Assignment #3
88-448: Digital Computer Architecture

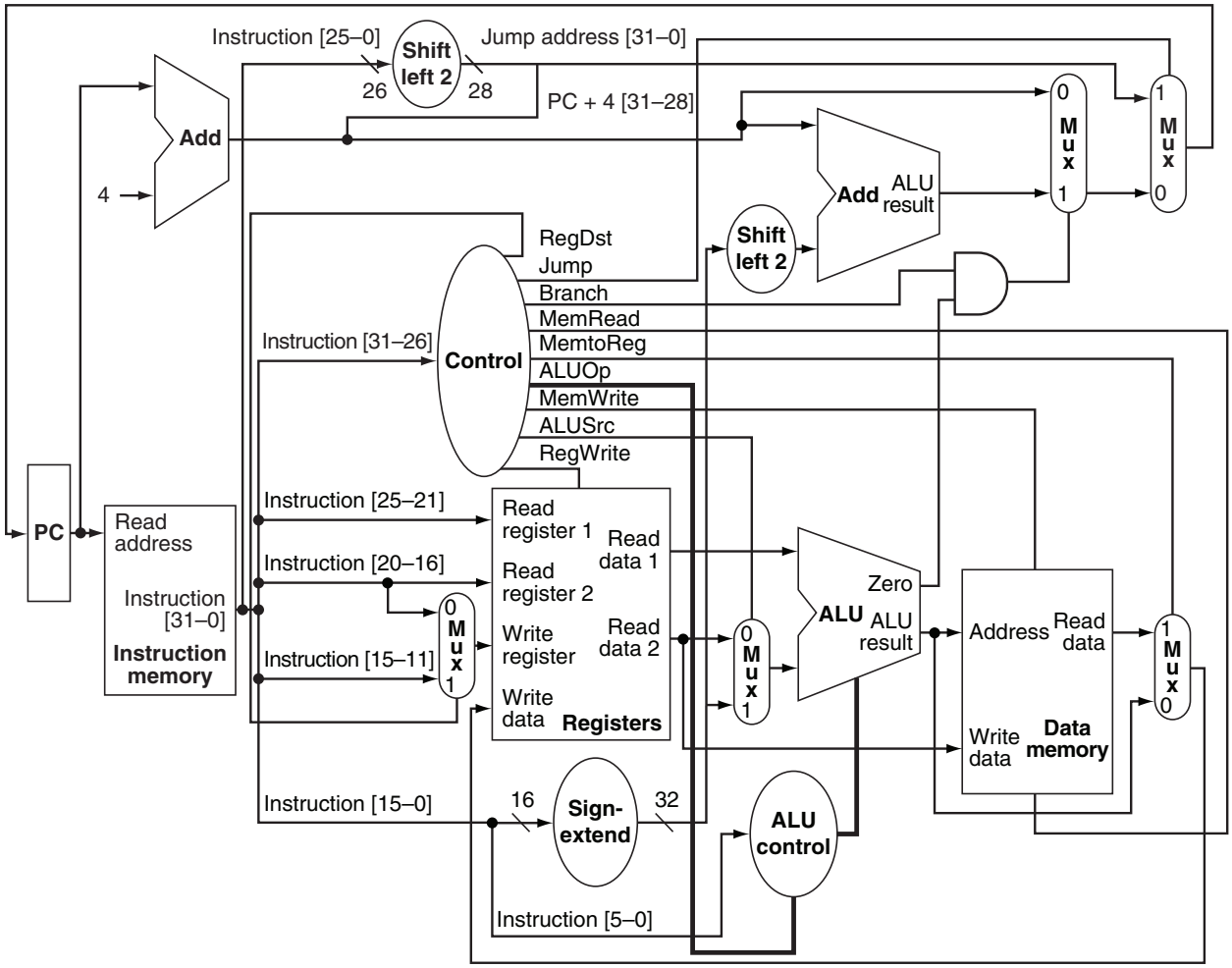
Use the Quartus 13.0sp1 tool ("web" version found in the undergraduate lab as well as www.altera.com) to implement the MIPS design found below (Fig 4.24 in 4th Rev Ed. of text). This is a single clock cycle implementation, no pipelining.

