Table 3-2. “mpy8s” Performance Figures

Parameter	Value
Code Size (Words)	10 + return
Execution Time (Cycles)	73 + return
Register Usage	• Low Registers :None • High Registers :4 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None

4 16 x 16 = 32 Unsigned Multiplication – “mpy16u”

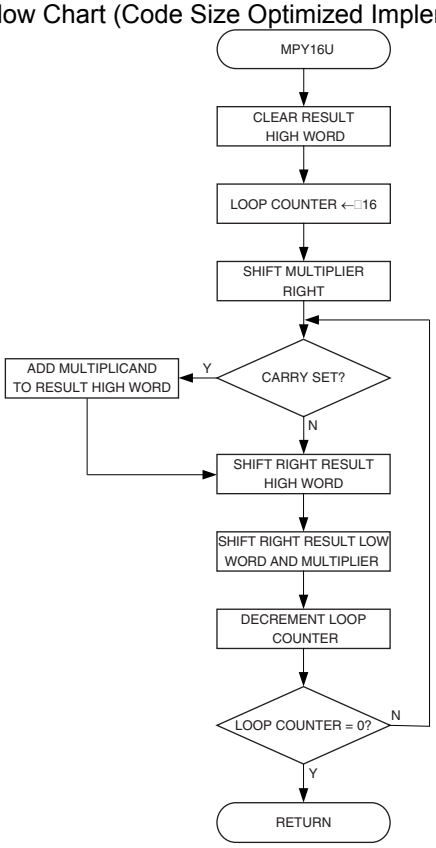
Both program files contain a routine called “mpy16u” which performs unsigned 16-bit multiplication. Both implementations are based on the same algorithm. The code size optimized implementation, however, uses looped code whereas the speed optimized code is a straight-line code implementation. Figure 4-1 shows the flow chart for the Code Size optimized (looped) version.

4.1 Algorithm Description

The algorithm for the Code Size optimized version is as follows:

1. Clear result High word (Bytes 2 and 3)
2. Load Loop counter with 16.
3. Shift multiplier right
4. If carry (previous bit 0 of multiplier Low byte) set, add multiplicand to result High word.
5. Shift right result High word into result Low word/multiplier.
6. Shift right Low word/multiplier.
7. Decrement Loop counter.
8. If Loop counter not zero, go to Step 4.

Figure 4-1. “mpy16u” Flow Chart (Code Size Optimized Implementation)



4.2 Usage

The usage of “mpy16u” is the same for both versions:

1. Load register variables “mp16uL”/”mp16uH” with multiplier Low and High byte, respectively.
2. Load register variables “mc16uH”/”mc16uH” with multiplicand Low and High byte, respectively.
3. Call “mpy16u”.
4. The 32-bit result is found in the 4-byte register variable “m16u3:m16u2:m16u1:m16u0”.

Observe that to minimize register usage, code and execution time, the multiplier and result Low word share the same registers.

4.3 Performance

Table 4-1. "mpy16u" Register Usage (Code Size Optimized Implementation)

Register	Input	Internal	Output
R16	"mc16uL" – Multiplicand Low Byte		
R17	"mc16uH" – Multiplicand High Byte		
R18	"mp16uL" – Multiplier Low Byte		"m16u0" – Result Byte 0
R19	"mp16uH" – Multiplier High Byte		"m16u1" – Result Byte 1
R20			"m16u2" – Result Byte 2
R21			"m16u2" – Result Byte 2
R22		"mcnt16u" – Loop Counter	

Table 4-2. "mpy16u" Performance Figures (Code Size Optimized Implementation)

Parameter	Value
Code Size (Words)	14 + return
Execution Time (Cycles)	153 + return
Register Usage	• Low Registers :None • High Registers :7 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None

Table 4-3. "mpy16u" Register Usage (Straight-line Implementation)

