double.s explained

The program double.s, when compiled and loaded to the experiment board, will act as a calculator that can only multiply the input number by two. When run, the LCD will display two zeros at the rightmost columns of both lines. The following keypad assignment was done:

7	8	9	*
4	5	6	*
1	2	3	*
*	0	*	C

This assignment is achieved by means of a lookup table.

Buttons assigned to "*" are not functional. "C" acts as CLEAR, which will return the calculator to its RESET state. (The word STATE appears in this context, as this is indeed a state machine, but we will defer a more formal treatment to a later example).

The simple calculator uses the LCD_line1 and LCD_line2 arrays as its X- and Y- registers, which, in conventional calculators, are assigned the role of keeping the user input and the result, respectively. As none of the internal registers of the microcontroller are capable of handling a 15 and 16 digit number, which are the input and result, respectively (the maximum number that fits into a 16 bit register is 65535), the numbers will be kept in the display arrays, each byte-sized element of which will be holding one decimal digit.

The arrays initially contain the following values:

LCD_line1 :
LCD_line2 :

Yes, those empty locations contain " ", or blank, or FKA the space character, which is 32 in decimal, or 0x20 in hexadecimal.

Other than the LCD variables, the .bss section declares two more variables:

X_cnt : number of digits so far entered by the user
carry : temporary storage for carry generated during multiplication

As expected, X_cnt and carry are both initialized to zero.

The calculator operates in the following sequence:

Each time the user enters a number,

① The previously entered digits are shifted left by one location.
② The newly entered digit is displayed on the right most corner of line1.
③ X_cnt is incremented by one.
④ The user entry is multiplied by two and the result is displayed on line2.

There are three exceptions to the above sequence:

① The user enters a zero as the first digit: This zero is discarded.
② For the first non-zero digit entered the display is not shifted.
③ If X_cnt = 15, further entries are discarded.

The multiplication is performed using the algorithm taught in primary school:

```
For i = 0 to (X_cnt)-1

    Multiply the 10^i th digit by 2
    Add CARRY from previous run of the loop
    Divide by 10
    10^i th digit of RESULT ← REMAINDER
    CARRY for the next run ← QUOTIENT
```

After a keypad entry has been detected, the program translates the user entry to its assigned value. The program first checks for "*". This entry makes the program to ignore the input.

The next checked character is "C". If "C" is detected, the display is cleared, and X_cnt is zeroed.

If neither "*", nor "C" is pressed, then the program checks for X_cnt = 15. If yes, the program ignores user input.

The next check is for X_cnt = 0. If this is the first digit entered by the user, then the display should not be shifted. The section of the program after the no_shift label performs a further check: If "0" is entered as the first digit it is discarded.

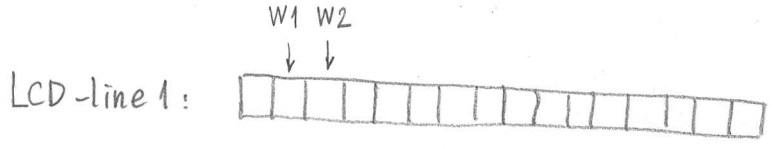
At this point of the program, there is a valid numeric entry which should be processed. First, the display is shifted:

```
mov #LCD_line1, W1
```

passes a pointer to the 1st element in the LCD_line1 array. As the number of digits entered by the user is limited by 15, this location will always be empty, hence it does not need to be shifted.

```
add W1, #1, W1
add W1, #1, W2
```

produces the following two pointers:



Consequently,

```
repeat #14
mov.b [W1++], [W2++]
```

transfers the whole array content by one location upwards.

The lines

```
mov #LCD_line1, W1
mov.b W10, [W1+15]
```

place the last entered digit to the very last byte of the LCD_line1 array.

The multiplication starts by

1. Making W11 to point to the end of LCD_line1
2. " W12 " " " " " " LCD_line2
3. Assigning zero to CARRY

```
mov.b [W11--], W0
```

takes the digit to be multiplied by two into W0, post decrementing W11 making it to point to the next digit.

```
subb W0, #0x30, W0      ; converst W0 from ASCII to its numeric value
add.b W0, W0            ; multiplies it by two

mov carry, W1
add W0, W1, W5          ; adds carry from previous multiplication
                        ; and stores the result in W5
```

```
mov #10, W4
repeat #17
div.u W5, W4    ; divides W5 by 10, and places the QUOTIENT and
                ; REMAINDER into W0, and W1, respectively.
```

The QUOTIENT becomes the CARRY for the next multiplication. The REMAINDER is the result to be displayed. It is first converted into ASCII and then placed into LCD_line 2, post decrementing W12.

Finally,

```
mov.b W0, [W12]
```

writes the final carry as the most significant digit of the result. Here, it is worthwhile to note the implementation of "leading zero suppression" in the code, which is a standard feature in a pocket calculator.