To simplify your implementation, avoid high levels of abstraction and keep your HDL coding to only a single or very few processes. You may use either VHDL or Verilog. Memory devices can simply be treated as arrays of registers for simplicity in the interfacing; **make their depth no more than 32 words**.

Use the MIPS reference card (found on the course web site) as a reference for the opcode and function numbers. The testing "code" is provided on the next page in assembly as well as HDL bit encoding.

This is not a group or team assignment; your work should be unique to you.

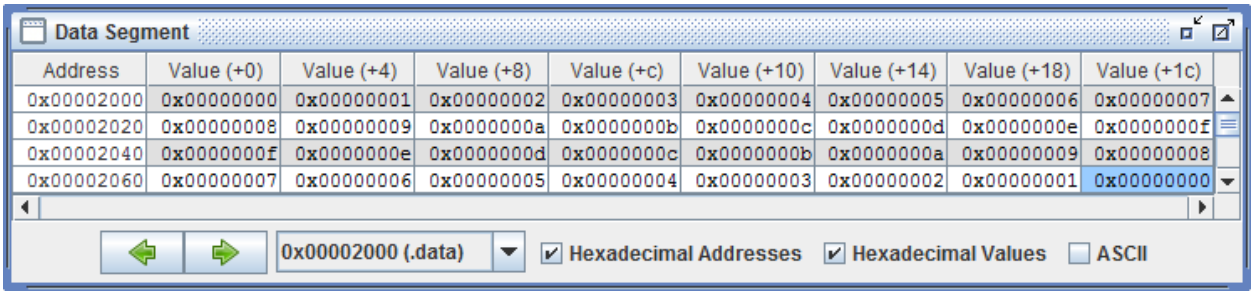
Please implement only the following opcodes: addiu, addu, beq, bne, j, lw, sltiu, sltu, sw, subu, syscall (syscall only used to stop your simulation).



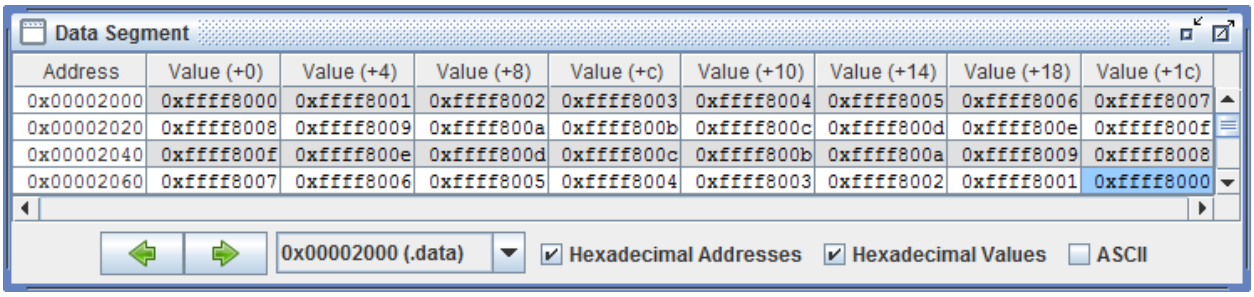
Testing code:

```
start:  addiu   $t0,$0,0        # 0
        addiu   $t1,$0,1        # 1
        addiu   $t2,$0,0        # 2
        addiu   $t3,$0,4        # 3
        addiu   $t4,$0,0x2000   # 4
loop1:  sw      $t2,0($t4)      # 5
        addu    $t2,$t2,$t1     # 6
        addiu   $t4,$t4,4       # 7
        sltiu   $at,$t2,16      # 8
        bne     $at,$0,loop1    # 9
        addiu   $t4,$t4,8       # 10
loop2:  subu    $t2,$t2,$t1     # 11
        sw      $t2,-8($t4)     # 12
        addu    $t4,$t4,$t3     # 13
        beq     $t2,$0,loop3    # 14
        j       loop2           # 15
loop3:  addiu   $t4,$0,0x1ff8   # 16
        addiu   $t3,$0,32       # 17
loop4:  lw      $t5,8($t4)      # 18
        addiu   $t5,$t5,-32768  # 19
        sw      $t5,8($t4)      # 20
        addu    $t2,$t2,$t1     # 21
        addiu   $t4,$t4,4       # 22
        sltu    $at,$t2,$t3     # 23
        bne     $at,$0,loop4    # 24
        addiu   $v0,$v0,10      # 25
        syscall                 # 26
```

