**HW#7 (CSC390) Due: 03/23/2018 by 11:00PM on Blackbaord**

**SOLUTION**

**Q1.** (Read chapter#3 (pages 178-182) of your text book before you solve the following questions)

1. MIPS assembler generates an exception error when overflow occurs in mathematical operation of signed numbers. Write a MIPS assembly language program that will check overflows during the element by element addition between the array elements (signed numbers) of two vectors A and B and store the results in C vector. If overflow occurs the code will store 0xFFFFFFFFhex in the corresponding memory location of the C vector, otherwise store the actual addition results.

Consider the following vectors A and B and store the results C vectors in your coding:

A = [1000000000, 2000000000, 2000000000, -1000000000, -2000000000];

B= [1000000000, -1000000000, 1000000000, -1000000000, -1000000000];

C=[. . . …. …. … ];

**SOLUTION:**

#Detecting Overflow in signed addition

# HW#7(Q#2\_a)

.data

A: .word 1000000000, 2000000000, 2000000000, -1000000000, -2000000000

B: .word 1000000000, -1000000000, 1000000000, -1000000000, -1000000000

C: .space 20 #allocting spce for storing results

.text

la $s0, A # Load Address of A

la $s1, B # Load Address of B

la $s2, C # Load Address of C

li $s3, 0 # Starting index of i

li $t5, 5 # Loop bound

li $s4,0xffffffff # Overflow Indicator

loop:

lw $t1, 0($s0) # Load A[i]

lw $t2, 0($s1) # Load B[i]

addu $t0, $t1, $t2 # $t0 = sum, but don’t trap

xor $t3, $t1, $t2 # Check if signs differ

slt $t3, $t3, $zero # $t3 = 1 if signs differ

bne $t3, $zero, No\_overflow # $t1, $t2 signs ? so no overflow

xor $t3, $t0, $t1 # signs =; sign of sum match too?

# $t3 negative if sum sign different

slt $t3, $t3, $zero # $t3 = 1 if sum sign different

bne $t3, $zero, Overflow # All 3 signs ?; go to overflow

No\_overflow:

sw $t0, 0($s2)

addi $s0, $s0, 4 # Go to A[i+1]

addi $s1, $s1, 4 # Go to B[i+1]

addi $s2, $s2, 4 # Go to C[i+1]

addi $s3, $s3, 1 # Increment index variable

bne $s3, $t5, loop # Compare with Loop Bound

j End

Overflow:

sw $s4, 0($s2) # Store 0xffffffff in C

addi $s0, $s0, 4 # Go to A[i+1]

addi $s1, $s1, 4 # Go to B[i+1]

addi $s2, $s2, 4 # Go to C[i+1]

