For this project you will make a pipelined LC-2200-32 without interrupt support

You should follow the five stage design which was discussed in class. You may use the ISA from your previous projects or design a new one, I recommend you use a design in which the Opcode does not just determine which instruction is being executed. Here are some ideas to help you choose your own.

You could have:

Swap RS1 for PC	bit 31
Swap RS2 for IMM	bit 30
ALU: ADD NAND SUB PASS	bit 29, 28
MEM LD, ST, PASS, to_PC	bit 27, 26
Extra for alignment	bit 25, 24

Or you could have:

Swap RS2 for IMM	bit 24
Swap RS1 for PC	bit 25
Use RD bits as SR2 register number	bit 26
Unused for alignment	bit 27
Instruction changes control flow	bit 28
Instruction uses memory	bit 29
ALU operation	bit 31, 30

RD	bits 23-20
RS1	bits 19-16
RS2	bits 15-12
IMM	bits 15-0

Notice that from this you could create opcodes which guarantee that you will be creating opcodes which have bits which differentiate everything you need to do, this is not mandatory, but you will have a much easier time if you start with a reasonable design here.

Doing so will allow you hardware to be simpler to design, and having 8-bit Opcodes, and 4 bit register numbers allows easily assembly of instructions into hex.

For example if ADD is Opcode 0x00 then we can assemble "ADD R5, R3, R8" using string concatenation to get 0x00 5 3 8 000.

Your pipeline must execute any program the same way as it would be executed on a non-pipelined version, that means you cannot have branch delay slots or delayed values, you must either stall the pipeline or forward data/predict the branch. This is because delay slots does not address the problem, it simply makes it someone else's problem.

You will be graded based on number of cycles a reference program takes to execute, with your grade being based somewhat on the speedup from a basic pipeline which does not implement data forwarding or branch prediction.
This does mean extra credit, as the baseline will be the minimum pipeline that would be acceptable, if you do the project properly with any enhancements (data forwarding, branch prediction) your speedup will be >1,
If you do the project to specification and it ends up being slower than the reference you will not lose credit.

You must have a register file with a register which is always 0 as in the previous two assignments.

You should have separate instruction and data memories, this will allow your processor to read a 32 bit instruction at the same time as loading a 32 bit value. You are free to use either byte addressable memory (as in the previous two) or word addressable memory (which is built in to logisim), I recommend word addressability because it will make everything easier.

The final stage of the project includes a program which has been written assuming you are using word addressability, you will need to change some of the offsets if you do not use a word addressable memory.

It must be a five stage pipeline as was discussed in class or you will receive no credit.

You should also have a counter which keeps track of the number of clock cycles that have passed (just use the built in counter object connected to the clock) this will allow performance comparisons to be done easily.

Required instructions are:

Arithmetic:
```
ADD     RD, RS1, RS2
ADDI    RD, RS1, IMM
NAND    RD, RS1, RS2
LDADR   RD, IMM
```

Memory:
There are two of each of these, they are just alternate ways of writing them in the assembly, they should be assembled identically.
```
SW      RS, RB, IMM
SW      RS, IMM(RB) (Same as previous instruction, with alt format)
LW      RD, RB, IMM
LW      RD, IMM(RB) (Same as previous instruction, with alt format)
```

Control flow:
```
BEQ     RS1, RS2, IMM
HALT
```
(this should disable clock ticks entirely, this can be accomplished with an AND gate between the clock and the datapath)
```
JALR    RT, RL
```
(you are still free to decide the order the steps happen in, but I recommend PC<-RT, and then RL<-old_PC, this will likely be easier to implement)

Once you have made your first draft of your ISA you should start on the processor, try to make it one instruction at a time, and after each change verify that the new instruction works, as well as all of the previously implemented ones, if you realize something would be easier if the ISA were different go back and change your ISA.

Once you have tested each instruction individually you should assemble and run benchmark.s, I have calculated the offsets for you, so you just need to fill in the bits for your ISA.

The base line execution cycles will be posted later.

Submission

- Pipeline circuit
- ISA
- Assembled benchmark
- The grade you think you deserve (may or may not be used)
- Optional: Explanation of any of the previous items.