The first loop fills memory from 0x2000 to 0x202f with values from 0x0 to 0xf (increasing). The second loop fills memory from 0x2040 to 0x206f with values from 0xf to 0x0 (decreasing). The result is shown below:



The last loop takes the values from 0x2000 to 0x206f and adds “-32768” to them to produce a large negative number. The sign extension should work such that the upper 16 bits of the results are all 1’s. The result is shown below:



In all three loops, memory is accessed using different offsets; this is to test that portion of your design. The memory area of 0x2000 to 0x206f will write to RAM from 0x0 to 0x1f. This address is used such that the code can be tested also in MARS since it will not allow you to write to program memory since it uses a shared memory architecture.

The last two lines perform a syscall to stop the running of the program. Your code should only stop on the syscall opcode; it doesn't need to evaluate the registers to determine the correct action, just stop.

VHDL	Verilog
ROM[0]<="00100100000010000000000000000000"; ROM[1]<="00100100000010010000000000000001"; ROM[2]<="00100100000010100000000000000000"; ROM[3]<="00100100000010110000000000000100"; ROM[4]<="00100100000011000010000000000000"; ROM[5]<="10101101100010100000000000000000"; ROM[6]<="00000001010010010101000000100001"; ROM[7]<="00100101100011000000000000000100"; ROM[8]<="00101101010000010000000000010000"; ROM[9]<="00010100001000001111111111111011"; ROM[10]<="00100101100011000000000000001000"; ROM[11]<="00000001010010010101000000100011"; ROM[12]<="10101101100010101111111111111000"; ROM[13]<="00000001100010110110000000100001"; ROM[14]<="00010001010000000000000000000001"; ROM[15]<="00001000000000000000000000001011"; ROM[16]<="00100100000011000001111111111000"; ROM[17]<="00100100000010110000000000100000"; ROM[18]<="10001101100011010000000000001000"; ROM[19]<="00100101101011011000000000000000"; ROM[20]<="10101101100011010000000000001000"; ROM[21]<="00000001010010010101000000100001"; ROM[22]<="00100101100011000000000000000100"; ROM[23]<="00000001010010110000100000101011"; ROM[24]<="00010100001000001111111111111001"; ROM[25]<="00100100010000100000000000001010"; ROM[26]<="00000000000000000000000000001100";	ROM[0]<=32'b00100100000010000000000000000000; ROM[1]<=32'b00100100000010010000000000000001; ROM[2]<=32'b00100100000010100000000000000000; ROM[3]<=32'b00100100000010110000000000000100; ROM[4]<=32'b00100100000011000010000000000000; ROM[5]<=32'b10101101100010100000000000000000; ROM[6]<=32'b00000001010010010101000000100001; ROM[7]<=32'b00100101100011000000000000000100; ROM[8]<=32'b00101101010000010000000000010000; ROM[9]<=32'b00010100001000001111111111111011; ROM[10]<=32'b00100101100011000000000000001000; ROM[11]<=32'b00000001010010010101000000100011; ROM[12]<=32'b10101101100010101111111111111000; ROM[13]<=32'b00000001100010110110000000100001; ROM[14]<=32'b00010001010000000000000000000001; ROM[15]<=32'b00001000000000000000000000001011; ROM[16]<=32'b00100100000011000001111111111000; ROM[17]<=32'b00100100000010110000000000100000; ROM[18]<=32'b10001101100011010000000000001000; ROM[19]<=32'b00100101101011011000000000000000; ROM[20]<=32'b10101101100011010000000000001000; ROM[21]<=32'b00000001010010010101000000100001; ROM[22]<=32'b00100101100011000000000000000100; ROM[23]<=32'b00000001010010110000100000101011; ROM[24]<=32'b00010100001000001111111111111001; ROM[25]<=32'b00100100010000100000000000001010; ROM[26]<=32'b00000000000000000000000000001100;