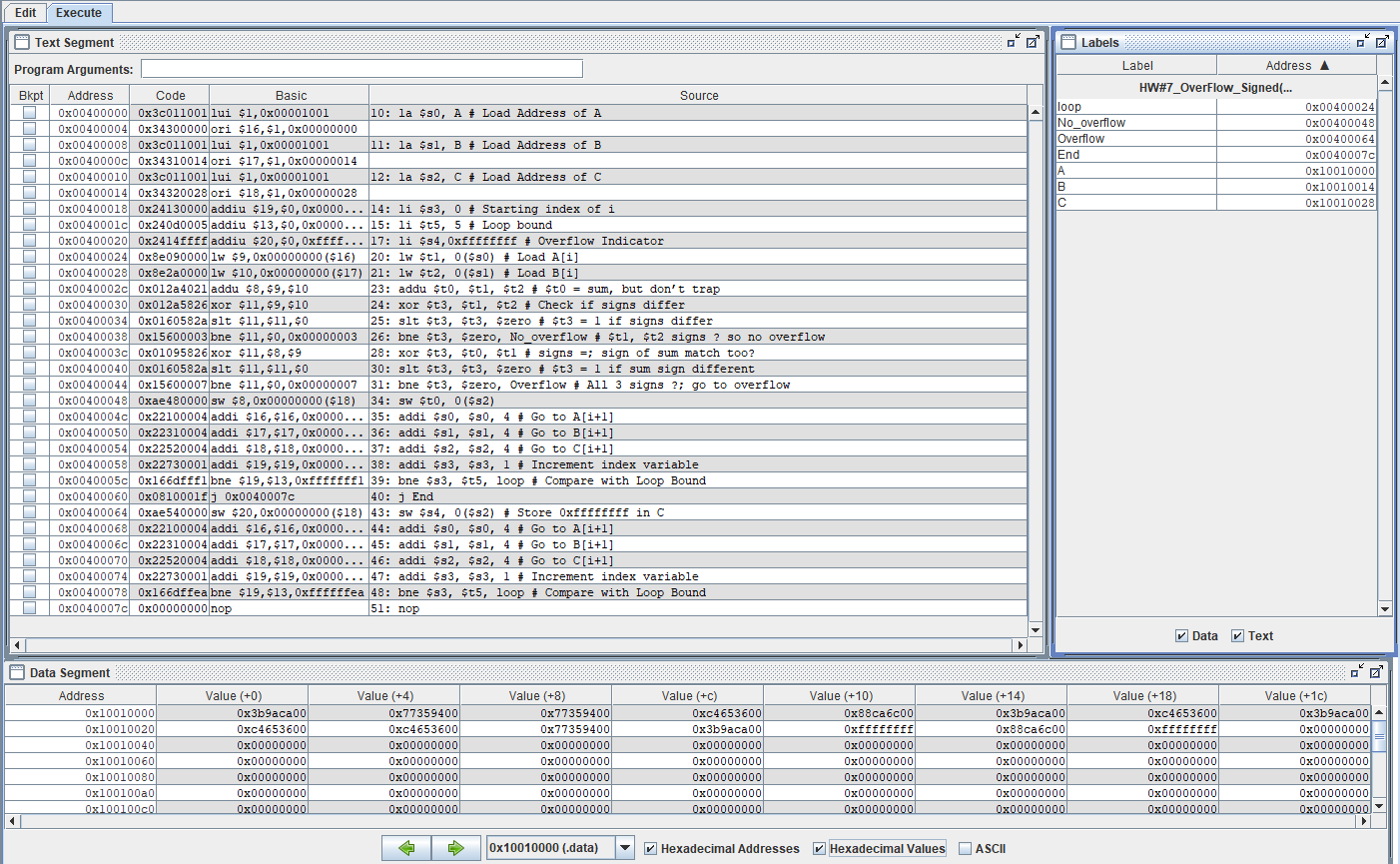
addi $s3, $s3, 1 # Increment index variable

bne $s3, $t5, loop # Compare with Loop Bound

End:

nop

**Output Print Screen:**



1. Write a MIPS assembly language program that will check overflows during the element by element addition between the array elements (unsigned numbers) of two vectors A and B and store the results in C vector. If overflow occurs the code will store 0x00000000 in the corresponding memory location of the C vector, otherwise store the actual addition results.

Consider the following vectors A and B and store the results C vectors in your coding:

A = [1000000000, 2000000000, 2000000000, 1000000000, 2000000000];

B = [3000000000, 3000000000, 1000000000, 4000000000, 2000000000];

C=[. . . …. …. … ];

**SOLUTION:**

#Detecting Overflow in unsigned addition

# HW#7\_Q1(b)

.data

A: .word 1000000000, 2000000000, 2000000000, 1000000000, 2000000000

B: .word 3000000000, 3000000000, 1000000000, 4000000000, 2000000000

C: .space 20 #allocting spce for storing results

.text

la $s0, A # Load Address of A

la $s1, B # Load Address of B

la $s2, C # Load Address of C

li $s3, 0 # Starting index of i

li $t5, 4 # Loop bound

li $s4,0x00000000 # Overflow Indicator

loop:

lw $t1, 0($s0) # Load A[i]

lw $t2, 0($s1) # Load B[i]

addu $t0, $t1, $t2 # $t0 = sum

nor $t3, $t1, $zero # $t3 = NOT $t1

# (2’s comp – 1: 232 – $t1 – 1)

sltu $t3, $t3, $t2 # (232 – $t1 – 1) < $t2

# ? 232 – 1 < $t1 + $t2

bne $t3,$zero,Overflow # if(232–1<$t1+$t2) goto overflow

sw $t0, 0($s2)

addi $s0, $s0, 4 # Go to A[i+1]

addi $s1, $s1, 4 # Go to B[i+1]

addi $s2, $s2, 4 # Go to C[i+1]

addi $s3, $s3, 1 # Increment index variable

bne $s3, $t5, loop # Compare with Loop Bound

j End

Overflow:

sw $s4, 0($s2) # Store 0xffffffff in C

addi $s0, $s0, 4 # Go to A[i+1]

addi $s1, $s1, 4 # Go to B[i+1]

addi $s2, $s2, 4 # Go to C[i+1]

