Name: **10.01.2023**
Student ID:

CENG311 Final Exam - Fall 2022
(120 minutes)

Q.1 (20 Points) Assume we have 4-bit memory (word) addresses to load data from memory. Consider that we have two distinct systems: 1) direct-mapped cache with 4 one-word blocks, and 2) 2-way set-associative cache with the same size (initially empty). Answer the following questions:

a) Give one simple example scenario with four memory accesses supporting the following statement: Set associative caches reduce cache miss rates compared to direct-mapped caches (Direct-mapped Miss Rate > Set Ass. Miss Rate). Show the final content of the cache.

Access	Address	Hit/Miss for direct-mapped	Hit/Miss for set ass.
1	_ _ _ _	___	___
2	_ _ _ _	___	___
3	_ _ _ _	___	___
4	_ _ _ _	___	___

Direct-mapped Miss Rate = ___ Set Ass. Miss Rate = ___

b) Give one simple example scenario with four memory accesses supporting the following statement: Set associative caches have no effect on cache miss rates compared to direct-mapped caches (Direct-mapped Miss Rate = Set Ass. Miss Rate). Show the final content of the cache.

Access	Address	Hit/Miss for direct-mapped	Hit/Miss for set ass.
1	_ _ _ _	___	___
2	_ _ _ _	___	___
3	_ _ _ _	___	___
4	_ _ _ _	___	___

Direct-mapped Miss Rate = ___ Set Ass. Miss Rate = ___

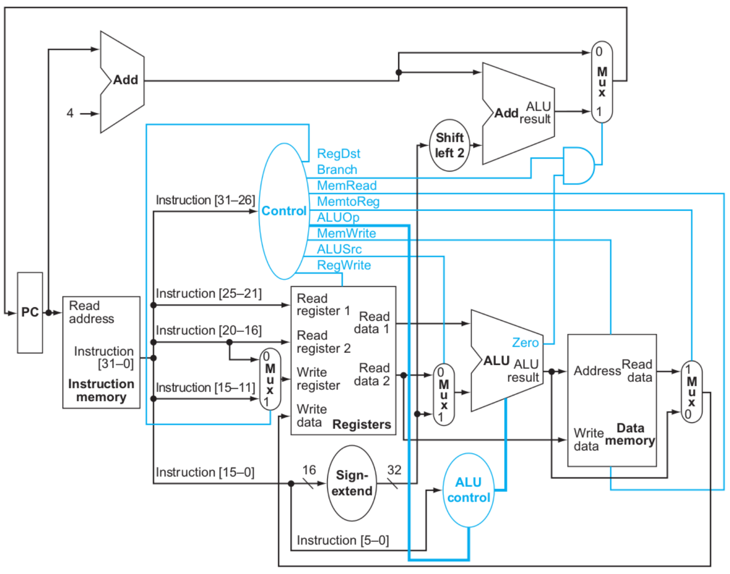
Q.2 (30 Points)

Assume that you are required to extend the single-cycle MIPS implementation so that it handles the following R-type instruction:

Load Word Register : **lwr Rd, Rt (Rs)**

The instruction reads the data from the memory address Reg[Rt]+Reg[Rs], which is the summation of the values in the register Rt and Rs.

Reg[Rd] = Mem[Reg[Rt]+Reg[Rs]]



a) Explain the required changes in the complete single-cycle datapath clearly (including changes in any component of the datapath including ALU, additional wires, muxes and control/selector signals). In case of no change in the datapath, then express it accordingly. Write the values of the control lines for the lwr instruction.

b) For the lwr instruction (with the opcode 0_{hex} and function code 6_{hex}), modify the Verilog code inside **control module** below (already given in your lab). Make sure that all the existing instructions still work correctly after your modifications.

```
module control(in, regdest, alusrc, memtoreg, regwrite, memread, memwrite, branch, aluop1,
aluop2);
        input [5:0] in;
        output regdest, alusrc, memtoreg, regwrite, memread, memwrite, branch, aluop1, aluop2;
        wire rformat,lw,sw,beq;
        assign rformat =~| in;
        assign lw = in[5]& (~in[4])&(~in[3])&(~in[2])&in[1]&in[0];
        assign sw = in[5]& (~in[4])&in[3]&(~in[2])&in[1]&in[0];
        assign beq = ~in[5]& (~in[4])&(~in[3])&in[2]&(~in[1])&(~in[0]);
        assign regdest = rformat;
        assign alusrc = lw|sw;
        assign memtoreg = lw;
        assign regwrite = rformat|lw;
        assign memread = lw;
        assign memwrite = sw;
        assign branch = beq;
        assign aluop1 = rformat;
        assign aluop2 = beq;
endmodule
```

Q.3 (20 Points) The following MIPS code could be used to increment the value of two integers that are in the main memory.

```
1: li $t0,1 # load 1 into $t0
2: lw $t1,100($t5) # load first variable
3: add $t1,$t0,$t1 # increment
4: sw $t1,100($t5) # store
5: lw $t2,104($t5) # load 2nd variable
6: add $t2,$t0,$t2 # increment
7: sw $t2,104($t5) # store
```

a) Assume that the code is executed on a 5-stage pipelined MIPS processor (with the stages IF, ID, EX, MEM and WB). By considering the instructions between **2-7**, draw the pipeline charts for the following cases. (Whenever a stage gets stalled, fill that stage with an "S". It is allowed to write and read a register at the same cycle. If data forwarding is allowed, show the forwarding.)

a.1) Assume that data forwarding is not allowed.

a.2) Assume that data forwarding is allowed.

b) Without changing what the code does, reorder the instructions to reduce the number of stalls. Draw the pipeline chart for the new order by showing the stages and forwarding (if necessary) clearly.

Q.4 (30 Points)

```
float dotprod(float x[8], float y[8])
{
        float sum = 0.0;
        int i;
        for (i = 0; i < 8; i++)
                sum += x[i] * y[i];
        return sum;
}
```

a) Does the function above have good spatial locality with respect to x and y arrays? Explain why.

b) With the following assumptions:
- Floats are 4 bytes, that x is loaded into the 32 bytes of contiguous memory starting at address 0 and that y starts immediately after x at address 32.
- A direct-mapped cache block is 16 bytes (big enough to hold four floats) and a total cache size of 32 bytes.
- The variables sum and i are stored in a CPU register and thus do not require a memory reference.

For each memory reference (first two are given below) in the code, identify it as a cache hit or miss. Show the intermediate steps and the final content of the cache.

Cache content after access to x[0]: **M**

x[0] x[1] x[2] x[3]

Cache content after access to y[0]: **M**

y[0] y[1] y[2] y[3]

c) If instead of defining x to be float x[8], we define it to be float x[12] and assume y starts immediately after x in memory. For the same memory references in the code (still accessing only 8 elements of the arrays), does it help to improve the miss rates? (**Hint:** Consider the cache-memory mappings for x[8] and x[12] cases.)