MATHEMATICAL INSTRUCTIONS

Multiply unsigned
MULU <ea>,Dn

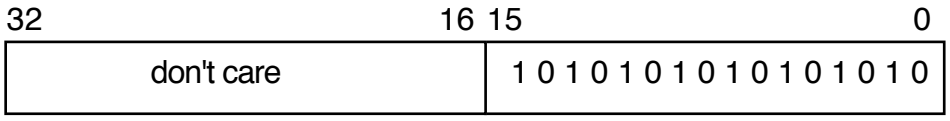
Action Multiplies the word length <ea> times the least significant word in Dn. The result is a long word.

Notes:
1. The lowest word of Dn contains the multiplier.
2. The result is a 32-bit long word.
3. The negative (N) and zero (Z) flags are set according to the result. The overflow (V) and carry (C) bits are always cleared since there can never be an overflow or carry with this instruction.

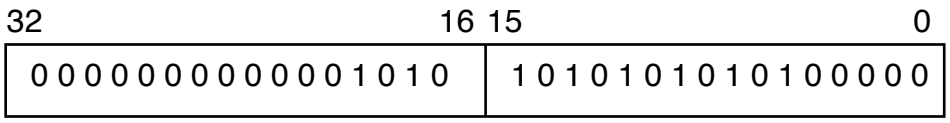
Example:

```
MULU #$10,D4
```

BEFORE



AFTER



result extends into upper word

Multiply signed
MULS <ea>,Dn

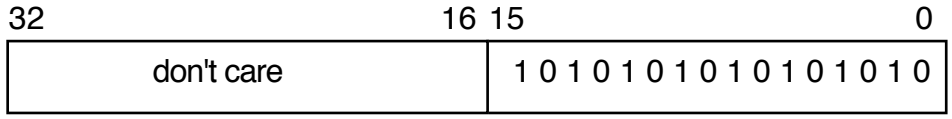
Action Multiplies the word length <ea> times the least significant word in Dn. The result is a sign extended long word.

Notes:
1. The lowest word of Dn contains the multiplier.
2. The result is a 32-bit long word which takes account of the multiplier and multiplicand's signs.
3. The negative (N) and zero (Z) flags are set according to the result. The overflow (V) and carry (C) bits are always cleared since there can never be an overflow or carry with this instruction.

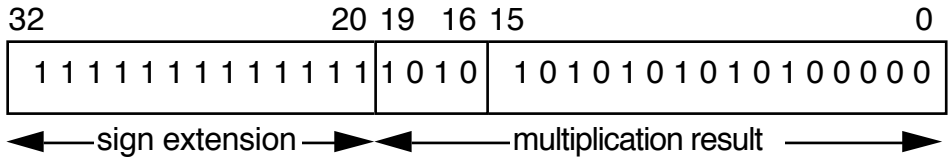
Example:

MULS #$10,D4

BEFORE

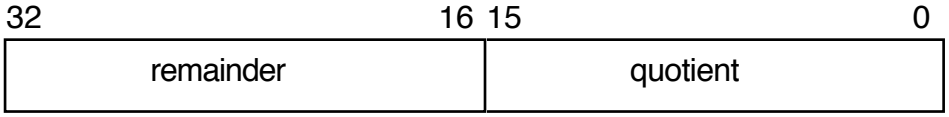


AFTER



Divide unsigned
DIVU <ea>,Dn

Action Divides the 32-bit integer in Dn by the 16-bit integer in <ea>. The most significant word of the 32-bit result is the remainder; the least significant word is the quotient.

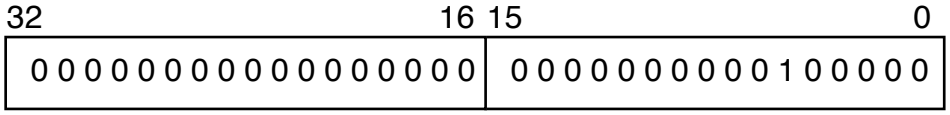


Notes:
1. There is also a DIVS but you will need to sign extend what's in Dn before you can divide with sign. This can be done using the instruction EXT.L, which extends the lowest word to a long word, for signed numbers.
2. You may use the instruction ANDI.L #$0000FFFF,Dn to clear bits 16-32 for unsigned number division.