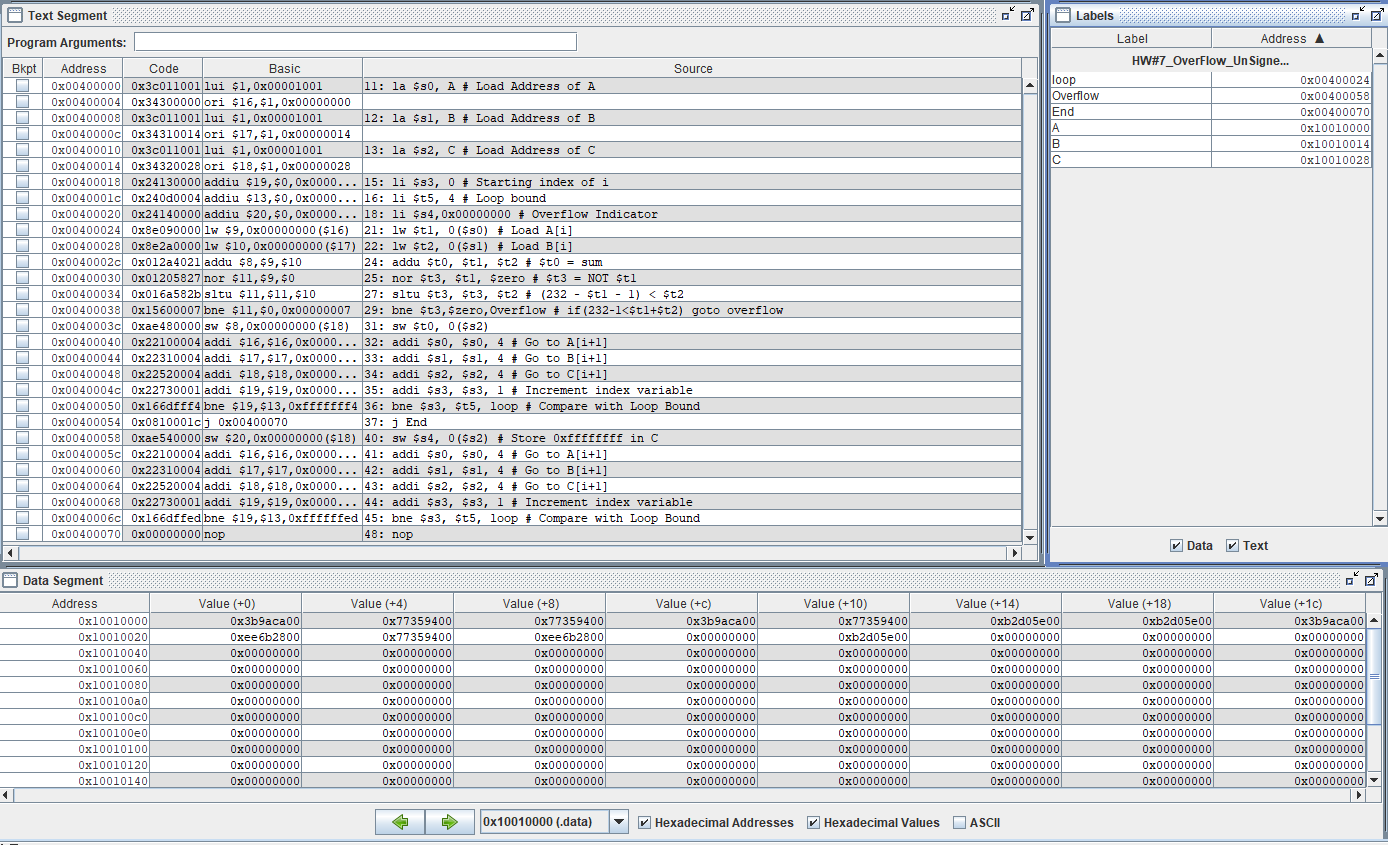
addi $s3, $s3, 1 # Increment index variable

