

Assignment #3

88-448: Digital Computer Architecture

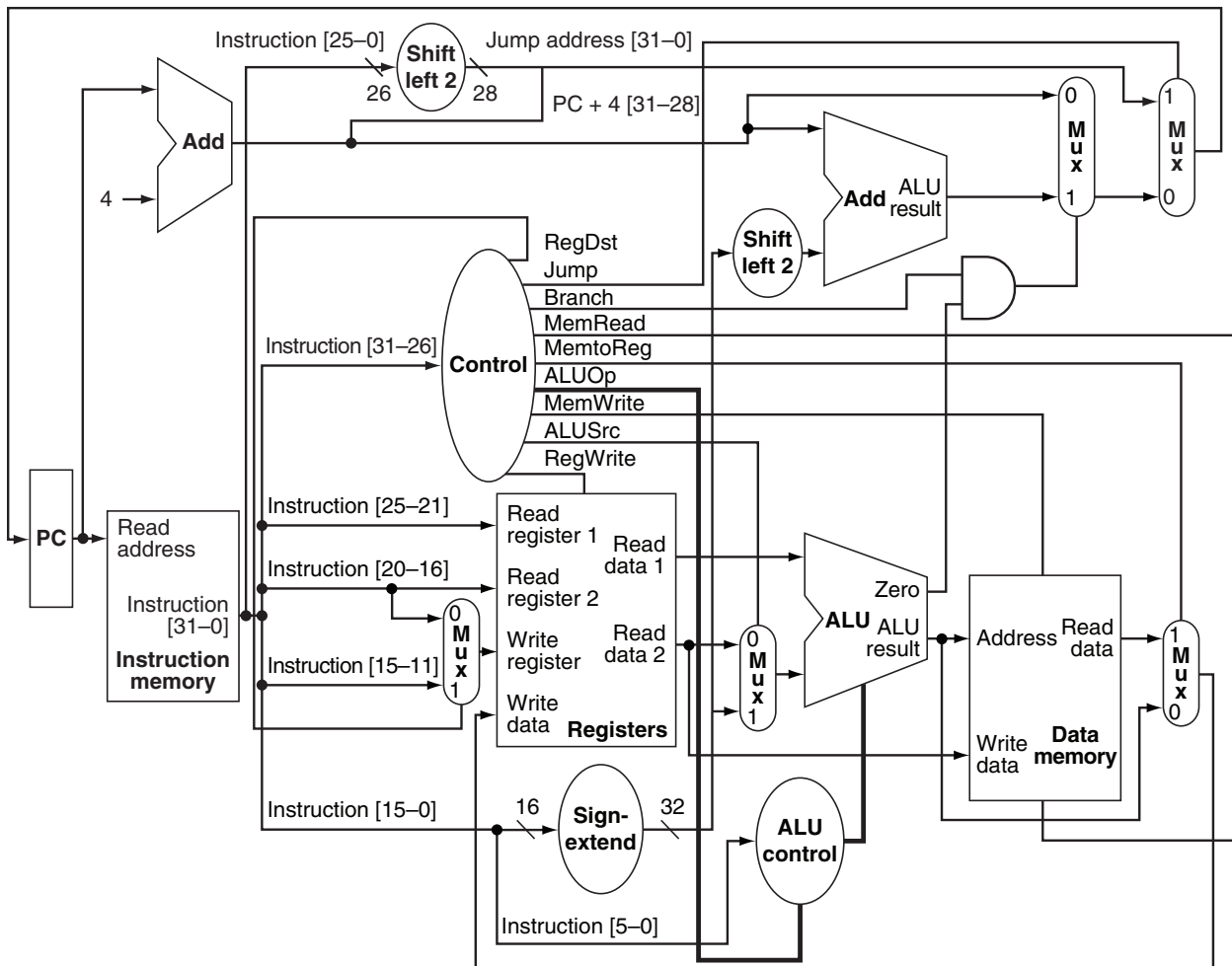
Use the Quartus 13.0sp1 tool ("web" version found in the undergraduate lab as well as www.altera.com) to implement the MIPS design found below (Fig 4.24 in 4th Rev Ed. of text). This is a single clock cycle implementation, no pipelining.

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.

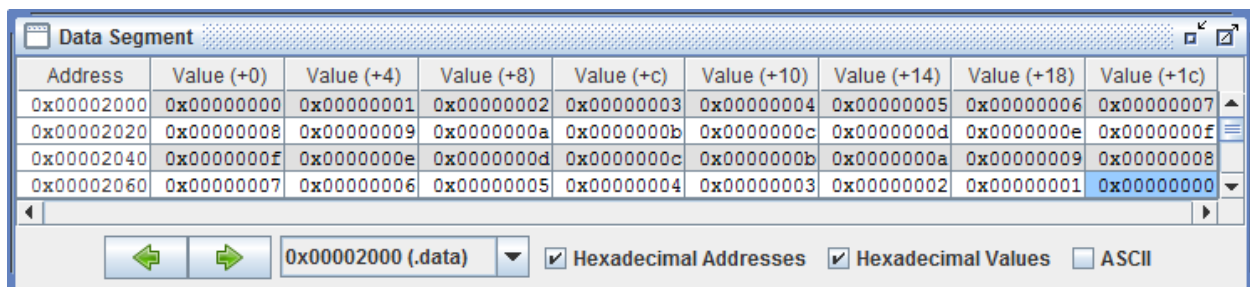
Please implement only the following opcodes: addiu, addu, beq, bne, j, lw, sltiu, sltu, sw, subu, syscall (only used to stop your simulation).



Testing code:

```
start:  addiu    $t0,$0,0
        addiu    $t1,$0,1
        addiu    $t2,$0,0
        addiu    $t3,$0,4
        addiu    $t4,$0,0x2000
loop1:  sw       $t2,0($t4)
        addu     $t2,$t2,$t1
        addiu    $t4,$t4,4
        sltiu    $at,$t2,16
        bne     $at,$0,loop1
        addiu    $t4,$t4,8
loop2:  subu     $t2,$t2,$t1
        sw       $t2,-8($t4)
        addu     $t4,$t4,$t3
        beq     $t2,$0,loop3
        j       loop2
loop3:  addiu    $t4,$0,0x1ff8
        addiu    $t3,$0,32
loop4:  lw       $t5,8($t4)
        addiu    $t5,$t5,-32768
        sw       $t5,8($t4)
        addu     $t2,$t2,$t1
        addiu    $t4,$t4,4
        sltu     $at,$t2,$t3
        bne     $at,$0,loop4
        addiu    $v0,$v0,10
        syscall
```

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:



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0x00000000	0x00000001	0x00000002	0x00000003	0x00000004	0x00000005	0x00000006	0x00000007
0x00002020	0x00000008	0x00000009	0x0000000a	0x0000000b	0x0000000c	0x0000000d	0x0000000e	0x0000000f
0x00002040	0x0000000f	0x0000000e	0x0000000d	0x0000000c	0x0000000b	0x0000000a	0x00000009	0x00000008
0x00002060	0x00000007	0x00000006	0x00000005	0x00000004	0x00000003	0x00000002	0x00000001	0x00000000

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:

