1. If $5$ contains $0xAF000A5D$ and $6$ contained $0x6$, what hexadecimal value would be in $4$ after each of the following?
   a) sll $4$, $5$, 3
   b) sllv $4$, $5$, $6$
   c) sra $4$, $5$, 3
   d) ror $4$, $5$, $6$

2. If $5$ contains $0xA5D$ and $6$ contained $0x63C$, what hexadecimal value would be in $4$ after each of the following?
   a) and $4$, $5$, $6$
   b) ori $4$, $5$, $0xBF$
   c) xor $4$, $5$, $6$
   d) nor $4$, $5$, $6$
   e) not $4$, $5$

3. Sometimes you want to manipulate individual bits in a “string of bits”. For example, you can represent a set of letters using a bit-string. Each bit in the bit-string is associated with a letter: bit position 0 with ‘A’, bit position 1 with ‘B’, ..., bit position 25 with ‘Z’. Bit-string bits are set to ‘1’ to indicate that their corresponding letters are in the set. For example, the set { ‘A’, ‘B’, ‘D’, ‘Y’ } would be represented as:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 0 0 0 0 0</td>
<td>0 1 0</td>
<td>0 1 0 1 1</td>
</tr>
</tbody>
</table>

To determine if a specific ASCII character, say ‘C’ ($67_{10}$) is in the set, you would need to build a “mask” containing a single “1” in bit position 2.
   a) What sequence of instructions could we use to build the mask needed for ‘C’ in $3$?
   b) If a bit-string set of letters is in register $5$, then what instructions can be used to check if the character ‘C’ (using the mask in $3$) is in the set contained in $5$?
4. Use Booth's algorithm to calculate the 8-bit product of $0110_2 \times 1011_2$.

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>Product register</th>
<th>Multiplier register</th>
<th>Previous bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0</td>
<td>0 0 0 0</td>
<td>1 0 1 1</td>
<td>0</td>
</tr>
<tr>
<td>- Multiplicand</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. You are to implement a function called "Multiply" that is passed two 32-bit signed integers and returns their 64-bit product. This function should use Booth's algorithm only, i.e., sure that you DO NOT use any form of the multiply (e.g., mul, mult, multu, etc.) assembly language instruction.

a) Since you want to follow the MIPS register conventions, what registers should be used to pass the two integers into Multiply?

b) What registers should be used to return the 64-bit product?

c) If Multiply does not call any subprograms(/procedures/functions/methods), what type of registers should we try to use?

d) How many times will you need to loop if we are checking one bit of the multiplier (and the previous bit) each iteration of the loop?
$a0, \text{Multiplicand} \quad \begin{array}{c}
\begin{array}{c}
0 \\
1 \\
1 \\
0 \\
\end{array}
\end{array}
\begin{array}{c}
\text{Product register} \\
\begin{array}{c}
x \\
\end{array}
\end{array}
\begin{array}{c}
\text{Multiplier register} \\
\begin{array}{c}
y \\
\end{array}
\end{array}
\begin{array}{c}
\text{Previous bit} \\
\begin{array}{c}
0 \\
0 \\
0 \\
0 \\
\end{array}
\end{array}
\end{array}

For (- Multiplicand), we'll subtract $a0

\begin{array}{c}
\begin{array}{c}
0 \\
0 \\
0 \\
\end{array}
0 \\
\end{array}
\begin{array}{c}
\begin{array}{c}
x \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
y \\
\end{array}
\end{array}
\begin{array}{c}
\begin{array}{c}
0 \\
0 \\
0 \\
\end{array}
\end{array}
\end{array}

\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
x \\
\end{array}
\end{array}
0 \\
0 \\
0 \\
\end{array}
\begin{array}{c}
{t6}
\end{array}

\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
x \\
\end{array}
\end{array}
\end{array}
\begin{array}{c}
{t5}
\end{array}

\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
x \\
\end{array}
\end{array}
\end{array}
\begin{array}{c}
{t6}
\end{array}

\text{e) How would you set the previous bit in } {t4} \text{ to zero?}

\text{f) How would you get the LSB of the multiplier into } {t5}?

\text{g) How would you implement the following if-statements?}
\quad \text{if LSB of multiplier (in } {t5} \text{) } \neq \text{ previous bit (in } {t4} \text{) then}
\quad \quad \text{if previous bit (in } {t4} \text{) } = 1 \text{ then}
\quad \quad \quad \quad {v1} = {v1} + {a0}
\quad \quad \quad \text{else}
\quad \quad \quad \quad {v1} = {v1} - {a0}
\quad \quad \text{end if}
\quad \text{end if}

\text{h) How would you get the LSB of the } {v1} \text{ into } {t6}?

\text{i) How would you shift the LSB of } {t6} \text{ to be the MSB of } {t6}?

\text{j) Now, how would you set the MSB of the } {v0} \text{ (multiplier reg.) to the MSB of } {t6}?

\text{k) How would you shift } {v1} \text{ to the right one bit position?
6) Complete the following Multiply code:

# Programmer: Mark Fienup
# Booth's algorithm implementation using shifting & logic instructions
.data
multiplicand: .word 103
multiplier: .word -25
result_upper: .word 0
result_lower: .word 0

.text
.globl main
main:
lw $a0, multiplicand # load multiplicand into $a0
lw $a1, multiplier # load multiplier into $a1
jal multiply # call multiply subprogram
sw $v0, result_lower # store 64-bit result
sw $v1, result_upper
li $v0, 10 # exit system call
syscall
endMain:
multiply: