Assignment #4

ELEC4480: Digital Computer Architecture

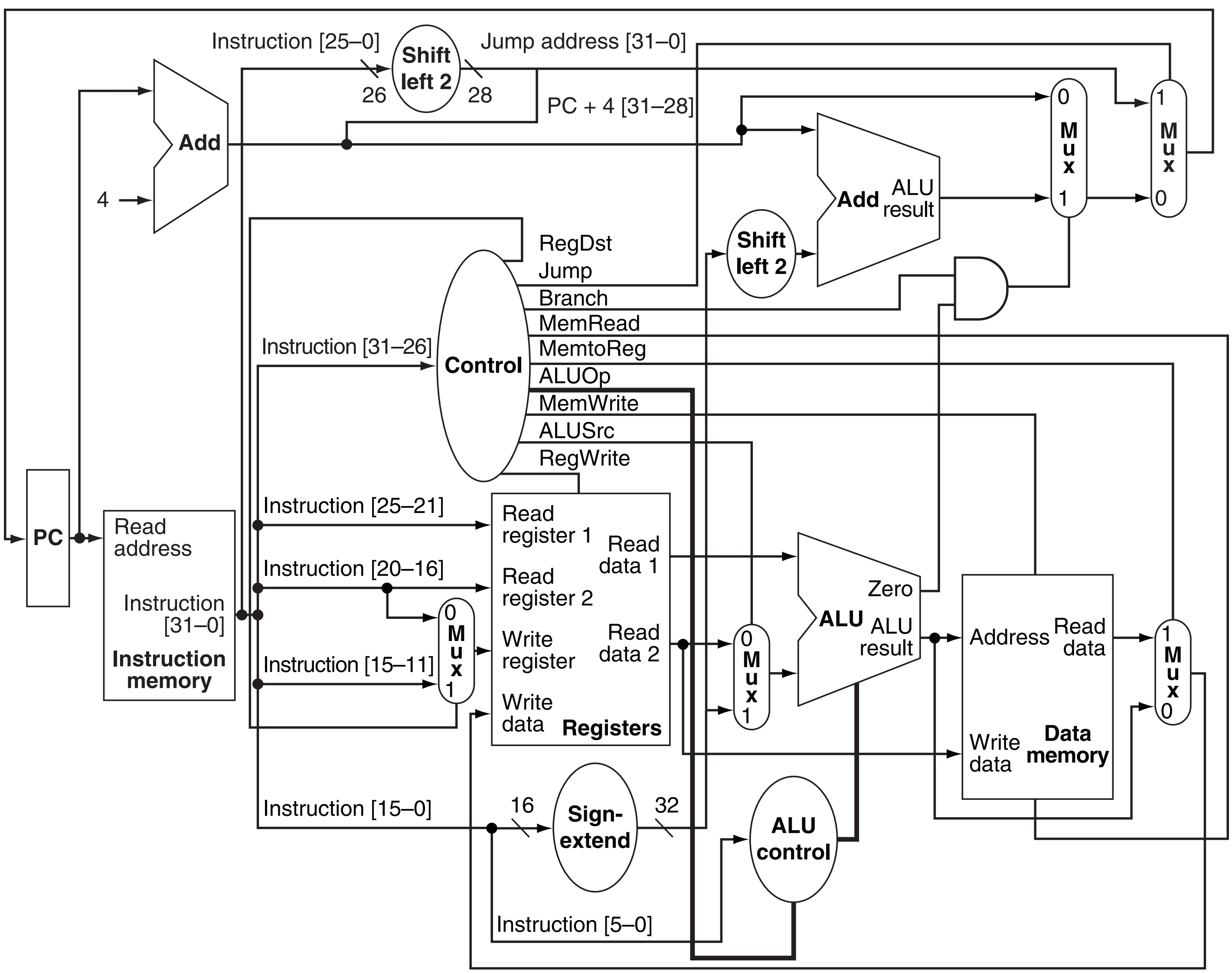
Use www.edaplayground.com or the Quartus tool to implement the MIPS architecture found below (Fig 4.24 in 4th Rev Ed. of text). Your design doesn't have to be a 1:1 implementation of this figure, however it needs to be able to run MIPS code that follows the opcodes on the reference sheet. This is a single 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 Verilog or VHDL; name your files "SID.v" or "SID.vhd" and upload it to Blackboard when completed. 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 MIPS 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. All previous years' submissions will be compared to yours, so please do not try to use someone else's code.

