CS/CoE 0447
Subtraction, Multiplication, and Division Examples

1. Perform three subtractions on 9-bit 2's complement binary numbers as follows:
   a) 206 – 54
   b) 68 – 56
   c) –51 – 76

For each of these subtractions, convert the numbers into 9-bit 2's complement, and for the numbers to subtract, or the negative numbers, convert them to their negative counterpart. Add those numbers and convert the resultant binary 2's complement number into decimal form. Show all your work.

a) 206 – 54

Conversion to 9-bit 2's Complement Notation from Decimal Notation	+206	−54
Step 1a. Convert positive part to binary. **Step 1b. Include preceding 0.** Step 1c. Preserve negative sign, if present.	**0**11001110	−**0**110110
Step 2. If negative sign is present, flip bits and add 1.	Not Applicable	1001001 +1
Step 3. Extend sign bit as needed.	011001110	**11**1001010

Decimal Notation	Two's Complement Notation	
206(decimal)	011001110	
−54(decimal)	+111001010	
	010011000	⇨ 152 (decimal)

b) 68 – 56

Conversion to 9-bit 2's Complement Notation from Decimal Notation	+68	-56
Step 1a. Convert positive part to binary. **Step 1b. Include preceding 0.** Step 1c. Preserve negative sign, if present.	**0**1000100	-**0**111000
Step 2. If negative sign is present, flip bits and add 1.	Not Applicable	1000111 +1
Step 3. Extend sign bit as needed.	**0**01000100	**11**1001000

Decimal Notation	Two's Complement Notation	
68(decimal)	001000100	
-56(decimal)	+111001000	
	000001100	⇨ 12 (decimal)

c) – 51 – 76

Conversion to 9-bit 2's Complement Notation from Decimal Notation	-51	-76
Step 1a. Convert positive part to binary. **Step 1b. Include preceding 0.** Step 1c. Preserve negative sign, if present.	-**0**110011	-**0**1001100
Step 2. If negative sign is present, flip bits and add 1.	1001100 +1	10110011 +1
Step 3. Extend sign bit as needed.	**11**1001101	**1**10110100

Decimal Notation	Two's Complement Notation	
-51(decimal)	111001101	
-76(decimal)	+110110100	
	110000001	⇨ -127 (decimal)

2. Show the steps for the multiplication of 10110110b and 10110100b (**unsigned**) using **Hardware Design 3** (http://www.cs.pitt.edu/~childers/CS0447/lectures/numbers3.pdf).
Here 10110110b is the multiplicand and 10110100b is the multiplier.

Fill up the columns:

Itera -tion	Multiplicand	Implementation 3	
		Step	Product(16-bit)
0	10110110b	initial values	0000 0000 1011 0100
1	10110110b	1: 0 -> no op 2: shift right (shift in c.o. bit*)	0000 0000 1011 0100 0000 0000 0101 1010
2	10110110b	1: 0 -> no op 2: shift right (shift in c.o. bit*)	0000 0000 0101 1010 0000 0000 0010 1101
3	10110110b	1: 1 -> product = product + multiplicand 2: shift right (shift in c.o. bit*)	1011 0110 0010 1101 0101 1011 0001 0110
4	10110110b	1: 0 -> no op 2: shift right (shift in c.o. bit*)	0101 1011 0001 0110 0010 1101 1000 1011
5	10110110b	1: 1 -> product = product + multiplicand 2: shift right (shift in c.o. bit*)	1110 0011 1000 1011 0111 0001 1100 0101
6	10110110b	1: 1 -> product = product + multiplicand 2: shift right (shift in c.o. bit*)	0010 0111 1100 0101 1001 0011 1110 0010
7	10110110b	1: 0 -> no op 2: shift right (shift in c.o. bit*)	1001 0011 1110 0010 0100 1001 1111 0001
8	10110110b	1: 1 -> product = product + multiplicand 2: shift right (shift in c.o. bit*)	1111 1111 1111 0001 0111 1111 1111 1000

*Shift in carry out bit of Step 1a in the flowchart on page 3 (slide 42) of (http://www.cs.pitt.edu/~childers/CS0447/lectures/numbers3.pdf). If Step 1a is bypassed, shift in 0.

