Homework 0: Intel x86 Assembly Language Review **Summer 2017**

Part 0:
This is an individual activity and is to be accompanied by screen images and a report.

Multiplication of two signed-binary numbers in 2's complement form can be performed by using binary addition and subtraction and a set of shift and rotate instructions. Below is the pseudo code for Booth's algorithm, which multiplies two signed-binary numbers in 2's complement. This algorithm is typically implemented by a hardware digital circuitry incorporated into an arithmetic logic unit. Additional information to confuse you can be found on the last two pages of this assignment.

Write the assembly code that will implement Booth's algorithm through utilization of resources internal to the processor (avoid using external memory due to significant latency introduced). Assemble, link, and debug your assembly program with MASM (or MASM32) and CodeView so that it executes in a user-visible window without errors.

Document the testing process to prove the functional correctness of your code.

You must clearly explain your code using appropriate documentation aids: readable layout, indentation, sparse commenting etc.

Procedure for multiplying two signed word-size binary numbers in 2's complement:

```
1. Initialize the counter to 16.
2. Place the Multiplicand in BX register.
3. Place the Multiplier in AX register.
4. Initialize DX to zero; DX:AX will hold the result
5. Clear the Carry Flag (CF) ; CLC instruction will do it!
6. Check Least Significant Bit (LSB) of Multiplier and CF.
7. If:
   a. LSB of Multiplier = CF, do nothing and go to step 8
   b. LSB of Multiplier=1 and CF=0, subtract Multiplicand from the
      high word of partial product in DX:AX
   c. LSB of Multiplier=0 and CF=1, add Multiplicand to the high word
      of DX:AX
8. Shift high word in DX, which holds a signed number (must preserve
      sign), right into low word of partial product DX:AX by one
      position (LSB of DX into MSB of AX) and the bit shifted out of
      the LSB of the low word should be the new value of CF.
9. If counter=0, go to step 11.
10. Go to step 6.
11. Exit.
```

Submit your source code, screen images of the assembly/linking of your code, and screen images of the code execution.

Part 1: Introduction to Assembler, Linker, Loader, and Locator Functionality

This is an individual activity and due with the project report.

You are asked to research and demonstrate your understanding regarding the function of an assembler, (compiler), a linker, a loader, and a locator. Specifically, starting with a source code file, which typically has assembly directives and instructions, explain what exactly happens to a program as it is processed by the assembler, linker, loader, and locator.

The discussion must be a sufficiently detailed technical analysis: imagine you will be lecturing on that topic so include relevant details you, as a student, would like to see to be able to learn effectively. In your discussion and presentation, you may consider including a simple assembly program and actually processing it through the assembler, linker, loader, and locator. Note that you will be addressing features and functionality of a "generic" assembler, linker, loader, locator and not necessarily MASM, Windows loader, or Paradigm Locate.

Booth's Multiplication Algorithm

The multiplication technique presented in Fig. 10-4 requires that the multiplier always be positive. Booth's algorithm treats a positive and negative multiplier uniformly and is more suitable for multiplication of signed-2's complement numbers. It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2^u to weight 2^v can be treated as $2^{u+1} - 2^v$. For example, if the multiplier is 001110 (+14), then $u = 3$ and $v = 1$ and $2^4 - 2^1 = 14$. As in all multiplication schemes, the algorithm requires the examination of the multiplier bits and shifting of the partial product. Prior to the shifting, the multiplicand may be either added to, or subtracted from, or may not change the partial product according to the following rules:

1. The multiplicand is subtracted from the partial product upon encountering the first 1 in a string of 1's in the multiplier.
2. The multiplicand is added to the partial product upon encountering the first 0 (provided that there was a previous 1) in a string of 0's in the multiplier.
3. The partial product does not change when the bit is identical to the previous multiplier bit.

The algorithm works for positive or negative multipliers in 2's complement representation. This is because a negative multiplier ends with a string of 1's and the last operation will be a subtraction of the appropriate weight. For example, a multiplier equal to −14 is represented as 110010 and treated as $-2^4 + 2^3 - 2^1 = -14$.

The hardware implementation of Booth's algorithm requires the register configuration of Fig. 10-1. As shown in the figure, Q_n designates the least significant bit of the multiplier in register QR. An extra flip-flop Q_{n+1} must be appended to QR to facilitate a double-bit inspection of the multiplier. The flow chart for Booth's algorithm in shown in Fig. 10-5. The AC and the appended bit Q_{n+1} are initially cleared to 0 and the sequence counter SC is set to a number equal to the number of bits in the multiplier. The two bits of the multiplier in Q_n and Q_{n+1} are inspected. If the two bits are equal to 10, it means that the first 1 in a string of 1's has been encountered. This requires a subtraction of the multiplicand from the partial product in the AC. If the two bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product. When the two bits are equal, the partial product does not change. The next step is to shift-right the partial product and the multiplier (including bit Q_{n+1}). This is an arithmetic shift-right (ashr) which requires no change in the AC sign bit. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the two numbers that are added have always opposite signs, a condition that excludes an overflow. The sequence counter is decremented and the computational loop is repeated n times.

