language of the algorithm described above. The example illustrates how a double precision result in R4 and R5 could be converted to packed decimal in a doubleword.

```
          LTR    R5,R5           DOES R5 LOOK NEGATIVE?
          BNM    LOOKSPOS        DON'T ADD TO R4 IF POSITIVE
          A      R4,=F'1'        WE ARE OFF BY 2 TO THE 32 POWER
LOOKSPOS  EQU    *
          CVD    R4,RESULTRT     CONVERT AND GET READY..
          MP     RESULT,TWOTO32  MULTIPLY BY 2 TO THE 32
          CVD    R5,DOUBWORD     CONVERT ODD REG TO DECIMAL
          AP     RESULT,DOUBWORD ADD THE TWO COMPONENTS
          ...
RESULT    DS    0PL16   NEED A LARGE AREA TO HOLD DOUBLE PRECISION
RESULTLF  DC     X'00000000' NEEDS TO BE 0'S AFTER CONVERTING R4
RESULTRT  DS     PL6    WORK AREA FOR R4
TWOTO32   DC     P'4294967296'  2 TO THE 32ND POWER
DOUBWORD  DS     D      CONVERSION AREA FOR CVD
```

**Examples**

**Some Unrelated Multiply Instructions**

```
L    R7,=F'100'   MULTIPLICAND GOES IN THE ODD REGISTER
M    R6,=F'10     R6 = X'00000000' = 0, R7 = X'000003E8' = 1000

L    R7,=F'3'     MULTIPLICAND GOES IN THE ODD REGISTER
M    R6,=F'-2'    R6 = X'FFFFFFFF', R7 = X'FFFFFFFA' = -6
```

```
L    R3,=F'8'      MULTIPLICAND GOES IN THE ODD REGISTER
M    R2,=F'1'      R2 = X'00000000', R3 = X'00000008'

L    R3,=F'8'      MULTIPLICAND GOES IN THE ODD REGISTER
M    R2,=F'0'      R2 = X'00000000', R3 = X'00000000'

L    R5,=X'FFFFFFFF'    ALL 1'S IN ODD REG
M    R4,=F'2'      MULTIPLYING BY 2 SHIFTS ALL BITS 1 BIT LEFT
                   R4 = X'00000001', R5 = X'FFFFFFFD', THIS IS A
                   DOUBLE PRECISION RESULT
```

## Tips

1) Know your data! In most cases, the product of a multiplication will fit in the odd register where it can easily be converted back to packed decimal. If you have any doubts about the size of a generated product, you must convert the double precision result from both the even and odd registers as described above.