3. Convert the following 8-bit binary numbers into Booth's encoding form:

01001011, 10011101, 01001101

Original:	01001011	10011101	01001101
Write Implied 0:	010010110	100111010	010011010
Encode each pair:	01001011 → -1	10011101 → -1	01001101 → -1
	01001011 → 0	10011101 → 1	01001101 → 1
	01001011 → 1	10011101 → -1	01001101 → -1
	01001011 → -1	10011101 → 0	01001101 → 0
	01001011 → 1	10011101 → 0	01001101 → 1
	01001011 → 0	10011101 → 1	01001101 → 0
	01001011 → -1	10011101 → 0	01001101 → -1
	01001011 → 1	10011101 → -1	01001101 → 1
Booth's encoded:	1-101-110-1	-10100-11-1	1-1010-11-1

4. Convert the following decimal numbers into 9-bit binary numbers in Booth's encoding form:

21, -217, -45

Original (dec.):	21	-217	-45
Orig.(9-bit 2's):	000010101	100100111	111010011
Write Implied 0:	0000101010	1001001110	1110100110
Encode each pair:	000010101 0 → -1	100100111 0 → -1	111010011 0 → -1
	000010101 0 → 1	100100111 0 → 0	111010011 0 → 0
	000010101 0 → -1	100100111 0 → 0	111010011 0 → 1
	000010101 0 → 1	100100111 0 → 1	111010011 0 → 0
	000010101 0 → -1	100100111 0 → 0	111010011 0 → -1
	000010101 0 → 1	100100111 0 → -1	111010011 0 → 1
	000010101 0 → 0	100100111 0 → 1	111010011 0 → -1
	000010101 0 → 0	100100111 0 → 0	111010011 0 → 0
	000010101 0 → 0	100100111 0 → -1	111010011 0 → 0
Booth's encoded:	0001-11-11-1	-101-10100-1	00-11-1010-1

Check by multiplying each Booth's digit by the weight of its place and then adding each product.	(0 × 256 = 0) +	(−1 × 256 = −256) +	(0 × 256 = 0) +
	(0 × 128 = 0) +	(0 × 128 = 0) +	(0 × 128 = 0) +
	(0 × 64 = 0) +	(1 × 64 = 64) +	(−1 × 64 = −64) +
	(1 × 32 = 32) +	(−1 × 32 = −32) +	(1 × 32 = 32) +
	(−1 × 16 = −16) +	(0 × 16 = 0) +	(−1 × 16 = −16) +
	(1 × 8 = 8) +	(1 × 8 = 8) +	(0 × 8 = 0) +
	(−1 × 4 = −4) +	(0 × 4 = 0) +	(1 × 4 = 4) +
	(1 × 2 = 2) +	(0 × 2 = 0) +	(0 × 2 = 0) +
	(−1 × 1 = −1) =	(−1 × 1 = −1) =	(−1 × 1 = −1) =
	21	−217	−45

5. Show the steps for the multiplication of 10110110b and 10110100b (**signed**) using **Booth's algorithm** (http://www.cs.pitt.edu/~childers/CS0447/lectures/numbers3.pdf).
Here 10110110b (-74decimal) is the multiplicand and 10110100b (-76decimal) is the multiplier.

Fill up the columns:

Iteration	Multiplicand	Booth's Algorithm	
		Step	Product(17-bit)
0	10110110b (-74decimal)	initial values	0000 0000 1011 0100
1	10110110b	1: 00 -> no op 2: Arithmetic shift right	0000 0000 0101 1010
2	10110110b	1: 00 -> no op 2: Arithmetic shift right	0000 0000 0010 1101
3	10110110b	1: 10 -> product = product – multiplicand 2: Arithmetic shift right	+0100 1010 (74 dec.) 0100 1010 0010 1101 0 0010 0101 0001 0110 1
4	10110110b	1: 01 -> product = product + multiplicand 2: Arithmetic shift right	+1011 0110 (-74 dec.) 1101 1011 0001 0110 1 1110 1101 1000 1011 0
5	10110110b	1: 10 -> product = product – multiplicand 2: Arithmetic shift right	+0100 1010 (74 dec.) 0011 0111 1000 1011 0 0001 1011 1100 0101 1
6	10110110b	1: 11 -> no op 2: Arithmetic shift right	0001 1011 1100 0101 1 0000 1101 1110 0010 1
7	10110110b	1: 01 -> product = product + multiplicand 2: Arithmetic shift right	+1011 0110 (-74 dec.) 1100 0011 1110 0010 1 1110 0001 1111 0001 0
8	10110110b	1: 10 -> product = product – multiplicand 2: Arithmetic shift right	+0100 1010 (74 dec.) 0010 1011 1111 0001 0 **0001 0101 1111 1000** 1

