1.) Convert the following assembly language program to machine code. The code begins at address 100\text{10}. Your answer must be in binary in the spaces provided (CMP at top, ADDI at bottom).

```
CMP X1, X2  SUBL5 X3, X1, X2
B.LT ELSEIF +2
ADDI X1, X3, #5
ELSEIF:
```

Second copy if you need it – but if you use it, indicate which to grade!!!:
2.) We are given a program that executes 6 adds, 4 branches, 6 loads, 2 store, and 2 subtracts. Our compiler can fill \( \frac{1}{2} \) of the delay slots in this code. If the single-cycle has a clock cycle of 10\( \text{ns} \), and the pipelined has a clock cycle of 3\( \text{ns} \), how much time will each machine take to complete the program? Assume that all machines are built like those discussed in class.

Single-cycle takes \( \frac{200}{\text{ns}} \).

Pipelined takes \( \frac{87}{\text{ns}} \).

**Single-cycle**

\[
\text{CPI} = 1.0
\]

\[
\#\text{instr} = 6 + 4 + 6 + 2 + 2 = 20
\]

\[
20 \times 10\text{ns} = 200\text{ns}
\]

**Pipeline**

\[
\text{cycles} = \text{issue + drain}
\]

\[
= (6 \times 1 + 4 \times 1.5 + 6 \times 1.5 + 2 \times 1 + 2 \times 1) + (6 \times 3 - 1)
\]

\[
= 6 + 6 + 9 + 2 + 2 + 4
\]

\[
= 29
\]

\[
\text{time} = \#\text{cycles} \times \text{clock period}
\]

\[
29 \times 3 = 87\text{ns}
\]
3.) Register X0 has the address of an array of integers in memory, while X1 has the length (and is at least 5, perhaps longer). In assembly, write a program that will set X15 to the maximum value in that array. Your program should be as simple and efficient as possible. It should be written for a normal (non-pipelined) CPU. You can modify the value in any of the registers, including X0 and X1.

```assembly
MAX = 0;
 For(i = 0; i < length; i++)
  If(MAX < U[i])
    MAX = U[i];

MAX = 0;
do {*
  If(MAX < U[i])
    MAX = U[i];
  V++;
  length--;
  Until: (length == 0);
ADDI X15, X31, #0 / MAX = 0

loop:
LDUR X2, [X0, #0] / get U[i]
CMP X15, X2
BGE no_update / don't update
ADDI X15, X2, #0 / MAX = U[i]

no_update:
ADDI X0, X0, #8 / V++
SUBI X1, X1, #1 / length--
CBNZ X1, loop
```
4.) We wish to add the instruction “BNEZ_R” – Branch to register if not equal to 0 – to the single-cycle processor from class. It is like a BR instruction, except it will NOT branch if the address is 0. The RTL for this instruction is given below:

Instruction = Mem[PC];
Cond = (true if Reg(Rd)! = 0);
if (Cond) PC = Reg[rd];
else PC = PC + 4;

Show any modifications to the datapath necessary to accommodate this instruction. You should try to modify the datapath as little as possible. Then, on the next page list the control setting for the BNEZ_R instruction (only).
Control settings:

<table>
<thead>
<tr>
<th></th>
<th>STU-PC Control Setting:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg2Loc</td>
<td>0</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>0</td>
</tr>
<tr>
<td>MemToReg</td>
<td>X</td>
</tr>
<tr>
<td>RegWrite</td>
<td>0</td>
</tr>
<tr>
<td>MemWrite</td>
<td>0</td>
</tr>
<tr>
<td>BrTaken</td>
<td>if(A(zero)) 0 else 2</td>
</tr>
<tr>
<td>UncondBr</td>
<td>X</td>
</tr>
<tr>
<td>ALUOp</td>
<td>B==0?</td>
</tr>
</tbody>
</table>
5.) Create a single-cycle processor that can do “SUB” and “BR” only. Draw the **datapath** for this machine (including the internals of the instruction fetch unit). Note that your machine should be as simple as possible.

**Datapath:**

\[
\text{Instruction} = \text{Mem[PC]} \quad \text{BR}
\]

\[
\text{Sub: } \text{Reg[RD]} = \text{Reg[EN]} - \text{Reg[EN]} / \text{PC} = \text{Reg[RD]}
\]

\[
\text{PC} = \text{PC} + 4
\]