Name:_____

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:

	unused	'Z'	'Y'	'X'	. . .	'E'	'D'	'C'	'B'	'A'
{ 'A', 'B', 'D', 'Y' } is	0 0 0 0 0 0	0	1	0		0	1	0	1	1
bit position:		25	24	23		4	3	2	1	0

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?

Name:_____

4. Use Booth's algorithm to calculate the 8-bit product of 0110_2 x 1011_2.

Multiplicand	Product register	Multiplier register	Previous bit
0 1 1 0	0 0 0 0	1 0 1 1	0
− Multiplicand 1 0 1 0			

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.

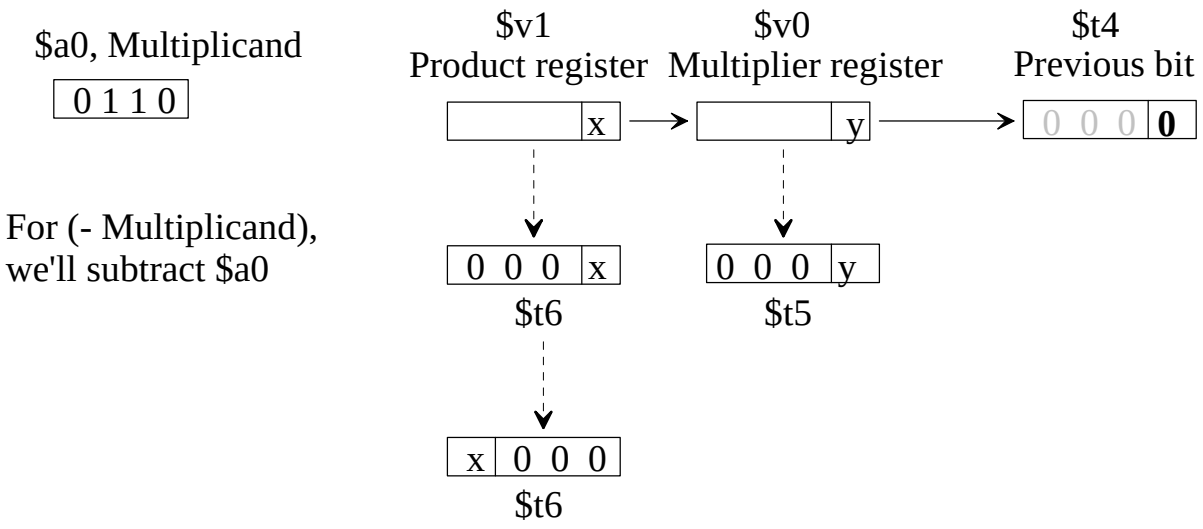
a) Since you want to follow the MIPS register convensions, what registers should be used to pass the two integers into Multiply?

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?

Name:_____



e) How would you set the previous bit in $t4 to zero?

f) How would you get the LSB of the multiplier into $t5?

g) How would you implement the following if-statements?

```
if  LSB of multiplier (in $t5) != previous bit (in $t4) then
        if previous bit (in $t4) == 1 then
                $v1 = $v1 + $a0
        else
                $v1 = $v1 - $a0
        end if
end if
```

h) How would you get the LSB of the $v1 into $t6?

i) How would you shift the LSB of $t6 to be the MSB of $t6?

j) Now, how would you set the MSB of the $v0 (multiplier reg.) to the MSB of $t6?

k) How would you shift $v1 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:
```

Name:_____

Below is the MIPS assembly language code for the multiply function:

```
multiply:
        # Input:  $a0 multiplicand, $a1 multiplier
        # Output: $v0 least-significant 32-bits of product, $v1 most-significant 32-bits of product

        move   $v0, $a1              # move multipler to $v0
        li     $v1, 0                # clear most-sign. 32 bits of product

        move   $t4, $zero            # initial previous LSB of multiplier
for1:
        li     $t2, 0                # for $t2 := 0 to 31 do
forTest1:
        bgt    $t2, 31, endFor1

        andi   $t5, $v0, 1           # $t5 = LSB of multiplier
        if1:                         # if $t5 != $t4 then
                beq $t5, $t4, endIf1
                if2:                 # if $t4 (previous bit) = 1 (and LSB of multiplier = 0)
                        beq $t4, $zero, else2
                then2:
                        add  $v1, $v1, $a0  # add multiplicand
                        j endIf2
                else2:
                        sub $v1, $v1, $a0   # subtract multiplicand
                endIf2:
        endIf1:
        andi   $t4, $v0, 1           # save LSB of multiplier to previous bit ($t4)

        srl    $v0, $v0, 1           # shift result right one bit
       # shift LSB of $v1 into MSB of $v0
        andi   $t6, $v1, 1           # get LSB of $v1
        ror    $t6, $t6, 1           # move LSB to MSB position
        or     $v0, $v0, $t6         # set MSB of $v0 with LSB of $v1

        sra    $v1, $v1, 1           # shift $v1 right one bit (sign extend)

        addi   $t2, $t2, 1
        j      forTest1
endFor1:
        jr     $ra                   # return to calling routine
endMultiply:
```