bne $s3, $t5, loop # Compare with Loop Bound

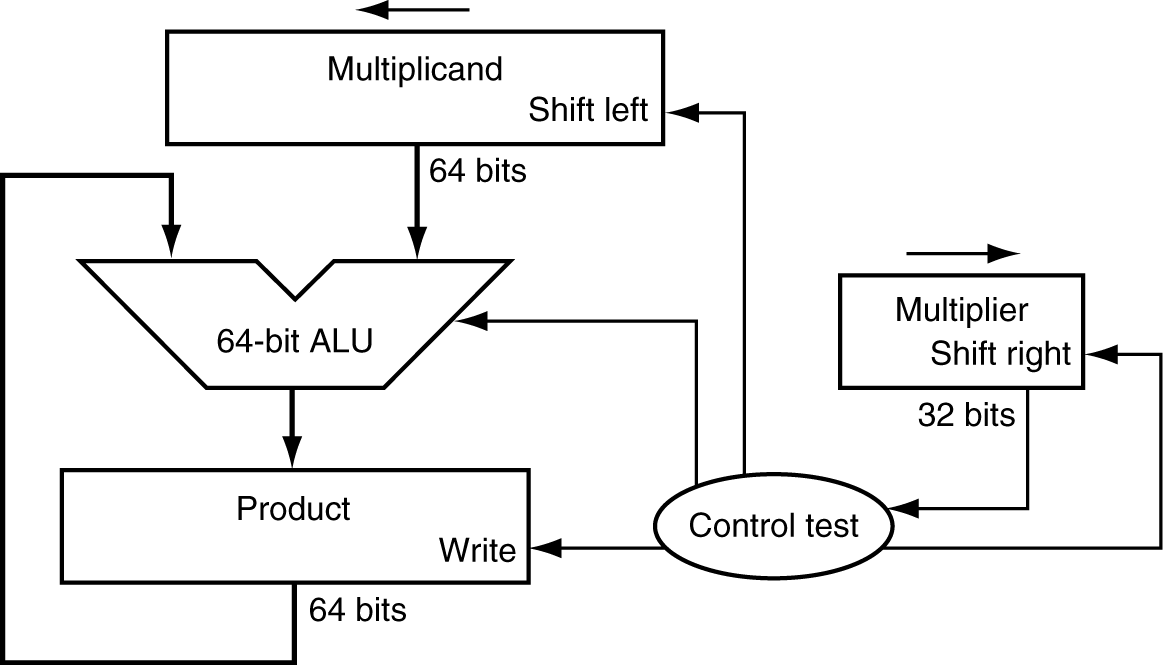
End:

