1. Use Booth's algorithm to calculate the 8-bit product of $0110_2 \times 1011_2$.

Multiplicand

0 1 1 0

Product register	Multiplier register	Previous bit
0 0 0 0	1 0 1 1	0

− Multiplicand

1 0 1 0

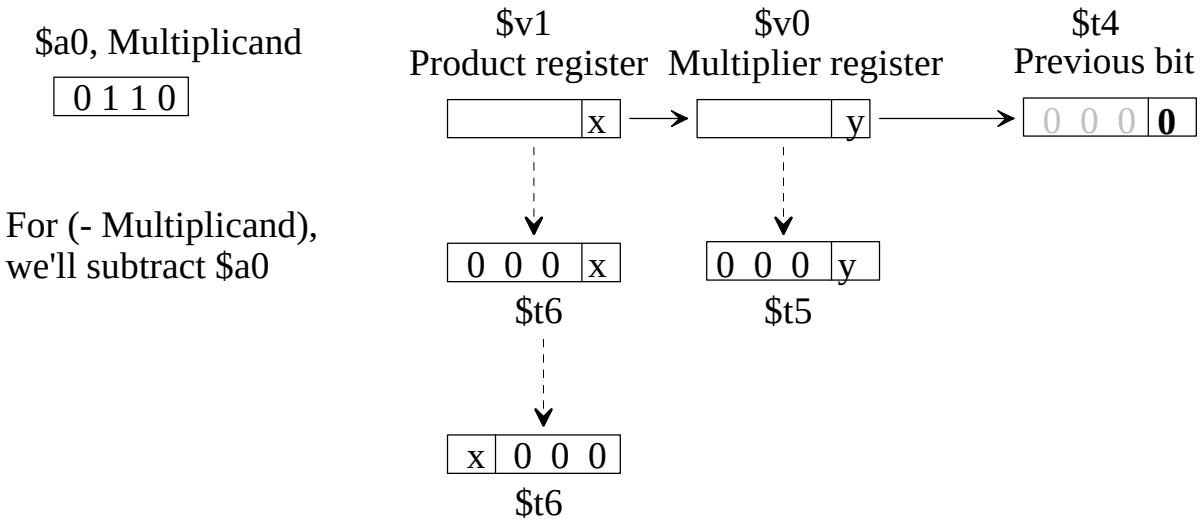
2. You are to implement a function called "Multiply" that is passed two 32-bit signed integers and returns their 64-bit product. This function should use Booth's algorithm only, i.e., sure that you DO NOT use any form of the multiply (e.g., mul, mult, multu, etc.) assembly language instruction.

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?



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?

2) 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:
```