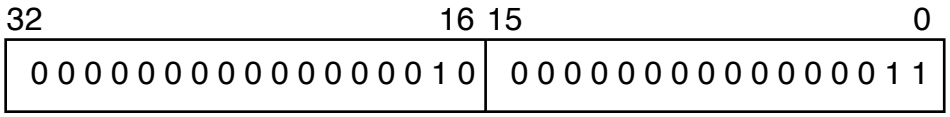
Example:

```
DIVU  #10,D4
```

BEFORE (D4 contains 32_{10}) Note that 32_{10} = $20 = %100000



AFTER (the result is a quotient of 3_{10} with a remainder of 2_{10})



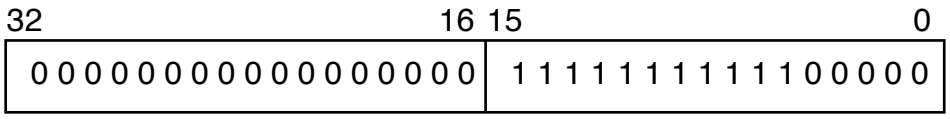
remainder = 2 quotient = 3

Example:
SUppose you want to do a signed divide of -32 in D4 by 10, i.e.

```
DIVS  #10,D4
```

Consider what happens if you put -32 in D4 using a MOVE immediate

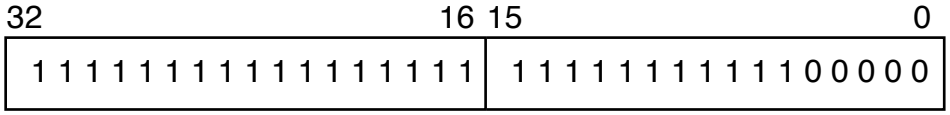
```
MOVE.W      #-32,D4
```



The -32 is sign-extended to a word.

You must extend this to a long word before you can do a DIVS

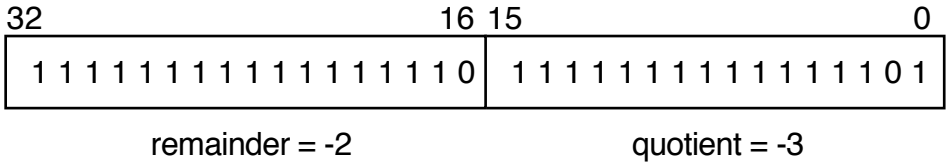
```
EXT.L D4
```



Now you can correctly use the DIVS

```
DIVS  #10,D4
```

to get the resulting quotient of -3_{10} with a remainder of -2_{10},
i.e. (D4) = $FFFE FFFD



MATH INSTRUCTIONS

Instruction	Comments
ADDI	Add a constant, cannot be used with An as a destination.
ADDQ	Adds an immediate constant between 1 and 8.
SUB SUBI SUBQ	flags in normal way
SUBX	Clears Z only if the result is non-zero, i.e. it sets Z to 1 if the result is zero else Z remains unchanged. This instruction subtracts the source and the X bit from the destination.
ADDX	basically the same as SUBX but adds.
SUBA	Doesn't effect status register.
NEG	Negates (subtracts from zero). WARNING: NEG is NOT a COMPLEMENT. It computes 0 - (Destination) - (X-bit)
NEGX	Adds X bit to destination, then subtracts from zero.
MULS MULU	Multiply two words.
DIVS DIVU	Divides a long word by a word. WARNING: Division by zero generates a TRAP.
EXT	Sign extend

EXAMPLE: DOUBLE PRECISION ADDITION

This program adds two 64-bit (8-byte) numbers together.