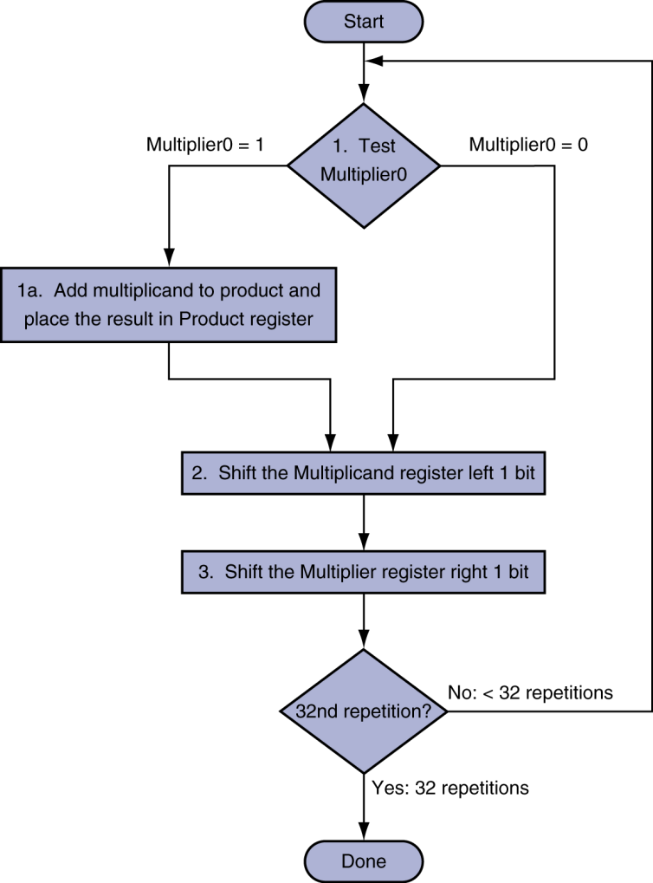
nop

Output Print Screen



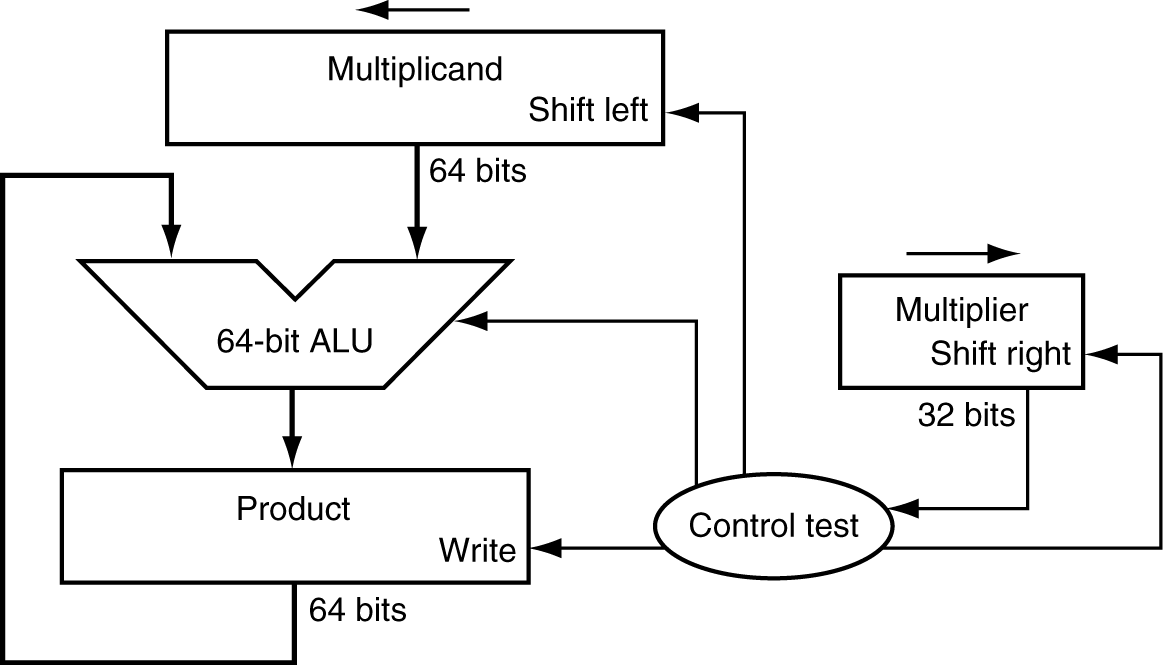
**Q2.** The following Circuit performs the multiplication of two 32-bit unsigned binary numbers and produces a 64-bit result. Figure 1 shows the three steps multiplication algorithm that the circuit performs to produce the result. Suppose, you are assigned to design a 5-bit multiplier circuit which will perform the multiplication of the two 5-bit unsigned numbers A=(11011)2 and B=(10011)2 and produce a 10-bit result. Draw the necessary hardware and show the results produced in each iteration of your algorithm. Hints: see figure 3.6 (page 187) of our text book.