Please implement only the following opcodes: addiu, addu, beq, bne, j, lw, sltiu, sltu, sw, subu, syscall (syscall is only used to stop your simulation; it is an R-type instruction with an opcode of 0, and function of 12).



MIPS Testing code:

```
start:  addiu   $t3,$0,4        # 0
        addiu   $t2,$0,0        # 1
        addiu   $t1,$0,1        # 2
        addiu   $t0,$0,0        # 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,$0,10       # 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:

Data Segment

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

0x00002000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

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:

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00002000	0xffff8000	0xffff8001	0xffff8002	0xffff8003	0xffff8004	0xffff8005	0xffff8006	0xffff8007
0x00002020	0xffff8008	0xffff8009	0xffff800a	0xffff800b	0xffff800c	0xffff800d	0xffff800e	0xffff800f
0x00002040	0xffff800f	0xffff800e	0xffff800d	0xffff800c	0xffff800b	0xffff800a	0xffff8009	0xffff8008
0x00002060	0xffff8007	0xffff8006	0xffff8005	0xffff8004	0xffff8003	0xffff8002	0xffff8001	0xffff8000

0x00002000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

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.

Your top level module/entity should be named "assign4" and have two inputs (CK, and RESET), and four outputs (DONE, OUTADDR[5], OUTDATA[32], OUTVALID) in that order. CK is the input clock (your logic should operate on the rising edge of the clock), and RESET is asserted HIGH for only one clock (you should initialize PC to 0 when this happens). When OUTVALID is HIGH, the instructor's test-bench will display both OUTADDR and OUTDATA; these are the memory address (word address, or byte address divided by 4) being written to and the value being written (during the SW instruction) there respectively. They should match the memory write patterns shown in the figures above except those are byte referenced (you should divide them by 4). When your design encounters the syscall instruction, it should assert the DONE signal to tell the test-bench the program is complete; it will then stop the simulation.

The testing program has been assembled and translated into COPY/PASTE code which you can use in your module.

VHDL MIPS Program	Verilog MIPS Program
ROM(0)<="00100100000010110000000000000100";	ROM[0]<=32'b00100100000010110000000000000100;
ROM(1)<="00100100000010100000000000000000";	ROM[1]<=32'b00100100000010100000000000000000;
ROM(2)<="00100100000010010000000000000001";	ROM[2]<=32'b00100100000010010000000000000001;
ROM(3)<="00100100000010000000000000000000";	ROM[3]<=32'b00100100000010000000000000000000;
ROM(4)<="00100100000011000010000000000000";	ROM[4]<=32'b00100100000011000010000000000000;
ROM(5)<="10101101100010100000000000000000";	ROM[5]<=32'b10101101100010100000000000000000;
ROM(6)<="00000001010010010101000000100001";	ROM[6]<=32'b00000001010010010101000000100001;
ROM(7)<="00100101100011000000000000000100";	ROM[7]<=32'b00100101100011000000000000000100;
ROM(8)<="00101101010000010000000000010000";	ROM[8]<=32'b00101101010000010000000000010000;
ROM(9)<="00010100001000001111111111111011";	ROM[9]<=32'b00010100001000001111111111111011;
ROM(10)<="00100101100011000000000000001000";	ROM[10]<=32'b00100101100011000000000000001000;
ROM(11)<="00000001010010010101000000100011";	ROM[11]<=32'b00000001010010010101000000100011;
ROM(12)<="10101101100010101111111111111000";	ROM[12]<=32'b10101101100010101111111111111000;
ROM(13)<="00000001100010110110000000100001";	ROM[13]<=32'b00000001100010110110000000100001;
ROM(14)<="00010001010000000000000000000001";	ROM[14]<=32'b00010001010000000000000000000001;
ROM(15)<="00001000000000000000000000001011";	ROM[15]<=32'b00001000000000000000000000001011;
ROM(16)<="00100100000011000001111111111000";	ROM[16]<=32'b00100100000011000001111111111000;
ROM(17)<="00100100000010110000000000100000";	ROM[17]<=32'b00100100000010110000000000100000;
ROM(18)<="10001101100011010000000000001000";	ROM[18]<=32'b10001101100011010000000000001000;
ROM(19)<="00100101101011011000000000000000";	ROM[19]<=32'b00100101101011011000000000000000;
ROM(20)<="10101101100011010000000000001000";	ROM[20]<=32'b10101101100011010000000000001000;
ROM(21)<="00000001010010010101000000100001";	ROM[21]<=32'b00000001010010010101000000100001;
ROM(22)<="00100101100011000000000000000100";	ROM[22]<=32'b00100101100011000000000000000100;
ROM(23)<="00000001010010110000100000101011";	ROM[23]<=32'b00000001010010110000100000101011;
ROM(24)<="00010100001000001111111111111001";	ROM[24]<=32'b00010100001000001111111111111001;
ROM(25)<="00100100000000100000000000001010";	ROM[25]<=32'b00100100000000100000000000001010;
ROM(26)<="00000000000000000000000000001100";	ROM[26]<=32'b00000000000000000000000000001100;

The Verilog and VHDL modules are listed below so that you can create the proper module/entity interface.

<u>Verilog module declaration:</u>