The program uses:

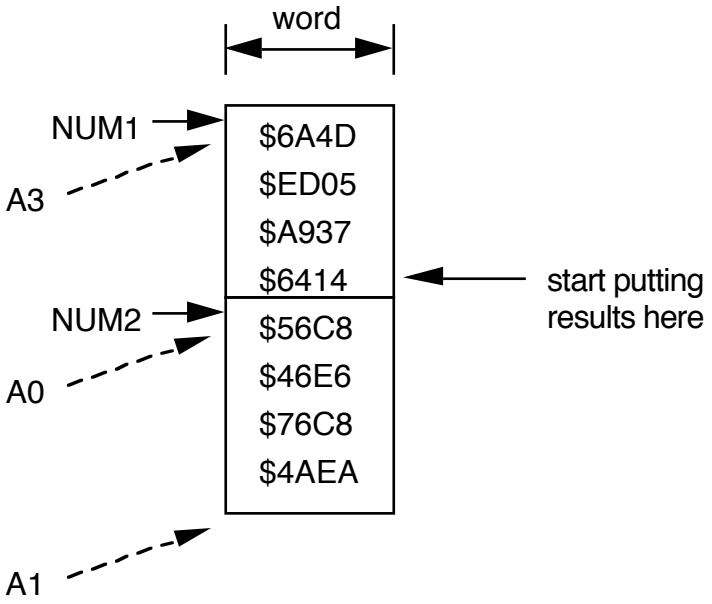
NUM1	64-bit number, 8 consecutive bytes
NUM2	64-bit number, 8 consecutive bytes stored immediately higher in memory than NUM1

Functional specification (pseudocode)

```
A3 = starting address of NUM1
A0 = A3 + 8                    ;starting address of NUM2
A1 = A0 + 8                    ;ending address of NUM2 plus 1
                                byte
X = 0                          ;clear X-bit

;loop starting with least significant bytes

FOR j = 1,4 DO
NUM1(j) = NUM1(j) + NUM2(j) + X;
```



MC68000 assembly code for double precision add program:

```
            ORG     $5000
NUM1        DC.W    $6A4D, $ED05,$A937,$6414
NUM2        DC.W    $56C8, $46E6,$76C8,$4AEA

BYTECNT     EQU     8                   ;number of bytes to add together
MAIN        LEA     NUM1,A3             ;use A3 for address first number
            LEA     BYTECNT(A3),A0      ;the second number begins 8
                                        bytes beyond the beginning of the
                                        first number - use address
                                        register indirect with
                                        displacement
            LEA     BYTECNT(A0),A1      ;address beyond end of second
                                        number
            MOVEQ   #0,CCR              ;clear the X bit of the SR
            MOVEQ   #BYTECNT-1,D2       ;set up loop counter, adjusted for
                                        DBRA.  MOVEQ is ok since
                                        counter is 7
LOOP        MOVE.B  -(A0),D0
            MOVE.B  -(A1),D1
            ADDX.B  D1,D0               ;D0=D0+D1+X-bit
            MOVE.B  D0,(A0)             ;save result in NUM1
            DBF     D2,LOOP             ;repeat 7 times
            END     MAIN
```

EXAMPLE: BINARY MULTIPLICATION

This program multiplies two 8-bit unsigned numbers using a shift and add algorithm to generate a 16-bit product.

The multiplier is in D2 and the multiplicand in D1.
The product is returned in D1.

algorithm:

1. Starting with most significant bit of multiplier, i.e. bit=8
2. Shift product to line up properly (product = 2*product)
3. If multiplier[bit] = 1 then product=product+multiplier
4. Decrement bit. If bit≥0 then goto 2.

The program uses:

MULTIPLICAND	8-bit number to be multiplied
MULTIPLIER	8-bit number that MULTIPLICAND is multiplied by
PRODUCT	32-bit result

Functional specification (pseudocode)

```
PRODUCT = 0;                          /*clear PRODUCT*/
BIT=8 /* starting at MSB */


FOR j = 1,8 DO                        /*do for each bit of MULTIPLIER*/
        BEGIN
        PRODUCT = PRODUCT*2;          /*shift PRODUCT left by one bit*/
        IF MULTIPLIER[9-bit] = 1 THEN
                PRODUCT = PRODUCT + MULTIPLICAND;
/* do calculations from most significant bit to least significant bit */

        BIT=BIT-1;                    /* decrement bit */
        END
```