A numerical example of Booth's algorithm is shown in Table 10-2 for $n = 5$. It shows the step-by-step multiplication of $(-9) \times (-13) = +117$. Note that the multiplier in QR is negative and the multiplicand in BR is also negative. The 10-bit product appears in the AC and QR and is positive. The final value of Q_{n+1} is the original sign bit of the multiplier and should not be taken as part of the product.

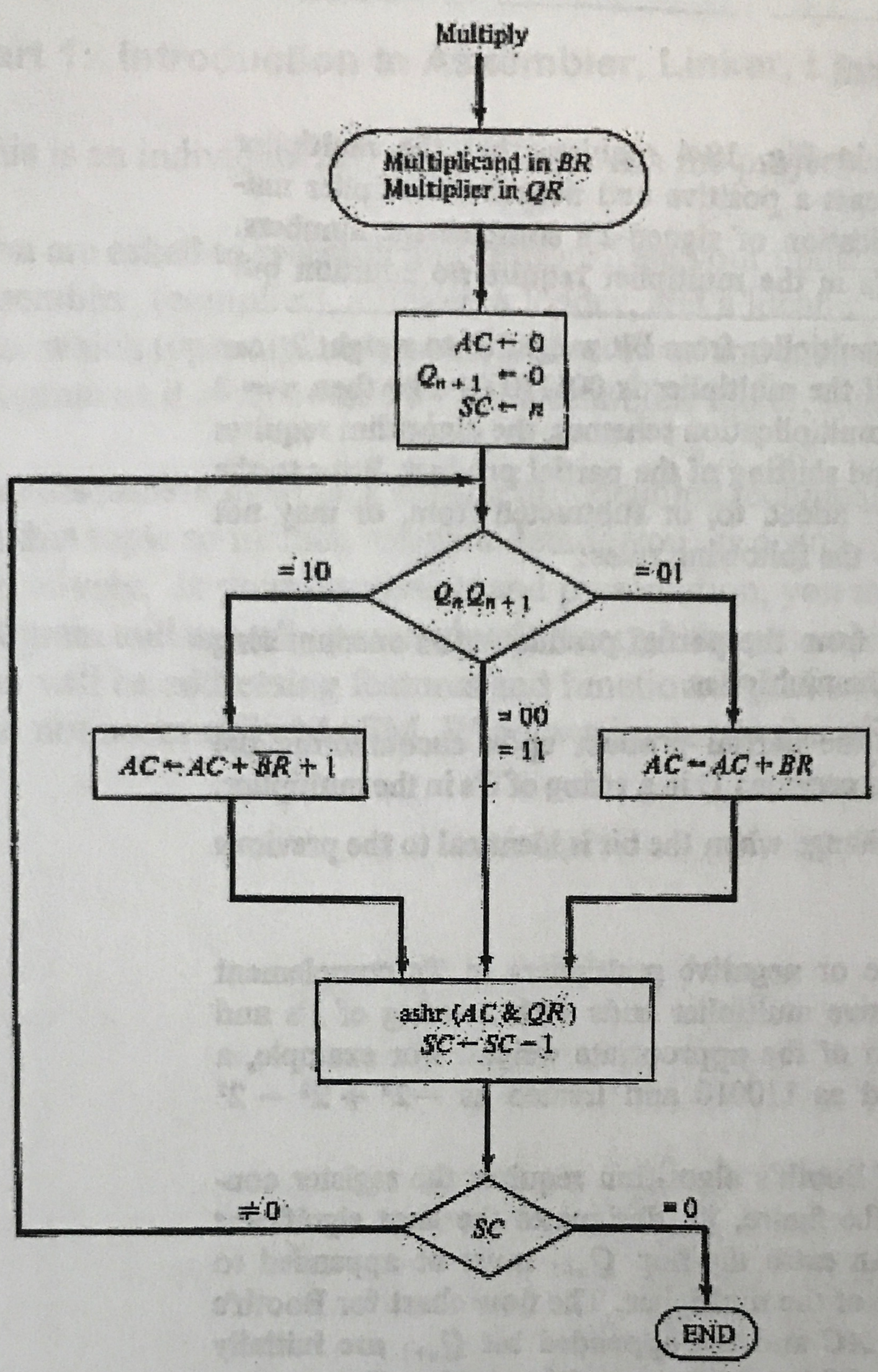

Fig. 10-5 Booth's algorithm for multiplication of signed-2's complement numbers.

TABLE 10-2 Example of Multiplication with Booth's Algorithm

$BR = 10111$
$\overline{BR} + 1 = 01001$

Q_nQ_{n+1}		AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	5
1 0	Subtract BR	01001			
		01001			
	ashr	00100	11001	1	4
1 1	ashr	00010	01100	1	3
0 1	Add BR	10111			
		11001			
	ashr	11100	10110	0	2
0 0	ashr	11110	01011	0	1
1 0	Subtract BR	01001			
		00111			
	ashr	00011	10101	1	0