Register	Input	Internal	Output
R16	"mc16uL" – Multiplicand Low Byte		
R17	"mc16uH" – Multiplicand High Byte		
R18	"mp16uL" – Multiplier Low Byte		"m16u0" – Result Byte 0
R19	"mp16uH" – Multiplier High Byte		"m16u1" – Result Byte 1
R20			"m16u2" – Result Byte 2
R21			"m16u2" – Result Byte 2

Table 4-4. "mpy16u" Performance Figures (Straight-line Implementation)

Parameter	Value
Code Size (Words)	105 + return
Execution Time (Cycles)	105 + return
Register Usage	• Low Registers :None • High Registers :6 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None

5 16 x 16 = 32 Signed Multiplication - “mpy16s”

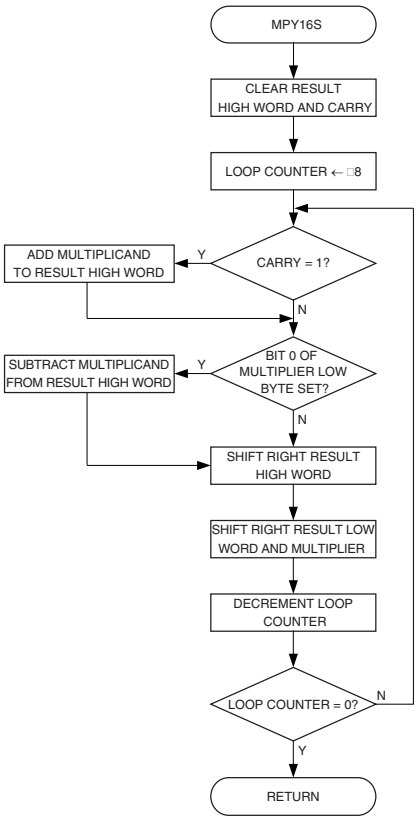
This subroutine, which is found in “avr200.asm” implements signed 16 x 16 multiplication. Negative numbers are represented as 2’s complement numbers. The application is an implementation of Booth’s algorithm. The algorithm provides both small and fast code. However, it has one limitation that the user should bear in mind; If all 32 bits of the result is needed, the algorithm fails when used with the most negative number (-32768) as the multiplicand.

5.1 Algorithm Description

The algorithm for signed 16 x 16 multiplication is as follows:

1. Clear result High word (Bytes 2&3) and carry.
2. Load Loop counter with 16.
3. If carry (previous bit 0 of multiplier Low byte) set, add multiplicand to result High word.
4. If current bit 0 of multiplier Low byte set, subtract multiplicand from result High word.
5. Shift right result High word into result Low word/multiplier.
6. Shift right Low word/multiplier.
7. Decrement Loop counter.
8. If Loop counter not zero, go to Step 3.

Figure 5-1. “mpy16s” Flow Chart



5.2 Usage

The usage of “mpy16s” is as follows:

1. Load register variables “mp16sL”/”mp16sH” with multiplier Low and High byte, respectively.
2. Load register variables “mc16sH”/”mc16sH” with multiplicand Low and High byte, respectively.
3. Call “mpy16s”.
4. The 32-bit result is found in the 4-byte register variable “m16s3:m16s2:m16s1:m16s0”.

Observe that to minimize register usage, code and execution time, the multiplier and result Low byte share the same register.

5.3 Performance

Table 5-1. “mpy16s” Register Usage

Register	Input	Internal	Output
R16	“mc16sL” – Multiplicand Low Byte		
R17	“mc16sH” – Multiplicand High Byte		
R18	“mp16sL” – Multiplier Low Byte		“m16s0” – Result Byte 0
R19	“mp16sH” – Multiplier High Byte		“m16s1” – Result Byte 1
R20			“m16s2” – Result Byte 2
R21			“m16s2” – Result Byte 2
R22		“mcnt16s” – Loop Counter	

Table 5-2. “mpy16s” Performance Figures

Parameter	Value
Code Size (Words)	16 + return
Execution Time (Cycles)	218 + return
Register Usage	• Low Registers :None • High Registers :7 • Pointers :None
Interrupts Usage	None
Peripherals Usage	None