```
module assign4(CK, RESET, DONE, OUTADDR, OUTDATA, OUTVALID);
input  CK;
input  RESET;
output DONE;
reg    DONE;
output [4:0] OUTADDR;
reg    [4:0] OUTADDR;
output [31:0] OUTDATA;
reg    [31:0] OUTDATA;
output OUTVALID;
reg    OUTVALID;

...

end module
```

VHDL module declaration:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity assign4 is
port (CK       : in  std_logic;
      RESET    : in  std_logic;
      DONE     : out std_logic;
      OUTADDR  : out std_logic_vector(4 downto 0);
      OUTDATA  : out std_logic_vector(31 downto 0);
      OUTVALID : out std_logic);
end assign4;

architecture rtl of assign4 is
...
end rtl;
```

The test benches for Verilog and VHDL are also provided for those who prefer to use www.edaplayground.com or non-interactive environments. **However, your code will be tested using www.edaplayground.com and should match the expected outputs shown on the last page of this assignment.**

Verilog test bench:

```
// for edaplayground
// set Testbench + Design to "SystemVerilog/Verilog"
// set Tools & Simulation to "Icarus Verilog 0.9.7"
module assign4_test();

reg CK;
reg RESET;
wire DONE;
wire [4:0] OUTADDR;
wire [31:0] OUTDATA;
wire OUTVALID;
reg [15:0] COUNT;

  assign4 U0 (CK, RESET, DONE, OUTADDR, OUTDATA, OUTVALID);

initial
begin
  CK=0;
  RESET=1;
  COUNT=0;
end

always #10 CK = ~ CK;

always @(posedge CK)
begin
  RESET <= 0;
  if (!RESET && OUTVALID) $display("%04x %2x = %08x",COUNT,OUTADDR,OUTDATA);
  COUNT <= COUNT + 1;
  if (DONE) $finish;
end

endmodule
```

VHDL test bench:

```
-- for edaplayground
-- set Testbench + Design to VHDL
-- Set Top entity to "assign4_test"
-- Set Tools & Simulator to "GHDL"
-- add "--std=08" to Import Options, Make Options, and Run Options
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_textio.all;
library std;
use std.textio.all;

entity assign4_test is
end assign4_test;

architecture ARCH OF assign4_test is

component assign4
port (CK       : in  std_logic;
      RESET    : in  std_logic;
      DONE     : out std_logic;
      OUTADDR  : out std_logic_vector(4 downto 0);
      OUTDATA  : out std_logic_vector(31 downto 0);
      OUTVALID : out std_logic);
end component;

signal CK       : std_logic := '0';
signal RESET    : std_logic := '1';
signal DONE     : std_logic;
signal OUTADDR  : std_logic_vector(4 downto 0);
signal OUTDATA  : std_logic_vector(31 downto 0);
signal OUTVALID : std_logic;
signal COUNT    : std_logic_vector(15 downto 0) := (others => '0');

begin
    U0: assign4 port map ( CK => CK, RESET => RESET,
        DONE => DONE, OUTADDR => OUTADDR, OUTDATA => OUTDATA,
        OUTVALID => OUTVALID );

    process (CK)
    variable line_v : LINE;
    begin
        if (CK'event and CK = '1') then
            RESET <= '0';
            COUNT <= std_logic_vector( unsigned(COUNT) + 1 );
            if (RESET = '0' and OUTVALID = '1') then
                write(line_v, to_hstring(unsigned(COUNT)));
                write(line_v, String'(" "));
                write(line_v, to_hstring(unsigned(OUTADDR)));
                write(line_v, String'(" = "));
                write(line_v, to_hstring(unsigned(OUTDATA)));
                writeline(output, line_v);
            end if;
            if (DONE = '1') then
                report "simulation finished successfully" severity FAILURE;
            end if;
        end if;
        CK <= NOT CK AFTER 10 ns;
   end process;
end ARCH;
```