(-74decimal) times (-76decimal) = (5624 decimal = **0001 0101 1111 1000** binary)

6. Show the steps for the multiplication of 00110111b and 10010001b (signed) using Booth's algorithm (available here: http://www.cs.pitt.edu/~childers/CS0447/lectures/numbers3.pdf). Here 00110111b is the multiplicand and 10010001b is the multiplier. Fill up the columns:

Iteration	Multiplicand	Booth's Algorithm	
		Step	Product(17-bit)
0	00110111b (55dec)	initial values	0000 0000 1001 0001 0
1	00110111b	1: 10 -> product = product – multiplicand 2: Arithmetic shift right	+1100 1001 (-55 dec.) 1100 1001 1001 0001 0 1110 0100 1100 1000 1
2	00110111b	1: 01 -> product = product + multiplicand 2: Arithmetic shift right	+0011 0111 (55 dec.) 0001 1011 1100 1000 1 0000 1101 1110 0100 0
3	00110111b	1: 00 -> no op 2: Arithmetic shift right	0000 0110 1111 0010 0
4	00110111b	1: 00 -> no op 2: Arithmetic shift right	0000 0011 0111 1001 0
5	00110111b	1: 10 -> product = product – multiplicand 2: Arithmetic shift right	+1100 1001 (-55 dec.) 1100 1100 0111 1001 0 1110 0110 0011 1100 1
6	00110111b	1: 01 -> product = product + multiplicand 2: Arithmetic shift right	+0011 0111 (55 dec.) 0001 1101 0011 1100 1 0000 1110 1001 1110 0
7	00110111b	1: 00 -> no op 2: Arithmetic shift right	0000 0111 0100 1111 0
8	00110111b	1: 10 -> product = product – multiplicand 2: Arithmetic shift right	+1100 1001 (-55 dec.) 1101 0000 0100 1111 0 **1110 1000 0010 0111** 1

(55decimal) times (-111decimal) = -6105 decimal = 1110 1000 0010 0111

7 In a table similar to the following one, show the steps for computing 00101101b (the dividend) divided by 0101b (the divisor, both numbers are unsigned) using **non-restoring division**. Non-restoring division is described online at: http://www.cs.pitt.edu/~childers/CS0447/lectures/division-floats.pdf. An example of doing non-restoring division is also shown.

Itera-tion	Divisor (4 bits)	Step (description)	Remainder Register (8 bits)	Dividend Register (8 bits)
0	0101 (5 dec)	initial values shift left by 1	0000 0000 0000 0000	0010 1101 (45 dec) 0101 1010
1		remainder = remainder – divisor (remainder<0) ⇒ shift left; r0=0	+(-5dec) +1111 1011 1111 1011 1111 0110	 0101 1010 1011 010**0**
2		remainder = remainder + divisor (remainder<0) ⇒ shift left; r0=0	+(5dec) +0000 0101 1111 1011 1111 0111	 1011 0100 0110 100**0**
3		remainder = remainder + divisor (remainder<0) ⇒ shift left; r0=0	+(5dec) +0000 0101 1111 1100 1111 1000	 0110 1000 1101 000**0**
4		remainder = remainder + divisor (remainder<0) ⇒ shift left; r0=0	+(5dec) +0000 0101 1111 1101 1111 1011	 1101 0000 1010 000**0**
5		remainder = remainder + divisor (remainder≮0) ⇒ shift left; r0=1	+(5dec) +0000 0101 0000 0000 0000 0001	 1010 0000 0100 000**1**
6		remainder = remainder – divisor (remainder<0) ⇒ shift left; r0=0	+(-5dec) +1111 1011 1111 1100 1111 1000	 0100 0001 1000 001**0**
7		remainder = remainder + divisor (remainder<0) ⇒ shift left; r0=0	+(5dec) +0000 0101 1111 1101 1111 1011	 1000 0010 0000 010**0**
8		remainder = remainder + divisor (remainder≮0) ⇒ shift left; r0=1	+(-5dec) +1111 1011 0000 0000 0000 0000	 0000 0100 0000 100**1**
Done		shift “left half of 16-bit remainder | dividend” register right by 1	**0000 0000**	0000 1001

45 / 5 = 9 (decimal) = 1001 (binary) 0