Figure

**SOLUTION:**

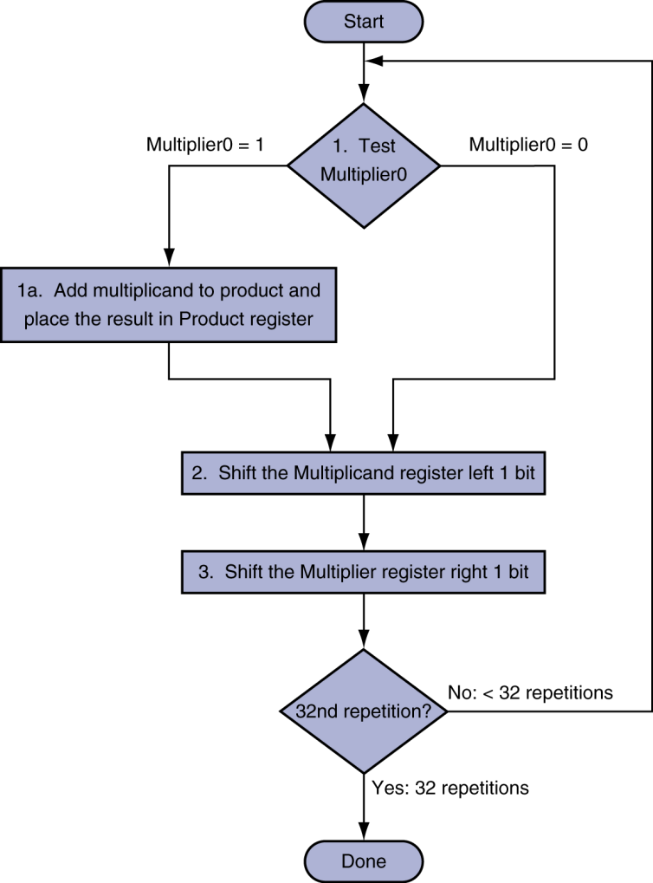


10 bits

5 bits

10 bits

10 bits



5 repetitions

5 repetitions

Figure

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Iteration** | **Step** | **Multiplier** | **Multiplicand** | **Product** |
| **0** | **Initial Value** | **10011** | **0000011011** | **0000000000** |
| **1** | 1a: 1=> prod=prod+Mcand | **10011** | **0000011011** | **0000011011** |
| 2: Shift left Multiplicand | **10011** | **0000110110** | **0000011011** |
| 3: Shift right Multiplier | **01001** | **0000110110** | **0000011011** |
| **2** | 1a: 1=> prod=prod+Mcand | **01001** | **0000110110** | **0001010001** |
| 2: Shift left Multiplicand | **01001** | **0001101100** | **0001010001** |
| 3: Shift right Multiplier | **00100** | **0001101100** | **0001010001** |
| **3** | 1: 0 => no operation | **00100** | **0001101100** | **0001010001** |
| 2: Shift left Multiplicand | **00100** | **0011011000** | **0001010001** |
| 3: Shift right Multiplier | **00010** | **0011011000** | **0001010001** |
| **4** | 1: 0 => no operation | **00010** | **0011011000** | **0001010001** |
| 2: Shift left Multiplicand | **00010** | **0110110000** | **0001010001** |
| 3: Shift right Multiplier | **00001** | **0110110000** | **0001010001** |
| **5** | 1a: 1=> prod=prod+Mcand | **00001** | **0110110000** | **1000000001** |
| 2: Shift left Multiplicand | **00001** | **1101100000** | **1000000001** |
| 3: Shift right Multiplier | **00000** | **1101100000** | **1000000001** |

