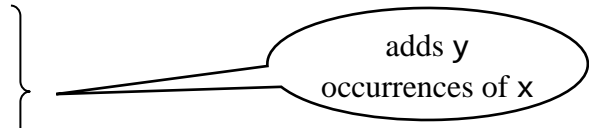


# 11 Multiplication and Division

## Multiplication

```
1 // ex1101.c Multiplication
2 #include <stdio.h>
3 int slowmul(int x, int y)
4 {
5     int product = 0;
6     while (y)
7     {
8         product = product + x;
9         y--;
10    }
11    return product;
12 }
13 //=====
14 int mul(int x, unsigned int y)
15 {
16     int product = 0;
17     while (y)
18     {
19         if (y & 1)                // is rightmost bit of y = 1
20             product = product + x; // accumulate multiplicand
21         y = y >> 1;                // right shift y (the multiplier)
22         x = x << 1;                // left shift x (the multiplicand)
23     }
24     return product;
25 }
26 //=====
27 int main()
28 {
29     printf("%d\n", slowmul(7, 255));
30     printf("%d\n", mul(7, 255));
31     return 0;
32 }
```



The diagram shows a callout bubble pointing to the loop in the `slowmul` function. The bubble contains the text "adds y occurrences of x".

# Multiplying by Hand

Works for unsigned and positive numbers:

$$\begin{array}{r} 0010 \text{ Multiplicand (2 decimal)} \\ 0011 \text{ Multiplier (3 decimal)} \\ \hline 0010 \\ 0010 \\ 0000 \\ 0000 \\ \hline 000110 \end{array}$$

Partial products

4-bit product (6 decimal)

Works for negative numbers:

$$\begin{array}{r} 1111 \text{ Multiplicand (-1)} \\ 1111 \text{ Multiplier (-1)} \\ \hline 1111 \\ 1111 \\ 1111 \\ 1111 \\ \hline 11100001 \end{array}$$

Partial products

4-bit product (+1)

*Rule:* Use same multiply algorithm for sign and unsigned multiplication only if product has same number of bits as operands.

```

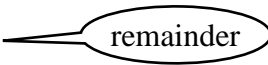
1 ; ex1101.a Multiplication (optimized)
2 startup: bl main
3          halt
4 ;=====
5          ; #include <stdio.h>
6 slowmul: ; r1 is x      ; int slowmul(int x, int y)
7          ; r2 is y      ; {
8
9          ; r0 is product
10         mov r0, 0      ; int product = 0;
11
12 @L0:     and r2, r2, r2 ; while (y)
13         brz @L1        ; {
14
15         add r0, r0, r1  ; product = product + x;
16
17         sub r2, r2, 1   ; y--;
18
19         br @L0          ; }
20 @L1:     ret            ; return product;
21         ; }
22
23 ;=====
24 mul:     ; r1 is x      ; int mul(int x, unsigned int y)
25         ; r2 is y      ; {
26
27         ; r0 is product
28         mov r0, 0      ; int product = 0;
29
30 @L2:     and r2, r2, r2 ; while (y)
31         brz @L3        ; {
32
33         and r3, r2, 1   ; if (y & 1)
34         brz @L4
35
36         add r0, r0, r1  ; product = product + x;
37
38 @L4:     ; shift right y ; y = y >> 1;
39         srl r2
40         ; shift left x   ; x = x << 1;
41         sll r1
42
43         br @L2          ; }
44
45 @L3:     ret            ; }
46 ;=====

```

```
47 main:      push lr          ; int main()
48            push fp          ; {
49            mov fp, sp
50
51            mov r1, 7          ; printf("%d\n", slowmul(7, 255));
52            mov r2, 255
53            bl slowmul
54            dout
55            nl
56
57            mov r1, 7          ; printf("%d\n", mul(7, 255));
58            mov r2, 255
59            bl mul
60            dout
61            nl
62
63            mov r0, 0          ; return 0;
64            mov sp, sp
65            pop fp
66            pop lr
67            ret
68            ; }
```

# Division

To divide  $x$  (the *dividend*) by  $y$  (the *divisor*), subtract  $y$  from  $x$  repeatedly until  $x$  goes negative. The number of subtractions minus 1 is the quotient.

$$\begin{array}{r} 11 \\ - 5 \\ \hline 6 \\ - 5 \\ \hline 1 \\ - 5 \\ \hline - 4 \end{array}$$


```
1 // ex1102.c  Division
2 #include <stdio.h>
3 short div(short x, short y)
4 {
5     int quotient = 0;
6     while (1)
7     {
8         x = x - y;
9         if (x < 0)
10             break;
11         quotient++;
12     }
13     return quotient;
14 }
15 //=====
16 int main()
17 {
18     printf("%d\n", div(77, 7));
19     return 0;
20 }
```

```

1 ; ex1102.a Division (optimized)
2 startup: bl main
3          halt
4 ;=====
5 div:      ; r1 is x      ; int div(short x, short y)
6           ; r2 is y      ; {
7
8           ; r0 is quotient
9           mov r0, 0      ; int quotient = 0;
10
11 @L0:      ; while (1)
12           ; {
13
14           sub r1, r1, r2 ; x = x - y;
15
16           brn @L1        ; if (x < 0)
17                           ; break;
18
19           add r0, r0, 1   ; quotient++;
20
21           br @L0         ; }
22
23 @L1:      ret            ; return quotient;
24           ; }
25 ;=====
26 main:     push lr        ; int main()
27           push fp        ; {
28           mov fp, sp
29
30           mov r1, 77      ; printf("%d\n", div(77, 7));
31           mov r2, 7
32           bl div
33           dout
34           nl
35
36           mov r0, 0      ; return 0;
37           mov sp, sp
38           pop fp
39           pop lr
40           ret
41           ; }

```