VHDL Simulation Expected Output	Verilog Simulation Expected Output
0007 00 = 00000000 000C 01 = 00000001 0011 02 = 00000002 0016 03 = 00000003 001B 04 = 00000004 0020 05 = 00000005 0025 06 = 00000006 002A 07 = 00000007 002F 08 = 00000008 0034 09 = 00000009 0039 0A = 0000000A 003E 0B = 0000000B 0043 0C = 0000000C 0048 0D = 0000000D 004D 0E = 0000000E 0052 0F = 0000000F 0059 10 = 0000000F 005E 11 = 0000000E 0063 12 = 0000000D 0068 13 = 0000000C 006D 14 = 0000000B 0072 15 = 0000000A 0077 16 = 00000009 007C 17 = 00000008 0081 18 = 00000007 0086 19 = 00000006 008B 1A = 00000005 0090 1B = 00000004 0095 1C = 00000003 009A 1D = 00000002 009F 1E = 00000001 00A4 1F = 00000000 00AB 00 = FFFF8000 00B2 01 = FFFF8001 00B9 02 = FFFF8002 00C0 03 = FFFF8003 00C7 04 = FFFF8004 00CE 05 = FFFF8005 00D5 06 = FFFF8006 00DC 07 = FFFF8007 00E3 08 = FFFF8008 00EA 09 = FFFF8009 00F1 0A = FFFF800A 00F8 0B = FFFF800B 00FF 0C = FFFF800C 0106 0D = FFFF800D 010D 0E = FFFF800E 0114 0F = FFFF800F 011B 10 = FFFF800F 0122 11 = FFFF800E 0129 12 = FFFF800D 0130 13 = FFFF800C 0137 14 = FFFF800B 013E 15 = FFFF800A 0145 16 = FFFF8009 014C 17 = FFFF8008 0153 18 = FFFF8007 015A 19 = FFFF8006 0161 1A = FFFF8005 0168 1B = FFFF8004 016F 1C = FFFF8003 0176 1D = FFFF8002 017D 1E = FFFF8001 0184 1F = FFFF8000	0007 00 = 00000000 000c 01 = 00000001 0011 02 = 00000002 0016 03 = 00000003 001b 04 = 00000004 0020 05 = 00000005 0025 06 = 00000006 002a 07 = 00000007 002f 08 = 00000008 0034 09 = 00000009 0039 0a = 0000000a 003e 0b = 0000000b 0043 0c = 0000000c 0048 0d = 0000000d 004d 0e = 0000000e 0052 0f = 0000000f 0059 10 = 0000000f 005e 11 = 0000000e 0063 12 = 0000000d 0068 13 = 0000000c 006d 14 = 0000000b 0072 15 = 0000000a 0077 16 = 00000009 007c 17 = 00000008 0081 18 = 00000007 0086 19 = 00000006 008b 1a = 00000005 0090 1b = 00000004 0095 1c = 00000003 009a 1d = 00000002 009f 1e = 00000001 00a4 1f = 00000000 00ab 00 = ffff8000 00b2 01 = ffff8001 00b9 02 = ffff8002 00c0 03 = ffff8003 00c7 04 = ffff8004 00ce 05 = ffff8005 00d5 06 = ffff8006 00dc 07 = ffff8007 00e3 08 = ffff8008 00ea 09 = ffff8009 00f1 0a = ffff800a 00f8 0b = ffff800b 00ff 0c = ffff800c 0106 0d = ffff800d 010d 0e = ffff800e 0114 0f = ffff800f 011b 10 = ffff800f 0122 11 = ffff800e 0129 12 = ffff800d 0130 13 = ffff800c 0137 14 = ffff800b 013e 15 = ffff800a 0145 16 = ffff8009 014c 17 = ffff8008 0153 18 = ffff8007 015a 19 = ffff8006 0161 1a = ffff8005 0168 1b = ffff8004 016f 1c = ffff8003 0176 1d = ffff8002 017d 1e = ffff8001 0184 1f = ffff8000;