Midterm Histogram

Average: 18.7  3/4 >= 14  1/2 >= 21  1/4 >= 24
1.) Convert the following assembly language program to machine code. The code begins at address 100\textsubscript{10}. Your answer must be in binary in the spaces provided (CMP at top, ADDI at bottom).

```assembly
CMP X1, X2
SUB S X3!, X1, X2
B.LT ELSEIF +2
ADDI X1, X3, #5
ELSEIF:
```

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```

Second copy if you need it – but if you use it, indicate which to grade!!!:

```assembly
```

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
```
2.) We are given a program that executes 6 adds, 4 branches, 6 loads, 2 store, and 2 subtractions.
Our compiler can fill $\frac{1}{2}$ the delay slots in this code. If the single-cycle has a clock cycle of 10ns, and the pipelined has a clock cycle of 3ns, 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 \( \frac{200}{1.6} \) ns.

Pipelined takes \( \frac{87}{3} \) ns.

Single-cycle
CPI = 1.6

$\# \text{instructions} = 6 + 4 + 6 + 2 + 2 = 20$

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

Pipeline

\( \# \text{cycles} = \text{issue} + \text{drain} \)

\( = (6 \times 1 + 4 \times 1.5 + 6 \times 1.5 + 2 \times 1 + 2 \times 1) + (5 \times 3 - 1) \)

\( = 6 + 6 + 9 + 2 + 2 + 4 \)

\( = 29 \text{ cycles} \)

\( \text{time} = \# \text{cycles} \times \text{clock period} \)

\( 29 \times 3 \text{ ns} \)

\( = 87 \text{ ns} \)
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.

```
MAX=0;
for(i=0; i<length; i++)
if(max < X[i])
   max = X[i];

MAX = 0;
do { 
   if(max < X[i]) max = X[i];
   V++;
   length--;
} until (length == 0);

ADDJ X15, X31, #0  // MAX = 0
LOOP:
LDUR X2, [X0, #0]  // get *V
CMP X15, X2
B.GE NO-UPDATE  // don't update
if(max>V)
   ADDJ X15, X2, #0
   @NO-UPDATE:
   ADDJ X9, 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>STU-PC Control Setting:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Reg2Loc</td>
<td>O</td>
</tr>
<tr>
<td>ALUSrc</td>
<td>O</td>
</tr>
<tr>
<td>MemToReg</td>
<td>X</td>
</tr>
<tr>
<td>RegWrite</td>
<td>O</td>
</tr>
<tr>
<td>MemWrite</td>
<td>O</td>
</tr>
<tr>
<td>BrTaken</td>
<td>if(zer) 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:

Instruction = Mem[PC]  BR

Sub: Reg[Rd] = Reg[Em] - Reg[En]  /  PC = Reg[Rd]

PC = PC + 4

Datapath diagram:

[Diagram showing the datapath with PC, instruction fetch, registers, and branch logic]
Instruction-Level Parallelism & Advanced Architectures

Key to pipelining was dealing with hazards

Advanced processors require significant hazard avoidance/flexibility

ILP = Instruction-Level Parallelism
Why ILP

Advanced processors optimize two factors:

Reduce clock period by heavy pipelining
  Greater pipelining means more hazards, delay slots

Reduce CPI
  What if we want a CPI < 1.0?
ILP Example

Source code:

1: ADDI X0, X0, #15
2: SUB X2, X1, X0
3: EORI X3, X0, #15
4: ORR X4, X3, X0

Constraint graph:

1: ADDI
2: SUB
3: EORI
4: ORR