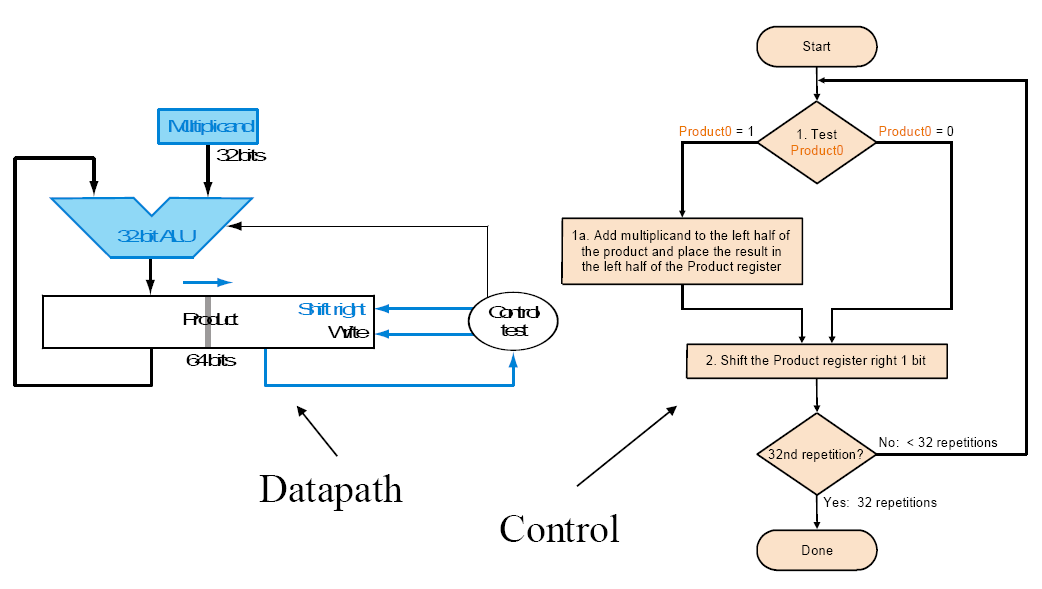
**11011 🡪 27**

**10011 🡪19**

**27x19 = 513 🡺 1000000001**

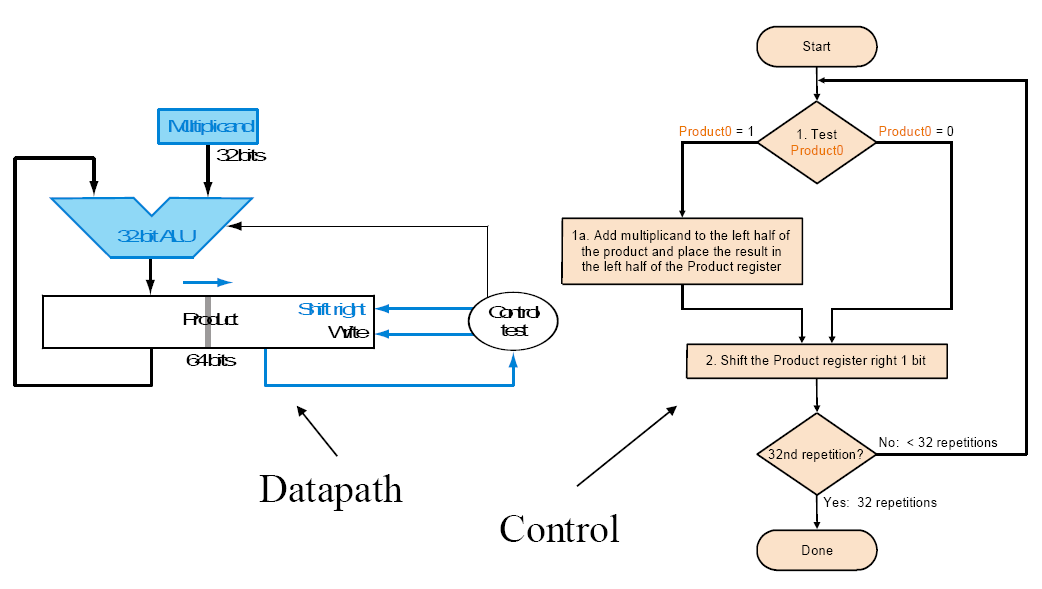
**Q3 (a).** The optimized version of the hardware and its corresponding algorithm is shown in figure 2. Note that the right half of the product register is now initialized with the multiplier.

Suppose, you are assigned to design a 5-bit multiplier circuit which will perform the multiplication of the two 5-bit signed numbers A=(11011)2 and B=(01011)2 and produce a 10-bit result. Draw the necessary hardware and show the results produced in each iteration of your algorithm. Hints: see slide 13 of the Chapter\_03.ppt posted on the Blackboard.



