Problem 1(a) [8 pts]: Binary tree for expression $a + 16 \times (b + 8 \times (c + 4 \times (d + e)))$ is shown below:

```
+     +     +     +     +
a     *     16    b     *
  *     +     8     c     *
  4     +     d     e
```

Problem 1(b) [8 pts]: Post order expression is obtained by traversing the above tree in the post order (left, right, root) fashion. Following is the post order expression:

```
a 16 b 8 c 4 d e + * + * * +
```

Problem 1(c) [8 pts]: Pre order expression is obtained by traversing the above tree in the pre order (root, left, right) fashion. Following is the post order expression:

```
+ a * 16 + b * 8 + c * 4 + d e
```

Problem 1(d) [10 pts]: Assuming a, b, c and d are the memory addresses (or labels) of the numbers a, b, c, d and e respectively, the CodeMill stack program is shown below. We assume that numbers are small enough such that their products do not exceed 16 bits. After the program is finished, the top of the stack contains the result.

```
PUS e;
PUS d;
ADS;
SLS;
SLS;
PUS c;
ADS;
SLS;
```
Problem 1(f) [8 pts]: We verify that the above program leaves 5025 at the top of the stack, which is the value of the expression, for the given values of a, b, c, d and e.

Problem 1(e) [8 pts]:
The stack after each instruction in the stack program is shown below. We show the stack as growing from left to right, assuming it was empty in the beginning. Suppose memory locations a, b, c, d and e contain decimal numbers 1, 2, 3, 4 and 5 respectively.

<table>
<thead>
<tr>
<th>Instruction executed</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUS e;</td>
<td>5</td>
</tr>
<tr>
<td>PUS d;</td>
<td>5</td>
</tr>
<tr>
<td>ADS;</td>
<td>9</td>
</tr>
<tr>
<td>SLS;</td>
<td>18</td>
</tr>
<tr>
<td>SLS;</td>
<td>36</td>
</tr>
<tr>
<td>PUS c;</td>
<td>36</td>
</tr>
<tr>
<td>ADS;</td>
<td>39</td>
</tr>
<tr>
<td>SLS;</td>
<td>78</td>
</tr>
<tr>
<td>SLS;</td>
<td>156</td>
</tr>
<tr>
<td>SLS;</td>
<td>312</td>
</tr>
<tr>
<td>PUS b;</td>
<td>312</td>
</tr>
<tr>
<td>ADS;</td>
<td>314</td>
</tr>
<tr>
<td>SLS;</td>
<td>628</td>
</tr>
<tr>
<td>SLS;</td>
<td>1256</td>
</tr>
<tr>
<td>SLS;</td>
<td>2512</td>
</tr>
<tr>
<td>SLS;</td>
<td>5024</td>
</tr>
<tr>
<td>PUS a</td>
<td>5024</td>
</tr>
<tr>
<td>ADS</td>
<td>5025</td>
</tr>
</tbody>
</table>
Problem 2(a) [10 pts]: We use the registers as follows: R0 as i, R1 as x, R2 as y, R3 as p, R4 as q and R5 as n.

Problem 2(b) [10 pts]: We need following instructions: LDI, SHL, ADD, MOV, JCD, HLT and INC instructions.

```
LDI R0, 1;          // i = 1
LDI R1, 1;          // x = 1
LDI R2, 2;          // y = 2
LDI R4, 4;          // q = 4
LDI R5, 3;          // n = 3
SHL R2;             // As we need 2*y and 2*q, we make y = 2*y and q = 2*q
SHL R4;             // We will restore them at the end
loop:
    JCD R0, >=, R5, out; // if i >= n, go to out
    SHL R1;              // x = 2*x
    ADD R1, R2,         // x = x + 2*y
    MOV R3, R1;         // p = x
    SHL R3;             // p = 2*p
    ADD R3, R4;         // p = p + 2*q
    INC R0;             // i++
    JMP loop;
out:
    SHR R2;             // restore y
    SHR R4;             // restore q
    HLT;                // done
```

Problem 2(c) [10 pts]: The CodeMill code changes as:

```
LDI R0, 1;          // i = 1
LDI R1, 1;          // x = 1
LDI R2, 2;          // y = 2
LDI R4, 4;          // q = 4
LDI R5, 3;          // n = 3
SHL R2;             // As we need 2*y and 2*q, we make y = 2*y and q = 2*q
SHL R4;             // We will restore them at the end
loop:
    SHL R1;              // x = 2*x
    ADD R1, R2,         // x = x + 2*y
    MOV R3, R1;         // p = x
    SHL R3;             // p = 2*p
    ADD R3, R4;         // p = p + 2*q
    INC R0;             // i++
    JCD R0, <, R5, loop; // Go to loop if i < n
out:
    SHR R2;             // restore y
    SHR R4;             // restore q
    HLT;                // done
```

Problem 2(d) [10 pts]: The two programs are not equivalent because in the second case the loop will always run at least once whereas in the first case the loop may not run at all if i > n in the beginning. Thus if n = 0, the first program will leave all the variables unchanged. But, the second program will run the loop once so the following variables would be changed as: i = 2, x = 6 and p = 10.