DETAILED EXAMPLE:

multiplier = 61_{16} (97_{10})
multiplicant = $6F_{16}$ (111_{10})

multiplier:	(D2)	=	00000000 01100001	($00 61)
multiplicand	(D1)	=	00000000 01101111	($00 6F)
initial product:	(D0)	=	00000000 00000000	($00 00)
shift product:	(D0)	=	00000000 00000000	($00 00)
MUL[8] = 0 don't add				
new product	(D0)	=	00000000 00000000	($00 00)
shift product:	(D0)	=	00000000 00000000	($00 00)
MUL[7] = 1 so add	(D1)	=	00000000 01101111	($00 6F)
new product	(D0)	=	00000000 01101111	($00 6F)
shift product:	(D0)	=	00000000 11011110	($00 DE)
MUL[6] = 1 so add	(D1)	=	00000000 01101111	($00 6F)
new product	(D0)	=	00000001 01001101	($01 4D)
shift product:	(D0)	=	00000010 10011010	($02 9A)
MUL[5] = 0 don't add				
new product	(D0)	=	00000010 10011010	($02 9A)
shift product:	(D0)	=	00000101 00110100	($05 34)
MUL[4] = 0 don't add				
new product	(D0)	=	00000101 00110100	($05 34)
shift product:	(D0)	=	00001010 01101000	($0A 68)
MUL[3] = 0 don't add				
new product	(D0)	=	00001010 01101000	($0A 68)
shift product:	(D0)	=	00010100 11010000	($14 D0)
MUL[2] = 0 don't add				
new product	(D0)	=	00010100 11010000	($14 D0)

shift product:	(D0)	=	00101001 10100000	($29 A0)
MUL[1] = 1 so add	(D1)	=	00000000 01101111	($00 6F)
new product	(D0)	=	00101010 00001111	($2A 0F)
final answer:	(D0)	=	00101010 00001111	($2A 0F)

where $2A0F = $10767_{10} = 97_{10} \times 111_{10}$

MC68000 assembly code for binary multiply program:

```
            ORG     $5000
A           DC.W    $61
B           DC.W    $62
RESULT      DS.L    1

MAIN        CLR.L   D0          ;clear 32-bit product register
            MOVE.L  D0,D1       ;clear upper word for ADD.L
            MOVE.W  A,D1        ;copy multiplicand into D1
            MOVE.W  B,D2        ;copy multiplier into D2
            MOVE.W  #16-1,D3    ;loop count = 16-1 for DBRA
                                instruction
LOOP        ADD.L   D0,D0       ;shift product left one bit
            ADD.W   D2,D2       ;shift multiplier left one bit
            BCC.S   STEP        ; Use carry to check whether to
                                add.  If carry=0 goto next step.
            ADD.L   D1,D0       ;if multiplier [15] was one then
                                add multiplicand.
STEP        DBRA    D3,LOOP     ;else continue with loop
            LEA     RESULT,A1   ;get where to put answer
            MOVE.L  D0,(A1)     ;store result
            END     MAIN
```

NOTES:

1. Program uses shift and add algorithm.
2. DBRA is equivalent to DBF and works in most assemblers.

REVIEW for Integer arithmetic functions

ADD.<size> <source>,<destination>
One operand MUST be a data register; affects all five status codes in CCR

Overflow (V)
Set if two like-signed numbers (both positive or both negative) are added and the has a different sign. This is causes by the result exceeding the 2's complement range of numbers, causing the sign bit to change. Mathematically, $V = C_s \oplus C_p$

The V and N flags are pertinent only for signed numbers but are set for all additions.

ADDA <ea>,An
If the destination is an address, the condition codes are not changed.

For adding multiple words, the extend can be used.

ADDX
Adds two (data) registers or memory locations. However, zero is only cleared if the result is non-zero; otherwise, zero is unchanged.

ADD.L D0,D2
ADDX.L D1,D3
The above code adds the double precision numbers together:

	X	
D1		D0
D3		D2

Memory to memory adds do not change X, Z. You must set them. For example:

MOVE #4,CCR ;sets Z bit, clears all others