Figure

**SOLUTION: Next Page**



5 repetitions

5 repetitions

10 bits

5 bits

5 bits

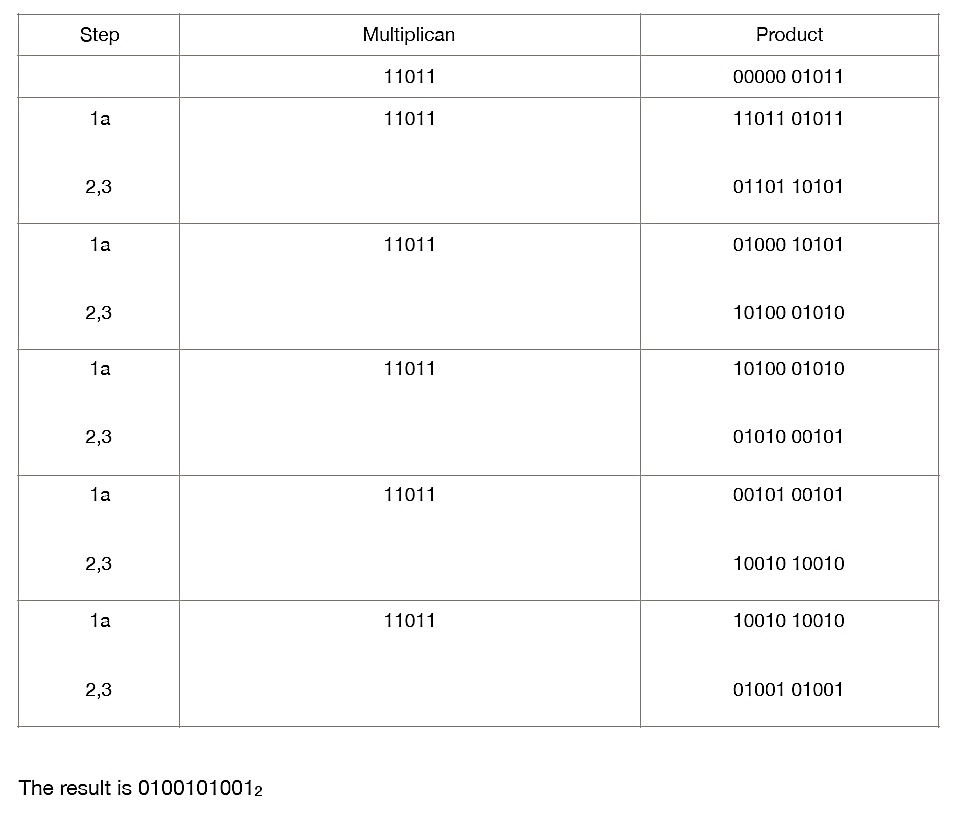
Figure

|  |  |  |
| --- | --- | --- |
| Steps | Multiplicand | Product |
| Initial Value | 11011 | 0000001011 |
| 1a | 11011 | 1101101011 |
| 2 , 3 | 11011 | 1110110101 |
| 1a | 11011 | 1100010101 |
| 2 , 3 | 11011 | 1110001010 |
| 1 | 11011 | 1110001010 |
| 2, 3 | 11011 | 1111000101 |
| 1a | 11011 | 1100100101 |
| 2 , 3 | 11011 | 1110010010 |
| 1 | 11011 | 1110010010 |
| 2, 3 | 11011 | **1111001001** |

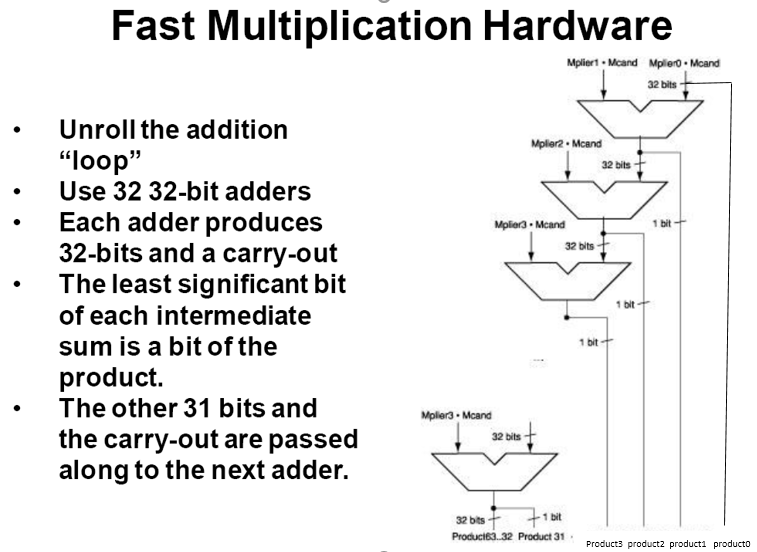
**11011 🡪 -5 ; 01011 🡪 11 ; -5x11 = -55 🡺 1111001001**

**Q3 (b).** Verify that the above can also be used to performthe multiplication of the two 5-bit **unsigned** numbers A=(11011)2 and B=(01011)2 and produce a 10-bit result. Show the results produced in each iteration of your algorithm

**SOLUTION:**



**Q4.** To improve the execution speed of the above multiplier circuits, a Fast Multiplication Hardware, as shown in figure 3, is designed using (n-1) adder circuits, which perform the multiplication of two n-bit numbers and produces the 2n-bit result. Suppose, you are assigned to design a 5-bit Fast Multiplication Hardware circuit which will perform the multiplication of the two 5-bit unsigned numbers A=(11011)2 and B=(10011)2 and produce a 10-bit result. Draw the necessary hardware and show the results produced in step of the multiplication process.



**31**

Figure

Figure 4